Browser-Independent JavaScript Secure Information Flow with Modular Proofs

José Fragoso Santos and Tamara Rezk

INRIA Sophia Antipolis-Méditerranée
firstname.lastname@inria.fr

Abstract. We present a JavaScript compiler that instruments code with information flow constraints. We prove that the compiler generates secure code w.r.t a noninterference property. We provide libraries that represent information flow specifications of DOM and other customary external interfaces. Using the compiler, we give realistic examples of JavaScript code and show how to encode common security flows in the web such as cookie stealing. The compiler is the result of inlining a new information flow monitor for JavaScript that uses a labeling that makes code instrumentation more efficient.

1 Introduction

Web application security violations represent a major risk [9]. Violations are mainly due to the permissive JavaScript semantics and integration of code coming from different origins, such as advertisement scripts. As an example of such a risk Jang et al. [19] study, using an enhanced browser, 50000 JavaScript-based websites and show that several sites, including Alexa global top-100, present privacy violating information flows vulnerabilities. This critical security situation has led to an increasing interest in sound and practical mechanisms to enforce JavaScript secure information flow.

Recent works on JavaScript security focus on dynamic mechanisms such as monitoring [17], secure multi-execution [12], and multi-facets [3]. These mechanisms are browser-dependent. Either browser code or the JavaScript machinery must be modified or a plugin must be installed in order for the mechanism to be applicable.

In this work we focus on a *browser-independent* information flow control mechanism because it can immediately be adopted in practice, JavaScript rewriting techniques are still subject to attacks [25, 29], and previous works dealing rigorously with the subject [32, 10, 27, 28] study simple languages limited to static variable references. We study a JavaScript subset without this restriction and extend it to external interfaces including part of the DOM API [18]. We propose how to extend the proofs of the security theorem of the compiler in a modular way by defining a lemma for each new external interface that is handled. In order to instrument JavaScript programs in an efficient manner, we consider a new JavaScript information flow monitor that departs from previous work [17] in what concerns dynamic labeling. We show that our labeling is more suitable for inlining information flow since it allows for less constraints enforcements at run-time and more efficient techniques to handle labels in compiled code. Finally, we discuss how the new monitor is compatible with standard static labeling and can be used to extract an information flow type system.

In summary, our contributions are:

- A new labeling semantics for JavaScript and noninterference property associated with it. The new labeling is suitable for efficient code instrumentation and compatible with static analyses.
- A sound JavaScript monitor for information flow with simplified constraints, compared to previous works.
- A compiler specification that generates information flow instrumented JavaScript code. A proof of correctness and security regarding termination-insensitive noninterference for a JavaScript subset not restricted to static variable references. We extend the instrumentation to external JavaScript interfaces, including DOM interfaces, and present a general method to obtain modular proofs of security for external interfaces.
- A prototype of the compiler available online as a web application [1] together with case studies encoding common web vulnerabilities, such as cookie stealing.

Related work There are many mechanisms to prevent attacks coming from JavaScript code. For example the Facebook Javascript Subset (FBJS) [14] was intended to prevent user-written gadgets to attack trusted code but it did not succeed in its goals as shown in recent work by Maffeis and Taly [26]. Google Caja [30] is similar to FBJS. A recent proof (2010) [25] shows that a subset of Caja has an isolation property called capability safety, a property less expressive than information flow properties. Yahoo ADsafe [11] statically validates JavaScript programs for security but as shown in Krishnamurthi et al. (2011) [29] its implementation reveals several bugs and other weaknesses. In contrast to other more ad-hoc isolation mechanisms, there are elegant techniques to define information flow mechanisms that can be proved sound and effectively help to prevent confidentiality and integrity attacks. In web applications there is a need for dynamic mechanisms (see LeGuernic thesis for an excellent survey on the subject [20]) for information flow control due to the dynamic nature of the JavaScript language. Austin and Flanagan [3] and Hedin and Sabelfeld [17] study runtime monitors for noninterference in JavaScript-like languages. The monitor we propose is substantially simpler than that of Hedin and Sabelfeld since labeling properties allows to have less security labels and simpler constraints. In particular, labeling properties makes it unnecessary to have an existence security level. Our monitor allows for more efficient code instrumentation as explained in the following section. Bohannon et al. [7] propose reactive noninterference, a noninterference property for reactive programs such as web scripts to replace the Same Origin Policy in browsers. Bielova et al. [6] later propose a an enforcement mechanism for reactive non-interference based on secure multi-execution [12] and implement it in Featherweight Firefox browser model. Venkatakrishnan [32] is the first to present a hybrid technique that relies on runtime information-flow tracking augmented with static analysis to reason about implicit flows that arise due to unexecuted paths in a program. Chudnov and Naumann [10] propose monitor inlining of an information flow-sensitive monitor for a simple imperative language. In parallel, Magazinius et al. [27] propose monitor inlining for an imperative language with *eval* with the novel feature of performing inlining on the fly to handle *eval*. In a follow up paper, Magazinius et al. [28] present automatic code rewriting for [27].

2 Interplay between Labeling and Instrumentation

Information flow policies are specified by labeling with security levels the observable resources of a program. In general, to check conformance with an information flow security property, labels from a security lattice are needed on intermediate interfaces such as properties and local variables. In the following, we assume given a lattice $\mathcal{L}$ of security levels. In the examples, we use $\mathcal{L} = \{H, L\}$ with $L \leq H$. Labeling variables or other resources in a program is frequently done statically via a mapping from statically referred resources to security labels. In JavaScript, resources are dynamically created which makes it infeasible to refer to them precisely at the static level as illustrated by the following JavaScript program:

$$x = \{\}; x[f()] = 1$$

Here f is a function that returns a string s obtained by concatenation of arbitrary user input. In this program, a reference to an object o is stored in variable x and then s is added to o as a property. Clearly, property s cannot be labeled before runtime since its name is not known. Hence, instead of considering static labelings, we consider dynamic labelings that map each property in every object to a security level. Existing works [17] opt for dynamically store the security level on the value of the property of an object. Consider the following example where as many different properties as the value of n are dynamically created:

```
1 while (n) {
2    o[n] = n;
3    n--}
```

Listing 1.1. Example 1 - Transformed

If security labels are assigned to values as in [17], we obtain the following code instrumentation that is semantically equivalent to the original program (if the original program is secure):

```
1 _pc = _lat.lub(_pc, n.level);
2 while(n.value) {
3    _lat.leq(_lat.lub(o.level, n.level, _pc), o.value.struct);
4    o[n.value] = {value: n.value, level: _lat.lub(o.level, n.level, _pc)};
5    _lat.leq(_pc, n.level); n.value--}
```

Listing 1.2. Example 1 - Transformed

Note that in line 4 the program creates a new object for each value of n, that acts as an "envelope" that contains the value of the new property and its security label (obtained by computation of a least upper bound $\_lat.lub(o.level, n.level, \_pc)$). In our approach, we rather associate the security label directly to the property of the object. Naturally, since every variable is semantically modeled as a property of a given scope object, dynamic labelings also map resources to security levels, as in the static case. Variables and object properties are thus treated uniformly. Instrumentation of the same original program using our approach gives:

```
1 _pc = _lat.lub(_pc, _lab['n']);
2 while (n) {
3    _lat.leq(_lat.lub(_lab['o'], _lab['n'], _pc), o.struct);
4    o[n] = n;
5    o.lab[n] = _lat.lub(_lab['o'], _lab['n'], _pc);
6    _lat.leq(_pc, _lab['n']);
7    n--}
```

Listing 1.3. Example 1 - Transformed

The resulting program is not creating new objects but only a new property in object o, modified at line 5, that corresponds to the labeling of the properties added to o with their corresponding security level.

3 JavaScript Labeling

We present JavaScript semantics extended with the computation of a dynamic labeling. The syntax of the JavasScript subset considered is given in Figure 1.

Objects and Memory. Objects, taken from $\mathcal{O}bj$, are modeled as partial functions from $\mathcal{S}tr$ to $\mathcal{V}al$. The strings in the domain of anObj object are deemed its properties. Some properties cannot be changed by the program, for clarity those properties are prefixed with an "@". In order to create a new object, the programmer must use the keyword new. References can be viewed as pointers to objects, in the sense that every expressions that creates an object in memory does not yield the object itself, but a free reference that points to it (that it is nondeterministically chosen). Since functions are executed in the environment on which they are defined, their internal representation must include a reference to the scope object that was active when the corresponding function literal was evaluated. Thus, at the semantic level, functions are represented as *function objects*. A function object has the following two properties:

- @*code*: stores the internal representation of the function literal;
- @$fscope$: stores a reference to the scope object that was active when the corresponding function literal was evaluated.

Consequently, the evaluation of a function literal triggers the creation of a new function object and yields the reference that points to it. A memory is a mapping from references to objects and the set of all memories is denoted by $\mathcal{M}em$. In the following, we assume that memories include references to two special objects:

- The *global object* is the object that is at the bottom of every scope chain. Formally: $[@this \mapsto \#global, @proto \mapsto null, @scope \mapsto null] \leq global$ (where we use $\leq$ for function extension). For every memory, we use $\#global$ as the reference to the global object.
- The *prototype object* is the object that is at the bottom of every prototype chain. Formally: $protObj = [@proto \mapsto null]$.

$e ::= x$		identifier
	v	primitive values
	$this$	this keyword
	$\{m_1 : e_1, \ldots, m_n : e_n\}$	object literal
	$\mathsf{function}(x)\{s\}$	function literal
	$e_1[e_2]$	member selector
	$x = e$	variable assignment
	$e_1[e_2] = e_3$	property assignment
	$\mathsf{new}\ e_1(e_2)$	constructor call
	$e_1(e_2)$	function call
	$e_1[e_2](e_3)$	method call
	$(e_1\ \mathsf{op}_2\ e_2)$	binary operations
$s ::= s_1; s_2$		sequence of statements
	$\mathsf{var}\ x$	variable declaration
	$\mathsf{if}(e)\{s_1\}\mathsf{else}\{s_2\}$	conditional
	$\mathsf{while}(e)\{s\}$	while loop
	$\mathsf{upgVar}(x, \sigma)$	extended syntax: variable upgrade
	$\mathsf{upgProp}(o, p, \sigma)$	extended syntax: property upgrade
	$\mathsf{upgStruct}(o, \sigma)$	extended syntax: property upgrade
$v ::=$	n	number
	m	string
	b	boolean $b \in \{true, false\}$
	$null$	null
	$undef$	undefined
	r	reference
	$\lambda x.s$	function runtime value

Fig. 1. JavaScript Core Syntax and Extension for Security Upgrades

Labelings. A security labeling is a function $\Gamma : \ (\mathcal{R}ef \mapsto \mathcal{S}tr) \cup \mathcal{R}ef \mapsto \mathcal{L}$ that maps references and properties from $\mathcal{S}tr$ to security labels. The set of security labeling is denoted Additionally, labelings map references of objects to their corresponding structure security level to keep track of dynamically added properties. To illustrate this point, consider the following program from [17] where object o does not initially have any properties:

$$\mathsf{if}(h)\ \{o[\text{"}p\text{"}] = 0\}; l = o[\text{"}p\text{"}] \tag{1}$$

Intuitively, the *structure* of object o depends on high variable h, because property "p" is created in a context that depends on h. Therefore, variable l depends on a high variable as well: its final value will be 0 if h is true or $undef$ otherwise. The reading effect is an upper bound on the levels of all the resources that are read during the evaluation of s. The computed dynamic security labeling must correctly reflect the actual dependencies entailed by the execution of the program. A dynamic security labeling establishes what resources are visible at each security level. Hence, after the execution of the program the level of each resource must correspond to the upper bound of the levels of the resources on which it depends. Although dynamic security labelings are constructed at runtime, the programmer must be able to specify at the static level which security levels are to be assigned to the resources that are created during execution. To this end, we extend the JavaScript syntax with three additional constructs:

- $\mathsf{upgVar}(x, \sigma)$ upgrades the level of variable x to the least upper bound between its current level and σ.
- $\mathsf{upgProp}(o, p, \sigma)$ upgrades the level of property p of the object referenced by variable o to the least upper bound between its current level and σ.
- $\mathsf{upgStruct}(o, \sigma)$ upgrades the structure security level of the object referred by o to the least upper bound between its current level and σ.

Scope Look-up. Scope is modelled *via scope objects*. A scope object is an object that maps the formal argument of the function that is currently executing to its current value. The only way to establish a new scope is through a function call, method call or constructor call. Since JavaScript is syntactically scoped, functions are executed in the scope on which they are defined. Therefore, scope objects have a special property @*scope* which stores the reference of the scope object that was active when the function that is being executed was stored in memory. The sequence of scope objects that can be accessed from a given scope object through the respective @*scope* properties is deemed a *scope chain*. In order to determine the value associated with a given variable, one has to inspect all the objects in the scope chain starting from the *active* scope object (that is, the one at the top of the scope chain). This behaviour is modeled by the semantic relation $\mathcal{R}_{Scope}$ which is presented in Definition 1. The relation $\mathcal{R}_{Scope}$ relates triples in $\mathcal{M}em \times \mathcal{R}ef \times \mathcal{S}tr$ with elements of $\mathcal{R}ef$. Informally, $\langle \mu, r_1, m\rangle\ \mathcal{R}_{Scope}\ r_2$ means that reference r_2 points to the scope object that is closest to the one pointed by r_1 in the corresponding scope chain (that is stored in memory μ) and which defines a binding for m.

Definition 1 ($\mathcal{R}_{Scope}$).

$$\textsc{Null}\quad \langle \mu, null, m\rangle\ \mathcal{R}_{Scope}\ null \qquad \textsc{Base}\quad \frac{m \in \mathsf{dom}(\mu(r))}{\langle \mu, r, m\rangle\ \mathcal{R}_{Scope}\ r} \qquad \textsc{Look-up}\quad \frac{m \notin \mathsf{dom}(\mu(r)) \qquad \langle \mu, \mu(r, @scope), m\rangle\ \mathcal{R}_{Scope}\ r'}{\langle \mu, r, m\rangle\ \mathcal{R}_{Scope}\ r'}$$

Prototype Look-up. In JavaScript every object has a prototype and stores a reference to it in an internal property here denoted by @*proto*. When a program tries to access a property m of an object o, JavaScript first checks if o has a property named m (that is, if $m \in o$). If it does not, JavaScript checks if the prototype of object o has a property named m and so forth. The sequence of objects that can be accessed from a given object through the respective @*proto* properties is deemed a *prototype chain*. The prototype chain look-up process is emulated by the semantic relation $\mathcal{R}_{Proto}$, presented in Definition 2, that relates tuples in $\mathcal{M}em \times \mathcal{R}ef \times \mathcal{S}tr \times \mathcal{L}ab$ with pairs in $\mathcal{R}ef \times \mathcal{L}$. If $\langle \mu, r, m, \Gamma\rangle\ \mathcal{R}_{Proto}\ \langle r', \sigma\rangle$, then r' is the closest reference to r in its corresponding prototype chain that defines a binding for m. Additionally, σ corresponds to the security level of the look-up process, which takes into consideration the whole prototype chain that is inspected and not only the security level associated with the searched property.

Definition 2 ($\mathcal{R}_{Proto}$).

$$\textsc{Null}\quad \langle \mu, null, m, \Gamma\rangle\ \mathcal{R}_{Proto}\ \langle null, \bot\rangle \qquad \textsc{Base}\quad \frac{m \in \mathsf{dom}(\mu(r))}{\langle \mu, r, m, \Gamma\rangle\ \mathcal{R}_{Proto}\ \langle r, \Gamma(r, m)\rangle}$$

$$\textsc{Look-up}\quad \frac{m \notin \mathsf{dom}(\mu(r)) \qquad r' \doteq \mu(r, @proto) \qquad \langle \mu, r', m, \Gamma\rangle\ \mathcal{R}_{Proto}\ \langle r'', \sigma\rangle \qquad \sigma' \doteq \Gamma(r, @proto) \sqcup \Gamma(r) \sqcup \sigma}{\langle \mu, r, m, \Gamma\rangle\ \mathcal{R}_{Proto}\ \langle r'', \sigma'\rangle}$$

Semantics Relation. Figures 2, 3, and 4 present the big-step semantics relation that computes the security labeling. The relation has the following form:

$$r_s, pc \vdash \langle \mu, s, \Gamma\rangle \Downarrow \langle \mu', v, \Gamma', \sigma\rangle$$

where r_s corresponds to the current scope object (also called *active* scope object), μ and μ' denote the original and the final memories respectively, s denotes the program to be executed and v the value to which the program evaluates, *pc* the security level of the program counter, Γ and Γ' the original and final labelings, and σ the *reading effect* of s. We use $f[d \mapsto v]$ to denote a function f' that coincides with f everywhere except in d of the domain where $f'(d) = v$. We use $\doteq$ for equality in the semantics.

Legal labelings. Given a memory μ, not all labelings Γ are legal labelings for μ. Particularly, we only consider those that assign a security level to every property of every object in the corresponding memory. Furthermore, a labeling only assigns security levels to existing properties of existing objects. Definition 3 formalizes the notion of *well labeled memory*, whereas Definition 4 extends this notion to big step transitions.

PROPERTY UPDATE

$$\frac{r_s, pc \vdash \langle \mu, e_1, \Gamma \rangle \Downarrow \langle \mu_1, r_1, \Gamma_1, \sigma_1 \rangle \qquad r_s, pc \vdash \langle \mu_1, e_2, \Gamma_1 \rangle \Downarrow \langle \mu_2, m_2, \Gamma_2, \sigma_2 \rangle \qquad r_s, pc \vdash \langle \mu_2, e_3, \Gamma_2 \rangle \Downarrow \langle \mu_3, v_3, \Gamma_3, \sigma_3 \rangle \qquad m_2 \in \mu_3(r_1) \qquad \Gamma_4 \doteq \Gamma_3\,[(r_1, m_2) \mapsto \sigma_1 \sqcup \sigma_2 \sqcup \sigma_3] \qquad \mu_4 \doteq \mu_3\,[(r_1, m_2) \mapsto v_3]}{r_s, pc \vdash \langle \mu, e_1[e_2] = e_3, \Gamma \rangle \Downarrow \langle \mu_4, v_3, \Gamma_4, \sigma_3 \rangle}$$

PROPERTY CREATION

$$\frac{r_s, pc \vdash \langle \mu, e_1, \Gamma \rangle \Downarrow \langle \mu_1, r_1, \Gamma_1, \sigma_1 \rangle \qquad r_s, pc \vdash \langle \mu_1, e_2, \Gamma_1 \rangle \Downarrow \langle \mu_2, m_2, \Gamma_2, \sigma_2 \rangle \qquad r_s, pc \vdash \langle \mu_2, e_3, \Gamma_2 \rangle \Downarrow \langle \mu_3, v_3, \Gamma_3, \sigma_3 \rangle \qquad m_2 \notin \mu_3(r_1) \qquad \mu_4 \doteq \mu_3\,[(r_1, m_2) \mapsto v_3] \qquad \Gamma_4 \doteq \Gamma_3\,[(r_1, m_2) \mapsto \sigma_1 \sqcup \sigma_2 \sqcup \sigma_3, r_1 \mapsto \Gamma_3(r_1) \sqcup \sigma_1 \sqcup \sigma_2]}{r_s, pc \vdash \langle \mu, e_1[e_2] = e_3, \Gamma \rangle \Downarrow \langle \mu_4, v_3, \Gamma_4, \sigma_3 \rangle}$$

PROPERTY LOOKUP

$$\frac{r_s, pc \vdash \langle \mu, e_1, \Gamma \rangle \Downarrow \langle \mu_1, r_1, \Gamma_1, \sigma_1 \rangle \qquad r_s, pc \vdash \langle \mu_1, e_2, \Gamma_1 \rangle \Downarrow \langle \mu_2, m_2, \Gamma_2, \sigma_2 \rangle \qquad \langle \mu_2, r_1, m_2, \Gamma_2 \rangle\ \mathcal{R}_{Proto}\ \langle r', \sigma' \rangle}{r_s, pc \vdash \langle \mu, e_1[e_2], \Gamma \rangle \Downarrow \langle \mu_2, \mu_2(r', m_2), \Gamma_2, \sigma_1 \sqcup \sigma_2 \sqcup \sigma' \rangle}$$

VARIABLE

$$\frac{\langle \mu, r_s, x \rangle\ \mathcal{R}_{Scope}\ r_x \qquad r_x \neq null}{r_s, pc \vdash \langle \mu, x, \Gamma \rangle \Downarrow \langle \mu, \mu(r_x, x), \Gamma, \Gamma(r_x, x) \sqcup pc \rangle}$$

ASSIGNMENT - 1

$$\frac{r_s, pc \vdash \langle \mu, e, \Gamma \rangle \Downarrow \langle \mu_1, v_1, \Gamma_1, \sigma_1 \rangle \qquad \langle \mu, r_s, x \rangle\ \mathcal{R}_{Scope}\ r_x \qquad r_x \neq null \qquad \Gamma_2 \doteq \Gamma_1\,[(r_x, x) \mapsto \sigma_1] \qquad \mu_2 \doteq \mu_1\,[(r_x, x) \mapsto v_1]}{r_s, pc \vdash \langle \mu, x = e, \Gamma \rangle \Downarrow \langle \mu_2, v_1, \Gamma_2, \sigma_1 \rangle}$$

ASSIGNMENT - 2

$$\frac{r_s, pc \vdash \langle \mu, e, \Gamma \rangle \Downarrow \langle \mu_1, v_1, \Gamma_1, \sigma_1 \rangle \qquad \langle \mu, r_s, x \rangle\ \mathcal{R}_{Scope}\ null \qquad \Gamma_2 \doteq \Gamma_1\,[(\#global, x) \mapsto \sigma_1] \qquad \mu_2 \doteq \mu_1\,[(\#global, x) \mapsto v_1]}{r_s, pc \vdash \langle \mu, x = e, \Gamma \rangle \Downarrow \langle \mu_2, v_1, \Gamma_2, \sigma_1 \rangle}$$

FUNCTION LITERAL

$$\frac{o_f \doteq [@fscope \mapsto r_s, @code \mapsto \lambda x.s, prototype \mapsto \#objProt] \qquad r_f \notin \mathsf{dom}(\mu) \qquad \mu' \doteq \mu\,[r_f \mapsto o_f] \qquad \Gamma' \doteq \Gamma\,[(r_f, @fscope) \mapsto pc, (r_f, @code) \mapsto pc, (r_f, prototype) \mapsto pc]}{r_s, pc \vdash \langle \mu, \mathsf{function}(x)\{s\}, \Gamma \rangle \Downarrow \langle \mu', r_f, \Gamma', pc \rangle}$$

Fig. 2. Instrumented Semantics of JavaScript Expressions - 1

FUNCTION CALL

$$\frac{\begin{array}{c} r_s, pc \vdash \langle \mu, e_0, \Gamma\rangle \Downarrow \langle \mu_0, r_0, \Gamma_0, \sigma_0\rangle \qquad r_s, pc \vdash \langle \mu_0, e_1, \Gamma_0\rangle \Downarrow \langle \mu_1, v_1, \Gamma_1, \sigma_1\rangle \\ pc' \doteq \Gamma_1(r_0, @fscope) \qquad r'_s \doteq \mu_1(r_0, @fscope) \qquad \lambda x.s \doteq \mu_1(r_0, @code) \\ r''_s \notin \mathsf{dom}(\mu_1) \qquad \Gamma' \doteq \Gamma_1\,[(r''_s, x) \mapsto \sigma_1, (r''_s, @scope) \mapsto pc' \sqcup \sigma_0, r''_s \mapsto pc' \sqcup \sigma_0] \\ o_s \doteq [x \mapsto v_1, @scope \mapsto r'_s] \qquad \mu' \doteq \mu_1\,[r''_s \mapsto o_s] \qquad r''_s, pc' \sqcup \sigma_0 \vdash \langle \mu', s, \Gamma'\rangle \Downarrow \langle \mu'', v, \Gamma'', \sigma'\rangle \end{array}}{r_s, pc \vdash \langle \mu, e_0(e_1), \Gamma\rangle \Downarrow \langle \mu'', v, \Gamma'', \sigma'\rangle}$$

METHOD CALL

$$\frac{\begin{array}{c} r_s, pc \vdash \langle \mu, e_0, \Gamma\rangle \Downarrow \langle \mu_0, r_0, \Gamma_0, \sigma_0\rangle \qquad r_s, pc \vdash \langle \mu_0, e_1, \Gamma_0\rangle \Downarrow \langle \mu_1, m_1, \Gamma_1, \sigma_1\rangle \\ r_s, pc \vdash \langle \mu_1, e_2, \Gamma_1\rangle \Downarrow \langle \mu_2, v_2, \Gamma_2, \sigma_2\rangle \qquad \langle \mu_2, r_0, m_1, \Gamma_2\rangle\ \mathcal{R}_{Proto}\ \langle r_m, \sigma_m\rangle \\ pc' \doteq \Gamma_2(r_m, @fscope) \qquad r'_s \doteq \mu_2(r_m, m_1)(@fscope) \qquad \lambda x.s \doteq \mu_2(r_m, m_1)(@code) \\ o_s \doteq [x \mapsto v_2, @scope \mapsto r'_s, @this \mapsto r_0] \qquad r''_s \notin \mathsf{dom}(\mu_2) \qquad \mu' \doteq \mu_2\,[r''_s \mapsto o_s] \\ \sigma' \doteq pc' \sqcup \sigma_0 \sqcup \sigma_1 \sqcup \sigma_m \qquad \Gamma'' \doteq \Gamma'\,[(r''_s, x) \mapsto \sigma_2, (r''_s, @scope) \mapsto \sigma', (r''_s, @this) \mapsto \sigma_0, r''_s \mapsto \sigma'] \\ r''_s, \sigma' \vdash \langle \mu', s, \Gamma''\rangle \Downarrow \langle \mu'', v, \Gamma''', \sigma''\rangle \end{array}}{r_s, pc \vdash \langle \mu, e_0[e_1](e_2), \Gamma\rangle \Downarrow \langle \mu'', v, \Gamma''', \sigma''\rangle}$$

CONSTRUCTOR CALL

$$\frac{\begin{array}{c} r_s, pc \vdash \langle \mu, e_0, \Gamma\rangle \Downarrow \langle \mu_0, r_0, \Gamma_0, \sigma_0\rangle \qquad r_s, pc \vdash \langle \mu_0, e_1, \Gamma_0\rangle \Downarrow \langle \mu_1, v_1, \Gamma_1, \sigma_1\rangle \\ pc' \doteq \Gamma_1(r_0, @fscope) \qquad r'_s \doteq \mu_1(r_0, @fscope) \qquad \lambda x.s \doteq \mu_1(r_0, @code) \qquad r_p \doteq \mu_1(r_0, prototype) \\ \sigma_p \doteq \Gamma_1(r_0, prototype) \qquad o \doteq [@proto \mapsto r_p] \qquad r_o \notin \mathsf{dom}(\mu_1) \qquad r''_s \notin \mathsf{dom}(\mu') \\ o_s \doteq [x \mapsto v_1, @scope \mapsto r'_s, @this \mapsto r_o] \qquad \mu' \doteq \mu_1\,[r_o \mapsto o, r''_s \mapsto o_s] \\ \Gamma' \doteq \Gamma_1\,[(r_o, @proto) \mapsto pc \sqcup \sigma_p, r_o \mapsto \sigma_0, (r''_s, x) \mapsto \sigma_1, (r''_s, @scope) \mapsto \sigma_0, (r''_s, @this) \mapsto \sigma_0, r''_s \mapsto \sigma_0] \\ r''_s, pc' \sqcup \sigma_0 \vdash \langle \mu', s, \Gamma'\rangle \Downarrow \langle \mu'', v, \Gamma'', \sigma'\rangle \end{array}}{r_s, pc \vdash \langle \mu, \mathsf{new}\ e_0(e_1), \Gamma\rangle \Downarrow \langle \mu'', v, \Gamma'', \sigma'\rangle}$$

THIS - 1

$$\frac{@this \in \mu(r_s) \qquad r_{this} \doteq \mu(r_s, @this) \qquad \sigma_{this} \doteq \Gamma(r_s, @this)}{r_s, pc \vdash \langle \mu, this, \Gamma\rangle \Downarrow \langle \mu, r_{this}, \Gamma, pc \sqcup \sigma_{this}\rangle}$$

THIS - 2

$$\frac{@this \notin \mu(r_s)}{pc, r_s \vdash \langle \mu, this, \Gamma\rangle \Downarrow \langle \mu, \#global, \Gamma, pc\rangle}$$

BIN OPERATOR

$$\frac{r_s, pc \vdash \langle \mu, e_1, \Gamma\rangle \Downarrow \langle \mu_1, v_1, \Gamma_1, \sigma_1\rangle \qquad r_s, pc \vdash \langle \mu, e_2, \Gamma\rangle \Downarrow \langle \mu_2, v_2, \Gamma_2, \sigma_2\rangle}{r_s, pc \vdash \langle \mu, e_1\ \mathsf{op}_2\ e_2, \Gamma\rangle \Downarrow \langle \mu_2, \Downarrow_{\mathsf{op}_2}(v_1, v_2), \Gamma_2, \sigma_1 \sqcup \sigma_2\rangle}$$

VALUE

$$r_s, pc \vdash \langle \mu, v, \Gamma\rangle \Downarrow \langle \mu, v, \Gamma, pc\rangle$$

Fig. 3. Instrumented Semantics of JavaScript Expressions - 2

VARIABLE UPGRADE

$$\frac{\langle \mu, r_s, x\rangle \ \mathcal{R}_{Scope}\ r_x \neq undef}{r_s, pc \vdash \langle \mu, \mathsf{upgVar}(x, \sigma), \Gamma\rangle \Downarrow \langle \mu, undef, \Gamma\left[(r_x, x) \mapsto \Gamma(r_x, x) \sqcup \sigma\right], pc\rangle}$$

PROPERTY UPGRADE

$$\frac{\langle \mu, r_s, o\rangle \ \mathcal{R}_{Scope}\ r_s^o \neq undef \qquad \langle \mu, \mu(r_s^o, o), p, \Gamma\rangle \ \mathcal{R}_{Proto}\ \langle r_p, \sigma'\rangle}{r_s, pc \vdash \langle \mu, \mathsf{upgProp}(o, p, \sigma), \Gamma\rangle \Downarrow \langle \mu, undef, \Gamma\left[(r_p, p) \mapsto \Gamma(r_p, p) \sqcup \sigma\right], pc\rangle}$$

STRUCTURE UPGRADE

$$\frac{\langle \mu, r_s, o\rangle \ \mathcal{R}_{Scope}\ r_s^o \neq undef \qquad r_o \doteq \mu(r_s^o, o)}{r_s, pc \vdash \langle \mu, \mathsf{upgStruct}(o, \sigma), \Gamma\rangle \Downarrow \langle \mu, undef, \Gamma\left[r_o \mapsto \Gamma(r_o) \sqcup \sigma\right], pc\rangle}$$

SEQ

$$\frac{r_s, pc \vdash \langle \mu, s_1, \Gamma\rangle \Downarrow \langle \mu_1, v_1, \Gamma_1, \sigma_1\rangle \qquad r_s, pc \vdash \langle \mu_1, s_2, \Gamma_1\rangle \Downarrow \langle \mu_2, v_2, \Gamma_2, \sigma_2\rangle}{r_s, pc \vdash \langle \mu, s_1; s_2, \Gamma\rangle \Downarrow \langle \mu_2, v_2, \Gamma_2, \sigma_2\rangle}$$

IF - 1

$$\frac{\begin{array}{c} r_s, pc \vdash \langle \mu, e, \Gamma\rangle \Downarrow \langle \mu, v', \Gamma', \sigma'\rangle \qquad v' \notin \{0, false, undef, null\} \\ r_s, pc \sqcup \sigma' \vdash \langle \mu', s_1, \Gamma'\rangle \Downarrow \langle \mu'', v'', \Gamma'', \sigma''\rangle \end{array}}{r_s, pc \vdash \langle \mu, \mathsf{if}(e)\{s_1\}\mathsf{else}\{s_2\}, \Gamma\rangle \Downarrow \langle \mu'', v'', \Gamma'', \sigma''\rangle}$$

IF - 2

$$\frac{\begin{array}{c} r_s, pc \vdash \langle \mu, e, \Gamma\rangle \Downarrow \langle \mu, v', \Gamma', \sigma'\rangle \qquad v' \in \{0, false, undef, null\} \\ r_s, pc \sqcup \sigma' \vdash \langle \mu', s_2, \Gamma'\rangle \Downarrow \langle \mu'', v'', \Gamma'', \sigma''\rangle \end{array}}{r_s, pc \vdash \langle \mu, \mathsf{if}(e)\{s_1\}\mathsf{else}\{s_2\}, \Gamma\rangle \Downarrow \langle \mu'', v'', \Gamma'', \sigma''\rangle}$$

WHILE-1

$$\frac{r_s, pc \vdash \langle \mu, e, \Gamma\rangle \Downarrow \langle \mu', v', \Gamma', \sigma'\rangle \qquad v' \in \{0, false, undef, null\}}{r_s, pc \vdash \langle \mu, \mathsf{while}(e)\{s\}, \Gamma\rangle \Downarrow \langle \mu', undef, \Gamma', pc\rangle}$$

WHILE-2

$$\frac{\begin{array}{c} r_s, pc \vdash \langle \mu, e, \Gamma\rangle \Downarrow \langle \mu', v', \Gamma', \sigma'\rangle \qquad v' \notin \{0, false, undef, null\} \\ r_s, pc \sqcup \sigma' \vdash \langle \mu', s, \Gamma'\rangle \Downarrow \langle \mu'', v'', \Gamma'', \sigma''\rangle \qquad r_s, pc \vdash \langle \mu'', \mathsf{while}(e)\{s\}, \Gamma''\rangle \Downarrow \langle \mu''', v''', \Gamma''', \sigma'''\rangle \end{array}}{r_s, pc \vdash \langle \mu, \mathsf{while}(e)\{s\}, \Gamma\rangle \Downarrow \langle \mu''', v''', \Gamma''', \sigma'''\rangle}$$

Fig. 4. Instrumented Semantics of JavaScript Statements

Definition 3 (Well Labeled Memory). *A memory μ is said to be well-labeled by labeling Γ iff the following hold:*

$$(r,p) \in \mathsf{dom}(\Gamma) \Rightarrow r \in \mathsf{dom}(\Gamma)$$
$$r \in \mathsf{dom}(\Gamma) \Rightarrow r \in \mathsf{dom}(\mu)$$
$$\forall r \in \mathsf{dom}(\mu)\ \mathsf{dom}(\mu(r)) = \mathsf{dom}(\Gamma|_r) \wedge r \in \mathsf{dom}(\Gamma)$$
$$@scope \in \mathsf{dom}(\mu(r))\ \Gamma(\mu(r,@scope)) \leq \Gamma(r,@scope) = \Gamma(r)$$
$$(@scope, @code \in \mathsf{dom}(\mu(r)) \wedge \Gamma(r,@code) = \Gamma(r,@scope)) \vee @scope, @code \notin \mathsf{dom}(\mu(r))$$

Definition 4 (Well Labeled Big Step Transition). *A transition $r_s, pc \vdash \langle \mu, s, \Gamma\rangle \Downarrow \langle \mu', v, \Gamma', \sigma\rangle$ is said to be well labeled if μ is well labeled by Γ, μ' is well labeled by Γ' and $\Gamma(r_s) \leq pc$.*

Lemma 1 (Well Labeling Preservation). *Given a transition $r_s, pc \vdash \langle \mu, s, \Gamma\rangle \Downarrow \langle \mu', v, \Gamma', \sigma\rangle$ such that μ is well labeled by Γ' and $\Gamma(r_s) \leq pc$, then μ' is well labeled by Γ'.*

Relation to JavaScript semantics. When labeling annotations are erased from the rules, the semantics rules that define $\Downarrow$ are compliant with JavaScript semantics. We use $\Downarrow_{JS}$ to refer to JavaScript semantics [13]. The following lemma states this compliance, where we assume for simplicity that the allocator of fresh references in memory is deterministic:

Lemma 2. *Let r_s be a scope reference, μ a memory, s a statement that is syntactically equal to statement s' except for possible upgrade statements. Let Γ, Γ' be labelings and σ a security level. Then*

$$r_s, pc \vdash \langle \mu, s', \Gamma\rangle \Downarrow \langle \mu', v, \Gamma', \sigma\rangle \text{ if and only if } r_s \vdash \langle \mu, s\rangle \Downarrow_{JS} \langle \mu', v\rangle$$

The dynamic labeling is used only for defining the noninterference property and is not needed for the code instrumentation of Section 6; the semantics does not impose any constraints (constraints are added in Section 5) but just propagates security levels according to the dynamic dependencies in the program.

Example. Program (1) starting with μ_1 and Γ_1 ends with μ_1' and Γ_1':

$$\mu_1 = \begin{bmatrix} (\#global,h) \mapsto 1, (\#global,o) \mapsto undef, \\ (\#global,l) \mapsto undef \end{bmatrix} \quad \Gamma_1 = \begin{bmatrix} (\#global,h) \mapsto H, (\#global,l) \mapsto L, \\ (\#global,o) \mapsto L, \#global \mapsto L \end{bmatrix}$$
$$\mu_1' = \begin{bmatrix} (\#global,h) \mapsto 1, (\#global,o) \mapsto \#o, \\ (\#global,l) \mapsto 0 \\ (\#o,p) \mapsto 0 \end{bmatrix} \quad \Gamma_1' = \begin{bmatrix} (\#global,h) \mapsto H, (\#global,l) \mapsto H, \\ (\#global,o) \mapsto L, (\#o,p) \mapsto H, \\ \#o \mapsto H, \#global \mapsto L \end{bmatrix} \tag{2}$$

4 Non-interference

In contrast to previous works [17, 2] on dynamic information flow analysis, we choose to separate the constraints for enforcing noninterference from the dynamic labeling. This allows us to distinguish the class of secure programs but rejected by the enforcement method due to conservative constraints from the class of insecure programs. In order to introduce the definition of noninterference, we first define low equality for JavaScript memories. As is standard in low equality relations in information flow security [5] , we rely on a mapping β that relates references of objects that are equally observable in two different memories. In the following, let β be a partial injective function $\beta : \mathcal{R}ef \hookrightarrow \mathcal{R}ef$ mapping references to references. We let $\mathcal{P}rim$ be the set of primitive values including all values ranged by v in Figure 1 except references and function runtime values.

Definition 5 (β-Equality). *The following rules define the β-equality relation:*

$$\text{OBJECT}\ \frac{\mathsf{dom}(o_1) = \mathsf{dom}(o_2) = P \qquad \forall p \in P\ o_1(p) \sim_\beta o_2(p)}{o_1 \sim_\beta o_2} \qquad \text{REFERENCE}\ \frac{r_1, r_2 \in \mathcal{R}ef \qquad r_1 = \beta(r_2)}{r_1 \sim_\beta r_2} \qquad \text{PRIM}\ \frac{v_1, v_2 \in \mathcal{P}rim \qquad v_1 = v_2}{v_1 \sim_\beta v_2}$$

$$\text{FUN}\ \frac{s_1 = s_2}{\lambda x.s_1 \sim_\beta \lambda x.s_2}$$

Definition 5 states that objects are beta related if their set of properties is the same, if references are related by β, primitive values are equal, and the body of function runtime values in memory are syntactically equal. Notice that we can relax the notion of β-equality by requiring semantics equivalence between function values.

The low equality definition relies on the β-equality for values that are visible in each object. If f is a function and V is a set of elements included in its domain, let $f|_V$ be a new function defined as f but restricted to domain V.

Definition 6 (Low equality). *Two memories μ_1 and μ_2 are said to be low equal with respect to Γ_1 and Γ_2, a security level σ, and a partial injective function $\beta : \mathcal{R}ef \hookrightarrow \mathcal{R}ef$, written $\mu_1, \Gamma_1 \approx_{\beta,\sigma} \mu_2, \Gamma_2$, if μ_1 and μ_2 are well labeled by Γ_1 and Γ_2 respectively, and for all references $r \in \mathsf{dom}(\beta)$, the following holds:*

1. $\{p \in \mathsf{dom}(\mu_1(r)) \mid \Gamma_1(r,p) \leq \sigma\} = \{p \in \mathsf{dom}(\mu_2(\beta(r))) \mid \Gamma_2(\beta(r),p) \leq \sigma\} = P$
2. $\Gamma_1|_{r,P} = \Gamma_2|_{\beta(r),P}$
3. $\mu_1(r)|_P \sim_\beta \mu_2(\beta(r))|_P$
4. $\big(\Gamma_1(r), \Gamma_2(\beta(r)) \leq \sigma \wedge \mathsf{dom}(\mu_1(r)) = \mathsf{dom}(\mu_2(\beta(r)))\big) \vee \Gamma_1(r), \Gamma_2(\beta(r)) \not\leq \sigma$

Low equality relates every two objects in the memories whose references are related by β. Bullet 1 requires that the names of the visible properties of the two objects are equal, that is those whose labeling is $\leq \sigma$. Bullet 2 requires that the two labelings map the properties in P to the same security levels. Bullet 3 requires that the two objects, obtained by projecting the original objects onto their visible properties P, $\mu_1(r)\,|_P$ and $\mu_2(\beta(r))\,|_P$, be β equal. Finally, bullet 4 requires that either the structure security level in the two objects is visible and their set of properties the same or the structure level is not visible.

We say a memory μ is well labeled by Γ if for every reference r in μ, Γ is defined in r and for every property p of object $\mu(r)$, $\Gamma(r,p)$ is defined. We use the notation $\beta(\Gamma)$ for a labeling Γ' such that $\Gamma'(\beta(r),p) = \Gamma(r,p)$ and $\Gamma'(\beta(r)) = \Gamma(r)$ for every reference r in the domain of Γ.

The noninterference property requires that execution of the program preserves low equality. A partial function $\beta : \mathcal{R}ef \hookrightarrow \mathcal{R}ef$ is said to be proper if it is injective and $\beta(\#global) = \#global$ and $\beta(\#protObj) = \#protObj$.

Definition 7 (JavaScript Noninterference). *A program s is noninterferent for Γ, μ_0, μ_1, pc, and β proper denoted $\mathbf{NI}(s, \Gamma, \mu_0, \mu_1, \beta, pc)$, if whenever $\mu_0, \Gamma \approx_{\beta,\sigma} \mu_1, \beta(\Gamma)$, $r, pc \vdash \langle \mu_0, s, \Gamma\rangle \Downarrow \langle \mu_0', v_0, \Gamma_0'\rangle$, and $\beta(r), pc \vdash \langle \mu_1, s, \beta(\Gamma)\rangle \Downarrow \langle \mu_1', v_1, \Gamma_1'\rangle$ for some r, then $\mu_0', \Gamma_0' \approx_{\beta',\sigma} \mu_1', \Gamma_1'$ for some $\beta \leq \beta'$. A program s is noninterferent for Γ, denoted $\mathbf{NI}(s, \Gamma)$, if for all μ_0, μ_1, pc, β, $\mathbf{NI}(s, \Gamma, \mu_0, \mu_1, \beta, pc)$.*

Low Equatility Properties. In the following, we present some useful properties of the low equality relation.

5 Information Flow Monitor

This section introduces the semantics relation $\Downarrow_{IF}$, called monitor semantics, that extends relation $\Downarrow$ with constraints to enforce noninterference. Semantics rules of $\Downarrow_{IF}$ coincide with those of $\Downarrow$ except for constraints added to the rules where existing security labels are possibly upgraded: [Prop Update], [Prop Creation], [Assign-1], [Assign-2], [Variable Upgrade], [Property Upgrade]and [Structure Upgrade]. Essentially, these restrictions are intended to forbid the so called *sensitive upgrades* [2] (visible changes in non visible contexts). We present the complete set of constraints added to the rules of $\Downarrow$ in order to define $\Downarrow_{IF}$:

[Prop Update] $\sigma_1 \sqcup \sigma_2 \leq \Gamma_3(r_1, m_2)$. Levels σ_1 and σ_2 correspond to the reading effects of expressions evaluated in order to update property m_2. The constraint prevents flows of information from the reading effects visible at the level of m_2. Note that reading effect levels are higher than the level of the context in which they are obtained, hence the constraint also prevents sensitive upgrades.

[Prop Creation] $\sigma_1 \sqcup \sigma_2 \leq \Gamma_3(r_1)$. Levels σ_1 and σ_2 correspond to the reading effects of expressions evaluated in order to create property m_2. The constraint prevents flows from the reading effects to the structure level of the object being extended.

[Assign-1] $pc \leq \Gamma_1(r_x, x)$. The constraint prevents sensitive upgrades by requiring that the level of variable x be higher than the level of the context (pc).

[Assign-2] $pc \leq \Gamma_1(\#global)$. This rule is applied if variable x is not previously defined and thus is added to the global object. The constraint prevents sensitive upgrades by requiring that the structure security level of the global to be higher than the level of the context (pc).

[Variable Upgrade] $pc \leq \Gamma(r_x, x)$ which prevents a sensitive upgrade.

[Property Upgrade] $pc \sqcup \Gamma(r_s^o, o) \sqcup \sigma' \leq \Gamma(r_p, p)$. The constraint prevents the upgrade of the level of a property p via an object referred by a higher property o. (Notice however that the same property p can be upgraded from an alias of reference in o which level is lower).

[Structure Upgrade] $pc \sqcup \Gamma(r_s^o, o) \leq \Gamma(r_o)$. The constraint prevents a structure upgrade via a an object referred by a higher property o.

The complete semantics can be checked in Appendix A.

In the following, we use $r, pc \vdash \langle \mu, s, \Gamma \rangle \not\Downarrow_{IF}$ to denote program divergence (Note that for our semantics style, divergence means that there is not a final configuration such that a semantics derivation exists). If a constraint of the monitor is not satisfied, we say the program diverges. The following lemma relates the labeling and monitor semantics.

Lemma 3. *If* $pc, r \vdash \langle \mu, s, \Gamma \rangle \Downarrow_{IF} \langle \mu', v, \Gamma', \sigma \rangle$ *then* $pc, r \vdash \langle \mu, s, \Gamma \rangle \Downarrow \langle \mu', v, \Gamma', \sigma \rangle$.

Below we present the confinement lemma of the monitor that states that monitored executions of a program, starting with low equal memories, always produce low equal final memories if the level of the initial program counter is not observable.

Lemma 4 (Confinement). *Given a JS program s, a memory μ, a labeling Γ, a security level pc and a reference r_s such that:*

$$r_s, pc \vdash \langle \mu, s, \Gamma \rangle \Downarrow_{IF} \langle \mu', v, \Gamma', \sigma \rangle$$

for some memory μ', value v, labeling Γ' and security level σ; then for every security level $\sigma' \in \mathcal{L}$ such that $pc \not\leq \sigma'$: $\mu, \Gamma \approx_{\mathsf{id}, \sigma'} \mu', \Gamma'$, where id *is the identity function defined on the domain of μ.*

Proof. Given an arbitrary σ', if $pc \leq \sigma'$ the result holds vacuously. Consider, therefore, an arbitrary σ' such that $pc \not\leq \sigma'$, the proof proceeds by induction on the structure of the derivation of:

$$pc, r_s \vdash \langle \mu, s, \Gamma \rangle \Downarrow_{IF} \langle \mu', v, \Gamma', \sigma \rangle.$$

Base cases. For the base cases, [Variable], [Value], [This-1]and [This-2]and [Function Literal], it suffices to note that $\mu \leq \mu'$ and $\Gamma \leq \Gamma'$, then applying Lemma 16, we conclude that $\mu, \Gamma \approx_{\mathsf{id}, \sigma'} \mu, \Gamma$.

Inductive cases. We distinguish four types of inductive cases:

1. Those that do not directly change the heap: [Prop Look-up], [Bin Operator], [If-1], [If-2], [While-1], [While-2], and [Seq-1].
2. Those that directly change the heap by allocating a new object: [Function Call], [Method Call], and [Constructor Call].
3. Those that directly change the heap either by creating a new property or by updating the value of an existing property of an object: [Prop Update], [Prop Creation], [Assign-1], and [Assign-2].
4. Those that only change the labeling: [Variable Upgrade], [Property Upgrade], and [Structure Upgrade].

[Prop Look-up]. By hypothesis, there exist μ', v, Γ' and σ such that:

$$r_s, pc \vdash \langle \mu, e_1[e_2], \Gamma \rangle \Downarrow_{IF} \langle \mu', v, \Gamma', \sigma \rangle$$

By inspection of the rule [Prop Look-up], we conclude that there must exist two intermediate configurations: $\langle \mu_1, r_1, \Gamma_1, \sigma_1 \rangle$ and $\langle \mu_2, m_2, \Gamma_2, \sigma_2 \rangle$, such that:

$$r_s, pc \vdash \langle \mu, e_1, \Gamma \rangle \Downarrow_{IF} \langle \mu_1, v_1, \Gamma_1, \sigma_1 \rangle \tag{3aa}$$

$$r_s, pc \vdash \langle \mu_1, e_2, \Gamma_1 \rangle \Downarrow_{IF} \langle \mu_2, v_2, \Gamma_2, \sigma_2 \rangle \tag{3ab}$$

where $\Gamma' = \Gamma_2$ and $\mu' = \mu_2$. Applying the induction hypothesis to equations 3aa and 3ab, it follows respectively:

$$\mu, \Gamma \approx_{\mathsf{id},\sigma'} \mu_1, \Gamma_1 \tag{3ac}$$
$$\mu_1, \Gamma_1 \approx_{\mathsf{id},\sigma'} \mu', \Gamma' \tag{3ad}$$

By the transitivity of $\approx_{\mathsf{id},\sigma'}$ (Lemma 15), it follows that: $\mu, \Gamma \approx_{\mathsf{id},\sigma'} \mu', \Gamma'$.

[Function Call]. By hypothesis, there exist μ'', Γ''', σ and v'' such that:

$$r_s, pc \vdash \langle \mu, e_0(e_1), \Gamma \rangle \Downarrow \langle \mu'', v'', \Gamma''', \sigma \rangle$$

By inspection of the [Function Call] rule, it follows that there exist two configurations $\langle \mu_0, v_0, \Gamma_0, \sigma_0 \rangle$ and $\langle \mu_1, v_1, \Gamma_1, \sigma_1 \rangle$ such that:

$$r_s, pc \vdash \langle \mu, e_0, \Gamma \rangle \Downarrow \langle \mu_0, v_0, \Gamma_0, \sigma_0 \rangle \tag{3ba}$$
$$r_s, pc \vdash \langle \mu_0, e_1, \Gamma_0 \rangle \Downarrow \langle \mu_1, v_1, \Gamma_1, \sigma_1 \rangle \tag{3bb}$$

Applying the induction hypothesis to Equations 3ba and 3bb and the transitivity of $\approx_{\mathsf{id},\sigma'}$ (Lemma 15), it follows that:

$$\mu, \Gamma \approx_{\mathsf{id},\sigma'} \mu_1, \Gamma_1 \tag{3bc}$$

Let μ' and Γ'' denote the memory and labeling obtained from μ_1 and Γ_1 respectively by allocating the new scope object. Since the reference corresponding to the newly allocated scope object is not in the domain of both μ_1 and Γ_1, it follows that: $\mu_1 \leq \mu'$ and $\Gamma_1 \leq \Gamma''$. Hence, applying Lemma 16, it follows that:

$$\mu_1, \Gamma_1 \approx_{\mathsf{id},\sigma'} \mu', \Gamma'' \tag{3bd}$$

By inspection of the [Function Call] rule, we conlude that there is a configuration $\langle \mu'', v', \Gamma''', \sigma'' \rangle$, such that:

$$r''_s, pc' \sqcup \sigma_0 \vdash \langle \mu', s, \Gamma'' \rangle \Downarrow \langle \mu'', v', \Gamma''', \sigma'' \rangle \tag{3be}$$

Applying Lemma 21 to Equation 3ba, it follows that $pc \leq \sigma_0$ and consequently: $pc \leq pc' \sqcup \sigma_0$. Since by hypothesis $pc \not\leq \sigma'$, we conclude that: $pc' \sqcup \sigma_0 \not\leq \sigma'$. Therefore, we can apply the induction hypothesis to Equation 3be and conclude that:

$$\mu', \Gamma'' \approx_{\mathsf{id},\sigma'} \mu'', \Gamma''' \tag{3bf}$$

Applying the transitivity of $\approx_{\mathsf{id},\sigma'}$ (Lemma 15) to Equations 3bc, 3bd and 3bf, it follows that:

$$\mu, \Gamma \approx_{\mathsf{id},\sigma'} \mu'', \Gamma'''$$

[Prop Update]. By hypothesis, there is a configuration $\langle \mu', v', \Gamma', \sigma \rangle$ such that:

$$r_s, pc \vdash \langle \mu, e_1[e_2] = e_3, \Gamma \rangle \Downarrow \langle \mu', v', \Gamma', \sigma \rangle$$

By inspection of the [Prop Update] rule, there must exist three configurations $\langle \mu_1, v_1, \Gamma_1, \sigma_1 \rangle$, $\langle \mu_2, v_2, \Gamma_2, \sigma_2 \rangle$ and $\langle \mu_3, v_3, \Gamma_3, \sigma_3 \rangle$ such that:

$$r_s, pc \vdash \langle \mu, e_1, \Gamma \rangle \Downarrow \langle \mu_1, r_1, \Gamma_1, \sigma_1 \rangle \tag{3ca}$$
$$r_s, pc \vdash \langle \mu_1, e_2, \Gamma_1 \rangle \Downarrow \langle \mu_2, m_2, \Gamma_2, \sigma_2 \rangle \tag{3cb}$$
$$r_s, pc \vdash \langle \mu_2, e_3, \Gamma_2 \rangle \Downarrow \langle \mu_3, v_3, \Gamma_3, \sigma_3 \rangle \tag{3cc}$$

Applying the induction hypothesis to Equations 3ca, 3cb and 3cc and the transitivity of $\approx_{\mathsf{id},\sigma'}$ (Lemma 15), it follows that:

$$\mu, \Gamma \approx_{\mathsf{id},\sigma'} \mu_3, \Gamma_3 \tag{3cd}$$

The final memory μ' and the final labeling Γ' are obtained from μ_3 and Γ_3 respectively in the following way:

$$\Gamma' \doteq \Gamma_3\left[(r_1, m_2) \mapsto \sigma_1 \sqcup \sigma_2 \sqcup \sigma_3\right] \quad \mu' \doteq \mu_3\left[(r_1, m_2) \mapsto v_3\right]$$

Additionally, we know that $\sigma_1 \sqcup \sigma_2 \leq \Gamma_3(r_1, m_2)$. Applying Lemma 21 to Equation 3ca, we conclude that $pc \leq \sigma_1$ and therefore $pc \sqsubseteq \Gamma_3(r_1, m_2)$. Since, by hypothesis, $pc \not\leq \sigma'$, it follows that $\Gamma_3(r_1, m_2) \not\leq \sigma'$. Hence, by Lemma 17, it follows that

$$\mu_3, \Gamma_3 \approx_{\mathsf{id},\sigma'} \mu', \Gamma' \tag{3ce}$$

Applying the transitivity of $\approx_{\mathsf{id},\sigma'}$ to Equations 3cd and 3ce, the result follows.

[Assign-1]. By hypothesis, there is a configuration $\langle \mu', v, \Gamma', \sigma \rangle$ such that:

$$r_s, pc \vdash \langle \mu, x = e, \Gamma \rangle \Downarrow \langle \mu', v, \Gamma', \sigma \rangle$$

By inspection of the [Assign-1] rule, we conclude that there is a configuration $\langle \mu_1, v_1, \Gamma_1, \sigma_1 \rangle$, such that:

$$r_s, pc \vdash \langle \mu, e, \Gamma \rangle \Downarrow \langle \mu_1, v_1, \Gamma_1, \sigma_1 \rangle \tag{3da}$$

where $v_1 = v$ and $\sigma_1 = \sigma$. Applying the induction hypothesis to Equation 3da, we conlcude that:

$$\mu, \Gamma \approx_{\mathsf{id},\sigma'} \mu_1, \Gamma_1 \tag{3db}$$

Again, by inspection of the [Assign-1] rule, we conclude that μ' and Γ' are obtained from μ_1 and Γ_1 in the following way:

$$\Gamma' = \Gamma_1\left[(r_x, x) \mapsto \sigma\right] \quad \mu' = \mu_1\left[(r_x, x) \mapsto v\right]$$

where $\langle \mu, r_s, x \rangle \; \mathcal{R}_{Scope} \; r_x$. By the definition of $\Downarrow_{IF}$, it follows that $pc \leq \Gamma_1(r_x, x)$. Since, $pc \not\leq \sigma'$, we conclude that $\Gamma_1(r_x, x) \not\leq \sigma'$. Moreover, applying Lemma 21 to Equation 3db, we conclude that $pc \leq \sigma$ and thus, since $pc \not\leq \sigma'$, it follows that $\sigma \not\leq \sigma'$. We can therefore apply Lemma 17 to conclude that:

$$\mu_1, \Gamma_1 \approx_{\mathsf{id},\sigma'} \mu', \Gamma' \tag{3dc}$$

Applying the transivity of $\approx_{\mathsf{id},\sigma'}$ to Equations 3db and 3dc, the result follows.

The next lemma establishes that the two scope references obtained by looking up for a variable in two low equal memories starting from two β-related scope references are also $\beta - related$.

Lemma 5 (Scope Chain Inspection on Low Equal Memories). *Given two memories μ_1 and μ_2 respectively well-labeled by Γ_1 and Γ_2, a partial injective functions on $\mathcal{R}ef$, β, two references r_1 and r_2, a security level σ, and a string $m \in \mathcal{S}tr$ such that:*

$$\mu_1, \Gamma_1 \approx_{\beta,\sigma} \mu_2, \Gamma_2 \tag{5a}$$
$$r_1 \sim_\beta r_2 \tag{5b}$$
$$\langle \mu_1, r_1, m \rangle \; \mathcal{R}_{Scope} \; r_1' \tag{5c}$$
$$\langle \mu_2, r_2, m \rangle \; \mathcal{R}_{Scope} \; r_2' \tag{5d}$$
$$\Gamma_1(r_1), \Gamma_2(r_2) \leq \sigma \tag{5e}$$

Then, $r_1' \sim_\beta r_2'$.

Proof. We proceed by induction on the derivation of $\langle \mu_1, r_1, m \rangle \; \mathcal{R}_{Scope} \; r_1'$.

The bases cases correspond to Rules [Null] and [Base], whereas the inductive case correspond to the Rule [Look-up].

[Null]. If the last rule to be applied on the derivation of $\langle \mu_1, r_1, m \rangle \; \mathcal{R}_{Scope} \; r_1'$ is [Null], it means that $r_1 = null = r_1'$. From the Hypothesis 5b, we conclude that $r_2 = null$. Therefore, it follows that $r_2' = null$ and $r_1' \sim_\beta r_2'$.

[Base]. If the last rule to be applied on the derivation of $\langle \mu_1, r_1, m\rangle \ \mathcal{R}_{Scope}\ r_1'$ is [Base], it means that $m \in \mathsf{dom}(\mu_1(r_1))$ and $r_1' = r_1$. Considering the Hypotheses 5a, 5b, and 5c, it follows directly from the definition of $\approx_{\beta,\sigma}$ that $\mathsf{dom}(\mu_1(r_1)) = \mathsf{dom}(\mu_2(r_2))$. Therefore, since $m \in \mathsf{dom}(\mu_1(r_1))$, we conclude that $m \in \mathsf{dom}(\mu_2(r_2))$ and consequently that $r_2' = r_2$.

[Look-up]. If the last rule to be applied on the derivation of $\langle \mu_1, r_1, m\rangle \ \mathcal{R}_{Scope}\ r_1'$ is [Look-up], it means that $m \notin \mathsf{dom}(\mu_1(r_1))$, $r_1 \neq null$:

$$\langle \mu_1, r_1'', m\rangle \ \mathcal{R}_{Scope}\ r_1' \tag{5f}$$

where $r_1'' = \mu_1(r_1, @scope)$. By the Hypothesis 5a, 5b, and 5c, we conclude that $\mathsf{dom}(\mu_1(r_1)) = \mathsf{dom}(\mu_2(r_2))$. Therefore, it follows that $m \notin \mathsf{dom}(\mu_2(r_2))$ and:

$$\langle \mu_2, r_2'', m\rangle \ \mathcal{R}_{Scope}\ r_2' \tag{5g}$$

where $r_2'' = \mu_2(r_2, @scope)$. Since by hypothesis, μ_1 is *well labeled* by Γ_1 and μ_2 is *well labeled* by Γ_2, it follows that:

$$\Gamma_1(r_1'') \leq \Gamma_1(r_1, @scope) \leq \Gamma_1(r_1) \tag{5h}$$
$$\Gamma_2(r_2'') \leq \Gamma_2(r_2, @scope) \leq \Gamma_2(r_2) \tag{5i}$$

From Hypothesis 5e and Equations 5h and 5i, it follows that:

$$\Gamma_1(r_1, @scope) \leq \sigma \tag{5j}$$
$$\Gamma_2(r_2, @scope) \leq \sigma \tag{5k}$$

From the Hypotheses 5a and 5b and Equations 5j and 5k, we conclude, by the definition of $\approx_{\beta,\sigma}$, that:

$$r_1'' \sim_\beta r_2'' \tag{5l}$$

From the Hypothesys 5d and Equations 5j and 5k, we conclude, by the definition of $\approx_{\beta,\sigma}$, that:

$$r_1'', r_2'' \leq \sigma \tag{5m}$$

Applying the induction hypothesis to the Hypothesis 5a and Equations 5f, 5g, and 5l, and 5m, we conclude that: $r_1' \sim_\beta r_2'$.

The next lemma establishes that the two scope references obtained by looking up for a property in two low equal memories starting from two β-related scope objects are either β-related or the two corresponding *look-up* levels are unobservable.

Lemma 6 (Prototype Chain Inspection on Low Equal Memories). *Given two memories μ_1 and μ_2 well labeled by Γ_1 and Γ_2 respectively, two references r_1 and r_2, a security level σ, an injective mapping on references β, and a string $m \in \mathcal{S}tr$ such that:*

$$\mu_1, \Gamma_1 \approx_{\beta,\sigma} \mu_2, \Gamma_2 \tag{6a}$$
$$r_1 \sim_\beta r_2 \tag{6b}$$
$$\langle \mu_1, r_1, m, \Gamma_1\rangle \ \mathcal{R}_{Proto}\ \langle r_1', \sigma_1\rangle \tag{6c}$$
$$\langle \mu_2, r_2, m, \Gamma_2\rangle \ \mathcal{R}_{Proto}\ \langle r_2', \sigma_2\rangle \tag{6d}$$

Then, it follows that: $(\sigma_1 \leq \sigma \vee \sigma_2 \leq \sigma) \Rightarrow (r_1' \sim_\beta r_2' \wedge \sigma_1 = \sigma_2)$.

Proof. To prove this result one has to prove the following two implications:

$$\sigma_1 \leq \sigma \Rightarrow (r_1' \sim_\beta r_2' \wedge \sigma_1 = \sigma_2) \tag{6e}$$
$$\sigma_2 \leq \sigma \Rightarrow (r_1' \sim_\beta r_2' \wedge \sigma_1 = \sigma_2) \tag{6f}$$

The proof of Equation 6e proceeds by induction on the derivation of Equation 6c, whereas the proof of 6f proceeds by induction on the derivation of 6d. Since, these proofs are symmetric we ommit the second.

We proceed by induction on the derivation of $\langle \mu_1, r_1, m, \Gamma_1\rangle \ \mathcal{R}_{Proto}\ \langle r_1', \sigma_1\rangle$. Furthermore, we assume $\sigma_1 \leq \sigma$. Hence, we have to show that $r_1' \sim_\beta r_2' \wedge \sigma_1 = \sigma_2$. The bases cases correspond to Rules [Null] and [Base], whereas the inductive case correspond to the Rule [Look-up].

[Null]. If the last rule to be applied on the derivation of $\langle \mu_1, r_1, m, \Gamma_1\rangle \ \mathcal{R}_{Proto}\ \langle r_1', \sigma_1\rangle$ is the [Null] Rule, it means that $r_1 = null$, $r_1' = null$, and $\sigma_1 = \bot$. By the Hypothesis 6b, we conclude that $r_2 = null$, and therefore $r_2' = null$ and $\sigma_2 = \bot$, from which the result follows.

[Base]. If the last rule to be applied on the derivation of $\langle \mu_1, r_1, m, \Gamma_1\rangle \ \mathcal{R}_{Proto}\ \langle r_1', \sigma_1\rangle$ is [Base], it means that $m \in \mathsf{dom}(\mu_1(r_1))$, $r_1' = \mu_1(r_1, m)$, and $\sigma_1 = \Gamma_1(r_1, m)$. By hypothesis, $\sigma_1 = \Gamma_1(r_1, m) \leq \sigma$; hence, it follows from Hypotheses 6a and 6b (by the definition of $\approx_{\beta,\sigma}$) that: $m \in \mathsf{dom}(\mu_2(r_2))$, $\Gamma_1(r_1, m) = \Gamma_2(r_2, m)$, and $\mu_1(r_1, m) = \mu_2(r_2, m)$, from which the result follows.

[Look-up]. If the last rule to be applied on the derivation of $\langle \mu_1, r_1, m, \Gamma_1\rangle \ \mathcal{R}_{Proto}\ \langle r_1', \sigma_1\rangle$ is the [Look-up] Rule, it means that $m \notin \mathsf{dom}(\mu_1(r_1))$. Hence, there is a security level σ_1' such that:

$$\langle \mu_1, r_1'', m, \Gamma_1\rangle \ \mathcal{R}_{Proto}\ \langle r_1', \sigma_1'\rangle \tag{6g}$$
$$\sigma_1 = \sigma_1' \sqcup \Gamma_1(r_1, @proto) \sqcup \Gamma_1(r_1) \tag{6h}$$

where $r_1'' = \mu_1(r_1, @proto)$. By the Hypothesis 6b, we conclude that $r_2 \neq null$. Since, by hypothesis, $\sigma_1 \leq \sigma$, it follows that $\Gamma_1(r_1) \leq \sigma$ entailing that $\Gamma_2(r_2) \leq \sigma$ (by the Hypotheses 6a and 6b). Since both $\mu_1(r_1)$ and $\mu_2(r_2)$ have low structure security levels, it follows that that their domains coincide and therefore $m \notin \mathsf{dom}(\mu_2(r_2))$. Hence, there is a security level σ_2' such that:

$$\langle \mu_2, r_2'', m, \Gamma_2\rangle \ \mathcal{R}_{Proto}\ \langle r_2', \sigma_2'\rangle \tag{6i}$$
$$\sigma_2 = \sigma_2' \sqcup \Gamma_2(r_2, @proto) \sqcup \Gamma_2(r_2) \tag{6j}$$

where $r_2'' = \mu_1(r_2, @proto)$. From Equation 6h and the fact that $\sigma_1 \leq \sigma$, it follows that $\Gamma_1(r_1, @proto) \leq \sigma$. Since by hypothesis, $\mu_1(r_1)$ and $\mu_2(r_2)$ coincide in their low domains, we conclude that $\Gamma_1(r_1, @proto) = \Gamma_2(r_2, @proto)$, entailing that:

$$r_1'' \sim_\beta r_2'' \tag{6k}$$

Applying the induction hypothesis to the Hypothesis 6a and Equations 6g, 6i, and 6k, we conclude:

$$\sigma_1' \leq \sigma \Rightarrow (r_1' \sim_\beta r_2' \wedge \sigma_1' = \sigma_2') \tag{6l}$$

It follows from Equation 6h and $\sigma_1 \leq \sigma$, that $\sigma_1' \leq \sigma$, thus proving the result.

The following lemma states that the low equality is preserved by the monitored semantics.

Lemma 7. *For any program s, two memories μ_1 and μ_2, respectively well labeled by Γ_1 and Γ_2, references r_1 and r_2 and security levels pc and σ, if there exists a partial injective function β on $\mathcal{R}ef$ such that $\beta(\#global) = \#global$, $\beta(\#protObj) = \#protObj$, and:*

$$\mu_1, \Gamma_1 \approx_{\beta,\sigma} \mu_2, \Gamma_2 \tag{7a}$$
$$r_1 \sim_\beta r_2 \tag{7b}$$
$$r_1, pc \vdash \langle \mu_1, s, \Gamma_1\rangle \Downarrow_{IF} \langle \mu_1', v_1, \Gamma_1', \sigma_1\rangle \tag{7c}$$
$$r_2, pc \vdash \langle \mu_2, s, \Gamma_2\rangle \Downarrow_{IF} \langle \mu_2', v_2, \Gamma_2', \sigma_2\rangle \tag{7d}$$

Then, there exists a functison β' such that $\beta \leq \beta'$ and the following hold:

$$\mu_1', \Gamma_1' \approx_{\beta',\sigma} \mu_2', \Gamma_2' \tag{7e}$$
$$\big(\sigma_1 \leq \sigma \vee \sigma_2 \leq \sigma\big) \Rightarrow \big(v_1 \sim_{\beta'} v_2 \wedge \sigma_1 = \sigma_2\big) \tag{7f}$$

Proof. We start by noting that if $pc \not\leq \sigma$, then we can apply Lemma 4 to the Hypotheses 7c and 7d to conclude that:

$$\mu_1, \Gamma_1 \approx_{\mathsf{id}_1,\sigma} \mu_1', \Gamma_1' \tag{7g}$$
$$\mu_2, \Gamma_2 \approx_{\mathsf{id}_2,\sigma} \mu_2', \Gamma_2' \tag{7h}$$

where id_1 and id_2 correspond to the identity function defined on the domain of μ_1 and to the identity function defined on the domain of μ_2 (respectively). Applying Lemma 20 to Hypothesis 7a and Equations 7g and 7h, Claim 7e follows. Since $pc \not\leq \sigma$, we conclude, applying Lemma 21, that $\sigma_1 \not\leq \sigma$ and $\sigma_2 \not\leq \sigma$, thus proving Claim 7f.

In the remaining of the proof, we assume $pc \leq \sigma$. We proceed by induction on the derivation of 7c.

[VALUE] If the last rule to be applied in the derivation of 7c is [VALUE], it means that $s = v$ for some value $v \in \mathcal{V}al$. Hence, by hypothesis, it follows that:

$$r_1, pc \vdash \langle \mu_1, v, \Gamma_1 \rangle \Downarrow_{IF} \langle \mu_1, v, \Gamma_1, pc \rangle \quad r_2, pc \vdash \langle \mu_2, v, \Gamma_2 \rangle \Downarrow_{IF} \langle \mu_2, v, \Gamma_2, pc \rangle$$

Entailing that $\mu_1' = \mu_1$, $\mu_2' = \mu_2$, $\Gamma_1' = \Gamma_1$, $\Gamma_2' = \Gamma_2$, $v_1 = v_2 = v$ and $\sigma_1 = \sigma_2 = pc$. Therefore, we conclude that $\mu_1', \Gamma_1' \approx_{\beta',\sigma} \mu_2', \Gamma_2'$, for $\beta' = \beta$. Furthermore, applying Lemma 25, it follows that $v_1 = v \sim_{\beta'} v = v_2$, thus verifying the second claim of the lemma.

[VARIABLE] If the last rule to be applied in the derivation of 7c is [VARIABLE], it means that $s = x$ for some variable x. Hence, applying the hypothesis, it follows that:

$$r_1, pc \vdash \langle \mu_1, x, \Gamma_1 \rangle \Downarrow_{IF} \langle \mu_1, v_1, \Gamma_1, \sigma_1 \rangle \tag{7ja}$$
$$r_2, pc \vdash \langle \mu_2, x, \Gamma_2 \rangle \Downarrow_{IF} \langle \mu_2, v_2, \Gamma_2, \sigma_2 \rangle \tag{7jb}$$

Entailing that $\mu_1' = \mu_1$, $\mu_2' = \mu_2$, $\Gamma_1' = \Gamma_1$, $\Gamma_2' = \Gamma_2$. Thus, as in the previous case, we can immediately conclude that: $\mu_1', \Gamma_1' \sim_{\beta',\sigma} \mu_2', \Gamma_2'$, for $\beta' = \beta$.

By inspection of [VARIABLE], we conclude that:

$$\langle \mu_1, r_1, x \rangle \; \mathcal{R}_{Scope} \; r_1' \tag{7jc}$$
$$\langle \mu_2, r_2, x \rangle \; \mathcal{R}_{Scope} \; r_2' \tag{7jd}$$

Where $v_1 = \mu_1(r_1', x)$, $v_2 = \mu_2(r_2', x)$, $\sigma_1 = \Gamma_1(r_1', x) \sqcup pc$, and $\sigma_2 = \Gamma_2(r_2', x) \sqcup pc$.

Since both Equations 7ja and 7jb constitue *well labeled big step transitions*, it follows that: $\Gamma_1(r_1) \leq pc$ and $\Gamma_2(r_2) \leq pc$. Hence, since $pc \leq \sigma$, we conclude that:

$$\Gamma_1(r_1), \Gamma_2(r_2) \leq \sigma \tag{7je}$$

Applying Lemma 5 to the Hypotheses 7a and 7b and Equations 7jc, 7jd, and soundness:proof:var:eq:5, we conclude that: $r_1' \sim_\beta r_2'$.

Suppose, $\sigma_1 \leq \sigma$, this means that $\Gamma_1(r_1', x) \leq \sigma$. Since $\mu_1'(r_1')$ and $\mu_2'(r_2')$ coincide in the low properties, it follows that $\Gamma_2(r_2', x) = \Gamma_1(r_1, x) \leq \sigma$ and $v_1 = v_2$. The symmetric argument can be presented for $\sigma_2 \leq \sigma$, thus following Claim 7f.

[LOOK-UP] If the last rule to be applied in the derivation of 7c is [LOOK-UP], it means that there are two expressions e_1 and e_2 such that $s = e_1[e_2]$. Hence, by inspection of the Rule [LOOK-UP], it follows that there are eight configurations such that:

$$r_1, pc \vdash \langle \mu_1, e_1, \Gamma_1 \rangle \Downarrow \langle \mu_{11}, r_{11}, \Gamma_{11}, \sigma_{11} \rangle \tag{7kaa}$$
$$r_1, pc \vdash \langle \mu_{11}, e_2, \Gamma_{11} \rangle \Downarrow \langle \mu_{12}, v_{12}, \Gamma_{12}, \sigma_{12} \rangle \tag{7kab}$$
$$r_2, pc \vdash \langle \mu_2, e_1, \Gamma_2 \rangle \Downarrow \langle \mu_{21}, r_{21}, \Gamma_{21}, \sigma_{21} \rangle \tag{7kac}$$
$$r_2, pc \vdash \langle \mu_{21}, e_2, \Gamma_{21} \rangle \Downarrow \langle \mu_{22}, v_{22}, \Gamma_{22}, \sigma_{22} \rangle \tag{7kad}$$

where $\mu_1' = \mu_{12}$, $\mu_2' = \mu_{22}$, $\Gamma_1' = \Gamma_{12}$, $\Gamma_2' = \Gamma_{22}$, and:

$$\langle \mu_{12}, r_{11}, v_{12}, \Gamma_{12} \rangle \ \mathcal{R}_{Proto} \ \langle r_1', \sigma_1' \rangle \tag{7kae}$$
$$\langle \mu_{22}, r_{21}, v_{22}, \Gamma_{22} \rangle \ \mathcal{R}_{Proto} \ \langle r_2', \sigma_2' \rangle \tag{7kaf}$$
$$\sigma_1 = \sigma_1' \sqcup \sigma_{11} \sqcup \sigma_{12} \tag{7kag}$$
$$\sigma_2 = \sigma_2' \sqcup \sigma_{21} \sqcup \sigma_{22} \tag{7kah}$$
$$v_1 = \mu_{12}(r_1', v_{12}) \tag{7kai}$$
$$v_2 = \mu_{22}(r_2', v_{22}) \tag{7kaj}$$
$$\tag{7kak}$$

Applying the induction hypothesis to Hypotheses 7a and 7b and Equations 7kaa and 7kac, we conclude that there is a mapping β_1 such that:

$$\mu_{11}, \Gamma_{11} \approx_{\beta_1, \sigma} \mu_{21}, \Gamma_{21} \tag{7kal}$$
$$\beta \sqsubseteq \beta_1 \tag{7kam}$$
$$\big(\sigma_{11} \leq \sigma \vee \sigma_{21} \leq \sigma\big) \Rightarrow \big(r_{11} \sim_{\beta_1} r_{21} \wedge \sigma_{11} = \sigma_{21}\big) \tag{7kan}$$

Analogously, applying the induction hypothesis to the Hypothesis 7b and to Equations 7kab, 7kad and 7kal, it follows that:

$$\mu_{12}, \Gamma_{12} \approx_{\beta_2, \sigma} \mu_{22}, \Gamma_{22} \tag{7kao}$$
$$\beta_1 \sqsubseteq \beta_2 \tag{7kap}$$
$$\big(\sigma_{12} \leq \sigma \vee \sigma_{22} \leq \sigma\big) \Rightarrow \big(v_{12} \sim_{\beta_2} v_{22} \wedge \sigma_{12} = \sigma_{22}\big) \tag{7kaq}$$

Observing that $\mu_1' = \mu_{12}$, $\mu_2' = \mu_{22}$, $\Gamma_1' = \Gamma_{12}$, and $\Gamma_2' = \Gamma_{22}$, Claim 7e follows.

Suppose $\sigma_1 \leq \sigma$, from Equation 7kag we conclude that: $\sigma_1' \leq \sigma$, $\sigma_{11} \leq \sigma$, and $\sigma_{12} \leq \sigma$. From $\sigma_{11} \leq \sigma$, we conclude (by Equation 7kan) that $r_{11} \sim_{\beta_2} r_{21}$. From $\sigma_{21} \leq \sigma$, we conclude (by Equation 7kaq) that $v_{12} = v_{22} = v$. Hence, applying Lemma 6 to Equations 7kao, 7kae and 7kae (remarking that $v_{12} = v_{22} = v$) we conclude that: $(\sigma_1' \leq \sigma \vee \sigma_2' \leq \sigma) \Rightarrow (r_1' \sim_{\beta_2} r_2' \wedge \sigma_1', \sigma_2' \leq \sigma)$, which entails that: $v_1 \sim_\beta v_2$. The symmetric argument can be presented for the case $\sigma_2 \leq \sigma$, thus following Claim 7f .

[Prop Update]. By hypothesis there are three expressions e_1, e_2, and e_3, such that $s = e_1[e_2] = e_3$. By inspection of [Prop Update]Rule, it follows that there are twelve configurations such that:

$$r_1, pc \vdash \langle \mu_1, e_1, \Gamma_1 \rangle \Downarrow_{IF} \langle \mu_{11}, r_{11}, \Gamma_{11}, \sigma_{11} \rangle \tag{7caa}$$
$$r_1, pc \vdash \langle \mu_{11}, e_2, \Gamma_{11} \rangle \Downarrow_{IF} \langle \mu_{12}, m_{12}, \Gamma_{12}, \sigma_{12} \rangle \tag{7cab}$$
$$r_1, pc \vdash \langle \mu_{12}, e_3, \Gamma_{12} \rangle \Downarrow_{IF} \langle \mu_{13}, v_{13}, \Gamma_{13}, \sigma_{13} \rangle \tag{7cac}$$
$$r_2, pc \vdash \langle \mu_2, e_1, \Gamma_2 \rangle \Downarrow_{IF} \langle \mu_{21}, r_{21}, \Gamma_{21}, \sigma_{21} \rangle \tag{7cad}$$
$$r_2, pc \vdash \langle \mu_{21}, e_2, \Gamma_{21} \rangle \Downarrow_{IF} \langle \mu_{22}, m_{22}, \Gamma_{22}, \sigma_{22} \rangle \tag{7cae}$$
$$r_2, pc \vdash \langle \mu_{22}, e_3, \Gamma_{22} \rangle \Downarrow_{IF} \langle \mu_{23}, v_{23}, \Gamma_{23}, \sigma_{23} \rangle \tag{7caf}$$

Where:

$$\mu_1' = \mu_{13}\left[(r_{11}, m_{12}) \mapsto v_{13}\right] \tag{7cag}$$
$$\Gamma_1' = \Gamma_{13}\left[(r_{11}, m_{12}) \mapsto \sigma_{11} \sqcup \sigma_{12} \sqcup \sigma_{13}\right] \tag{7cah}$$
$$\sigma_1 = \sigma_{13} \tag{7cai}$$
$$\mu_2' = \mu_{23}\left[(r_{21}, m_{22}) \mapsto v_{23}\right] \tag{7caj}$$
$$\Gamma_2' = \Gamma_{23}\left[(r_{21}, m_{22}) \mapsto \sigma_{21} \sqcup \sigma_{22} \sqcup \sigma_{23}\right] \tag{7cak}$$
$$\sigma_2 = \sigma_{23} \tag{7cal}$$

From Hypothesis 7a and 7b and applying the induction hypothesis three times consecutively to Equations 7caa to 7caf, it follows that there is a partial injective function on $\mathcal{R}ef$ such that:

$$\mu_{13}, \Gamma_{13} \approx_{\beta',\sigma} \mu_{23}, \Gamma_{23} \tag{7cam}$$
$$\big(\sigma_{11} \leq \sigma \vee \sigma_{21} \leq \sigma\big) \Rightarrow \big(r_{11} \sim_{\beta'} r_{21} \wedge \sigma_{11} = \sigma_{21}\big) \tag{7can}$$
$$\big(\sigma_{12} \leq \sigma \vee \sigma_{22} \leq \sigma\big) \Rightarrow \big(m_{12} = m_{22} \wedge \sigma_{12} = \sigma_{22}\big) \tag{7cao}$$
$$\big(\sigma_{13} \leq \sigma \vee \sigma_{23} \leq \sigma\big) \Rightarrow \big(v_{13} \sim_{\beta'} v_{23} \wedge \sigma_{13} = \sigma_{23}\big) \tag{7cap}$$

Observe that Equation 7cap corresponds to the second claim of the Lemma.

We proceed by case analysis on the levels corresponding to the evaluation of e_1 and e_2.

- $\sigma_{11}, \sigma_{12}, \sigma_{21}, \sigma_{22} \leq \sigma$. In this case, from Equations 7can and 7cao, we conclude that $r_{11} \sim_\beta r_{21}$ and $m_{12} = m_{22}$. Hence, applying Lemma 18, it follows that $\mu_1', \Gamma_1' \approx_{\beta',\sigma}$, thus proving the lemma.
- $\sigma_{11} \not\leq \sigma \vee \sigma_{21} \not\leq \sigma$. From Equation 7can, it follows that $\sigma_{11}, \sigma_{21} \not\leq \sigma$. By the definition of $\Downarrow_{IF}$, it follows that:
$$\Gamma_{13}(r_{11}, m_{12}), \Gamma_{23}(r_{21}, m_{22}) \not\leq \sigma \tag{7cba}$$
We apply Lemma 17 to conclude that $\mu_{13}, \Gamma_{13} \approx_{\mathsf{id},\sigma} \mu_1', \Gamma_1'$ and $\mu_{23}, \Gamma_{23} \approx_{\mathsf{id},\sigma} \mu_2', \Gamma_2'$. Hence, applying Lemma 20 to the previous to equations and Equation 7cam, the result follows.
- $\sigma_{21} \not\leq \sigma \vee \sigma_{22} \not\leq \sigma$. This case is shown using a similar argument as in the previous one.

[Assign-1] If the last rule to be applied in the derivation of 7c is [Assign-1], it means that there is one expression e and a variable x such that $s = x = e$. Hence, by inspection of the Rule [Assign-1], it follows that there are two configurations such that:

$$r_1, pc \vdash \langle \mu_1, e, \Gamma_1 \rangle \Downarrow \langle \mu_1'', v_1, \Gamma_1'', \sigma_1 \rangle \tag{7ca}$$
$$r_2, pc \vdash \langle \mu_2, e, \Gamma_2 \rangle \Downarrow \langle \mu_2'', v_2, \Gamma_2'', \sigma_2 \rangle \tag{7cb}$$

Where:

$$\langle \mu_1, r_1, x \rangle \ \mathcal{R}_{Scope}\ r_x^1 \quad r_x^1 \neq null \tag{7cc}$$
$$\mu_1' = \mu_1'' \left[(r_x^1, x) \mapsto v_1\right] \tag{7cd}$$
$$\Gamma_1' = \Gamma_1'' \left[(r_x^1, x) \mapsto \sigma_1\right] \tag{7ce}$$
$$\langle \mu_2, r_2, x \rangle \ \mathcal{R}_{Scope}\ r_x^{21} \tag{7cf}$$
$$(r_x^{22} = r_x^{21} \wedge r_x^{21} \neq null) \vee r_x^{22} = \#global \tag{7cg}$$
$$\mu_2' = \mu_2'' \left[(r_x^{22}, x) \mapsto v_2\right] \tag{7ch}$$
$$\Gamma_2' = \Gamma_2'' \left[(r_x^{22}, x) \mapsto \sigma_2\right] \tag{7ci}$$

Applying the induction hypothesis to Hypotheses 7a and 7b and Equations 7ca and 7cb, we conclude that:

$$\mu_1'', \Gamma_1'' \approx_{\beta',\sigma} \mu_2'', \Gamma_2'' \tag{7cj}$$
$$\big(\sigma_1 \leq \sigma \vee \sigma_2 \leq \sigma\big) \Rightarrow \big(v_1 \sim_{\beta'} v_2 \wedge \sigma_1 = \sigma_2\big) \tag{7ck}$$

for a partial injective function β' which extends β. Equation 7ck corresponds to the second claim of the lemma. It remains to prove the first claim.

Applying Lemma 5 to Hypotheses 7a and 7b and Equations 7cc and 7cf, we conclude that: $r_x^1 \sim_\beta r_x^{21}$. Since $r_x^1 \neq null$, it follows that $r_x^{21} \neq null$, $r_x^{22} = r_x^{21}$ and thus:

$$r_x^1 \sim_\beta r_x^{22} \tag{7cl}$$

Since both Equations 7ca and 7ca consist of well-labeled big step transitions (by Lemma 1), it follows that: $\Gamma_1''(r_1), \Gamma_2''(r_2) \leq pc$. Since $pc \leq \sigma$, we conclude that:

$$\Gamma_1''(r_1), \Gamma_2''(r_2) \leq pc \tag{7cm}$$

Applying Lemma 18 to Hypothesis 7b and Equations 7cd, 7ce, 7ch, 7ci, 7cj, 7ck, 7cl, and 7cm, we conclude that: $\mu_1', \Gamma_1' \approx_{\beta',\sigma} \mu_2', \Gamma_2'$, thus proving the first claim of the lemma.

[Assign-2] If the last rule to be applied in the derivation of 7c is [Assign-2], it means that there is one expression e and a variable x such that $s = x = e$. Hence, by inspection of the Rule [Assign-2], it follows that there are two configurations such that:

$$r_1, pc \vdash \langle \mu_1, e, \Gamma_1 \rangle \Downarrow \langle \mu_1'', v_1, \Gamma_1'', \sigma_1 \rangle \tag{7da}$$
$$r_2, pc \vdash \langle \mu_2, e, \Gamma_2 \rangle \Downarrow \langle \mu_2'', v_2, \Gamma_2'', \sigma_2 \rangle \tag{7db}$$

Where:

$$\langle \mu_1, r_1, x \rangle \ \mathcal{R}_{Scope}\ null \tag{7dc}$$
$$\mu_1' = \mu_1''\left[(\#global, x) \mapsto v_1\right] \tag{7dd}$$
$$\Gamma_1' = \Gamma_1''\left[(\#global, x) \mapsto \sigma_1\right] \tag{7de}$$
$$\langle \mu_2, r_2, x \rangle \ \mathcal{R}_{Scope}\ r_x^{21} \tag{7df}$$
$$(r_x^{22} = r_x^{21} \wedge r_x^{21} \neq null) \vee r_x^{22} = \#global \tag{7dg}$$
$$\mu_2' = \mu_2''\left[(r_x^{22}, x) \mapsto v_2\right] \tag{7dh}$$
$$\Gamma_2' = \Gamma_2''\left[(r_x^{22}, x) \mapsto \sigma_2\right] \tag{7di}$$

Applying the induction hypothesis to Hypotheses 7a and 7b and Equations 7da and 7db, we conclude that:

$$\mu_1'', \Gamma_1'' \approx_{\beta',\sigma} \mu_2'', \Gamma_2'' \tag{7dj}$$
$$\left(\sigma_1 \leq \sigma \vee \sigma_2 \leq \sigma\right) \Rightarrow \left(v_1 \sim_{\beta'} v_2 \wedge \sigma_1 = \sigma_2\right) \tag{7dk}$$

for a partial injective function β' which extends β. Equation 7ck corresponds to the second claim of the lemma. It remains to prove the first claim.

Applying Lemma 5 to Hypotheses 7a and 7b and Equations 7dc and 7df, we conclude that: $r_x^1 \sim_\beta r_x^{21}$. Since $r_x^1 = null$, it follows that $r_x^{21} = null$, $r_x^{22} = null$. Thus, the last rule to be applied in the derivation of 7d is [Assign-2]. Since by hypothesis $\beta(\#global) = \#global$ and $\beta \leq \beta'$, we conclude that $\beta'(\#global) = \#global$. From Equation 7dj, it, therefore, follows:

$$\#global \sim_{\beta'} \#global \tag{7dl}$$

Since both Equations 7da and 7da consist of well-labeled big step transitions (by Lemma 1), it follows that: $\Gamma_1''(r_1), \Gamma_2''(r_2) \leq pc$. Since $pc \leq \sigma$, we conclude that:

$$\Gamma_1''(r_1), \Gamma_2''(r_2) \leq pc \tag{7dm}$$

Applying Lemma 18 to Hypothesis 7b and Equations 7dd, 7de, 7dh, 7di, 7dj, 7dk, 7dl, and 7dm, we conclude that: $\mu_1', \Gamma_1' \approx_{\beta',\sigma} \mu_2', \Gamma_2'$, thus proving the first claim of the lemma.

[Function Literal] If the last rule to be applied in the derivation of 7c is [Function Literal], it means that $s = \mathsf{function}(x)\{s\}$. Hence, by inspection of the Rule [Function Literal], it follows that:

$$r_1, pc \vdash \langle \mu_1, \mathsf{function}(x)\{s\}, \Gamma_1 \rangle \Downarrow \langle \mu_1', r_1', \Gamma_1', pc \rangle \tag{7ea}$$
$$r_2, pc \vdash \langle \mu_2, \mathsf{function}(x)\{s\}, \Gamma_2 \rangle \Downarrow \langle \mu_2', r_2', \Gamma_2', pc \rangle \tag{7eb}$$

Where:

$$\mu_1' = \mu_1\,[r_1' \mapsto o_1] \tag{7ec}$$
$$\Gamma_1' = \Gamma_1\,[(r_1', @fscope) \mapsto pc, (r_1', @code) \mapsto pc, (r_1', prototype) \mapsto pc] \tag{7ed}$$
$$\mu_2' = \mu_2\,[r_2' \mapsto o_2] \tag{7ee}$$
$$\Gamma_2' = \Gamma_2\,[(r_2', @fscope) \mapsto pc, (r_2', @code) \mapsto pc, (r_2', prototype) \mapsto pc] \tag{7ef}$$
$$o_1 = [@fscope \mapsto r_1, @code \mapsto \lambda x.s, prototype \mapsto protObj] \tag{7eg}$$
$$o_2 = [@fscope \mapsto r_2, @code \mapsto \lambda x.s, prototype \mapsto protObj] \tag{7eh}$$

where $r_1' \notin \mathsf{dom}(\mu_1)$ and $r_2' \notin \mathsf{dom}(\mu_2)$. From the definitions of o_1 and o_2 and noting that $pc \leq \sigma$, $r_1 \sim_\beta r_2$ (Hypothesis 7b), and $\beta(protObj) = protObj$, it follows that $o_1 \sim_\beta o_2$. Thus, applying Lemma 19, it follows that: $\mu_1', \Gamma_1' \approx_{\beta',\sigma} \mu_2', \Gamma_2'$, where $\beta \leq \beta'$ and $\beta'(r_1') = r_2'$. Additionally, observing that $\sigma_1 = \sigma_2 = pc$, both of the claims of the lemma follow.

[Bin Operator] If the last rule to be applied in the derivation of 7c is [Bin Operator], it means that $s = e_1\ \mathsf{op}_2\ e_2$. Hence, by inspection of the Rule [Bin Operator], it follows that there exist four transitions:

$$r_1, pc \vdash \langle \mu_1, e_1, \Gamma_1 \rangle \Downarrow \langle \mu_{11}, v_{11}, \Gamma_{11}, \sigma_{11} \rangle \quad r_1, pc \vdash \langle \mu_{12}, e_2, \Gamma_{11} \rangle \Downarrow \langle \mu_{12}, v_{12}, \Gamma_{12}, \sigma_{12} \rangle \tag{7fa}$$
$$r_2, pc \vdash \langle \mu_2, e_1, \Gamma_2 \rangle \Downarrow \langle \mu_{21}, v_{21}, \Gamma_{21}, \sigma_{21} \rangle \quad r_2, pc \vdash \langle \mu_{22}, e_2, \Gamma_{21} \rangle \Downarrow \langle \mu_{22}, v_{22}, \Gamma_{22}, \sigma_{22} \rangle \tag{7fb}$$

Where $\mu_1' = \mu_{12}$, $\Gamma_1' = \Gamma_{12}$, $\mu_2' = \mu_{22}$, and $\Gamma_2' = \Gamma_{22}$. From the Hypotheses 7a and 7b and Equations 7fa and 7fb, we concude, applying the induction hypothesis two times consecutively, that there is a β' which extends β such that:

$$\mu_1', \Gamma_1' \approx_{\beta',\sigma} \mu_2', \Gamma_2' \tag{7fc}$$
$$\big(\sigma_{11} \leq \sigma \vee \sigma_{21} \leq \sigma\big) \Rightarrow \big(v_{11} = v_{21} \wedge \sigma_{11} = \sigma_{21}\big) \tag{7fd}$$
$$\big(\sigma_{12} \leq \sigma \vee \sigma_{22} \leq \sigma\big) \Rightarrow \big(v_{12} = v_{22} \wedge \sigma_{12} = \sigma_{22}\big) \tag{7fe}$$

Observe that we use $=$ instead of $\sim_{\beta'}$ in Equations 7fd and 7fe, because a binary operation cannot be performed on references. Equation 7fc corresponds to the first claim of the lemma. Combining Equations 7fd and 7fe, we obtain:

$$\big(\sigma_{11} \sqcup \sigma_{12} \leq \sigma \vee \sigma_{21} \sqcup \sigma_{22} \leq \sigma\big) \Rightarrow \big(\Downarrow_{\mathsf{op}_2}(v_{11}, v_{12}) = \Downarrow_{\mathsf{op}_2}(v_{21}, v_{22}) \wedge \sigma_{11} \sqcup \sigma_{12} = \sigma_{21} \sqcup \sigma_{22}\big) \tag{7ff}$$

Observing that $\sigma_1 = \sigma_{11} \sqcup \sigma_{12}$ and $\sigma_2 = \sigma_{21} \sqcup \sigma_{22}$, it follows that Equation 7ff corresponds to the second claim of the lemma.

[This-1] If the last rule to be applied in the derivation of 7c is [This-1], it means that $@this \in \mathsf{dom}(\mu_1(r_1))$ and that:

$$\mu_1' = \mu_1 \quad \Gamma_1' = \Gamma_1 \tag{7ga}$$
$$v_1 = \mu_1(r_1, @this) \quad \sigma_1 = pc \sqcup \Gamma_1(r_1, @this) \tag{7gb}$$

From Hypotheses 7a and 7b, it follows that $\mathsf{dom}(\mu_1(r_1)) = \mathsf{dom}(\mu_2(r_2))$, we therefore conclude that: $@this \in \mathsf{dom}(\mu_2(r_2))$, from which it follows that the last rule to be applied in the derivation of 7d is also [This-1]. Hence, we conclude that:

$$\mu_2' = \mu_2 \quad \Gamma_2' = \Gamma_1 \tag{7gc}$$
$$v_2 = \mu_2(r_2, @this) \quad \sigma_2 = pc \sqcup \Gamma_2(r_2, @this) \tag{7gd}$$

From Equations 7ga and 7gc, the first claim of the lemma follows. To prove the second claim let us suppose that $\sigma_1 \leq \sigma$. This implies that $\Gamma_1(r_1, @this) \leq \sigma$. Since the objects pointed by r_1 and r_2 coincide in their

low properties (Hypothesis 7b), we conclude that: $\Gamma_1(r_1, @this) = \Gamma_2(r_2, @this)$ (from which it follows that $\sigma_1 = \sigma_2$) and that $v_1 = \mu_1(r_1, @this) = \mu_2(r_2, @this) = v_2$, thus proving the second claim of the lemma.

[This-2] If the last rule to be applied in the derivation of 7c is [This-2], it means that $@this \notin \mathsf{dom}(\mu_1(r_1))$ and that:

$$\mu_1' = \mu_1 \quad \Gamma_1' = \Gamma_1 \tag{7ha}$$
$$v_1 = \#global \quad \sigma_1 = pc \tag{7hb}$$

From Hypotheses 7a and 7b, it follows that $\mathsf{dom}(\mu_1(r_1)) = \mathsf{dom}(\mu_2(r_2))$, we therefore conclude that: $@this \notin \mathsf{dom}(\mu_2(r_2))$, from which it follows that the last rule to be applied in the derivation of 7d is also [This-2]. Hence, we conclude that:

$$\mu_2' = \mu_2 \quad \Gamma_2' = \Gamma_1 \tag{7hc}$$
$$v_2 = \#global \quad \sigma_2 = pc \tag{7hd}$$

From Equations 7ga and 7gc, the first claim of the lemma follows. To prove the second claim, it is enough to note that, by hypothesis $\beta(\#global) = \#global$ and $\sigma_1 = \sigma_2$.

[Function Call] If the last rule to be applied in the derivation of 7c is [Function Call], then by hypothesis there are two expressions e_1 and e_2, such that $s = e_1(e_2)$. By inspection of [Function Call], it follows that there are six transitions:

$$r_1, pc \vdash \langle \mu_1, e_1, \Gamma_1 \rangle \Downarrow_{IF} \langle \mu_{11}, r_{11}, \Gamma_{11}, \sigma_{11} \rangle \tag{7ia}$$
$$r_1, pc \vdash \langle \mu_{11}, e_2, \Gamma_{11} \rangle \Downarrow_{IF} \langle \mu_{12}, v_{12}, \Gamma_{12}, \sigma_{12} \rangle \tag{7ib}$$
$$r_1'', pc_1' \sqcup \sigma_{11} \vdash \langle \mu_1'', s_1, \Gamma_1'' \rangle \Downarrow_{IF} \langle \mu_1', v_1, \Gamma_1', \sigma_1 \rangle \tag{7ic}$$
$$r_2, pc \vdash \langle \mu_2, e_1, \Gamma_2 \rangle \Downarrow_{IF} \langle \mu_{21}, r_{21}, \Gamma_{21}, \sigma_{21} \rangle \tag{7id}$$
$$r_2, pc \vdash \langle \mu_{21}, e_2, \Gamma_{21} \rangle \Downarrow_{IF} \langle \mu_{22}, v_{22}, \Gamma_{22}, \sigma_{22} \rangle \tag{7ie}$$
$$r_2'', pc_2' \sqcup \sigma_{21} \vdash \langle \mu_2'', s_2, \Gamma_2'' \rangle \Downarrow_{IF} \langle \mu_2', v_2, \Gamma_2', \sigma_2 \rangle \tag{7if}$$

Where:

$$pc_1' = \Gamma_{12}(r_{11}, @fscope) \quad r_1' = \mu_{12}(r_{11}, @fscope) \quad \lambda x_1.s_1 = \mu_{12}(r_{11}, @code) \tag{7ig}$$
$$\Gamma_1'' = \Gamma_{12}\left[(r_1'', x_1) \mapsto \sigma_{12}, (r_1'', @scope) \mapsto pc_1' \sqcup \sigma_{11}, r_1'' \mapsto pc_1' \sqcup \sigma_{11}\right] \tag{7ih}$$
$$o_1 = [x_1 \mapsto v_1, @scope \mapsto r_1'] \quad \mu_1'' = \mu_{12}\left[r_1'' \mapsto o_1\right] \tag{7ii}$$
$$pc_2' = \Gamma_{22}(r_{21}, @fscope) \quad r_2' = \mu_{22}(r_{21}, @fscope) \quad \lambda x_2.s_2 = \mu_{22}(r_{21}, @code) \tag{7ij}$$
$$\Gamma_2'' = \Gamma_{22}\left[(r_2'', x_2) \mapsto \sigma_{22}, (r_2'', @scope) \mapsto pc_2' \sqcup \sigma_{21}, r_2'' \mapsto pc_2' \sqcup \sigma_{21}\right] \tag{7ik}$$
$$o_2 = [x_2 \mapsto v_2, @scope \mapsto r_2'] \quad \mu_2'' = \mu_{22}\left[r_2'' \mapsto o_2\right] \tag{7il}$$

and $r_1'' \notin \mathsf{dom}(\mu_{12})$ and $r_2'' \notin \mathsf{dom}(\mu_{22})$. From Hypotheses 7a and 7b and Equations 7ia, 7ib, 7id, and 7ie, it follows, applying the induction hypothesis, that there is a partial injective function β' on $\mathcal{R}ef$ that extends β such that:

$$\mu_{12}, \Gamma_{12} \approx_{\beta', \sigma} \mu_{22}, \Gamma_{22} \tag{7im}$$
$$\left(\sigma_{11} \leq \sigma \vee \sigma_{21} \leq \sigma\right) \Rightarrow \left(r_{11} \sim_{\beta'} r_{21} \wedge \sigma_{11} = \sigma_{21}\right) \tag{7in}$$
$$\left(\sigma_{12} \leq \sigma \vee \sigma_{22} \leq \sigma\right) \Rightarrow \left(v_{12} \sim_{\beta'} v_{22} \wedge \sigma_{12} = \sigma_{22}\right) \tag{7io}$$

There are two cases to consider: the case where $\sigma_{11} \leq \sigma$ and $pc_1' \leq \sigma$ and the opposite one.

If $\sigma_{11} \leq \sigma$ and $pc_1' \leq \sigma$, it follows from Equation 7in that $\sigma_{21} = \sigma_{11} \leq \sigma$ and $r_{11} \sim_{\beta'} r_{21}$. Since $r_{11} \sim_{\beta'} r_{21}$, $pc_1' = \Gamma_{12}(r_{11}, @fscope) \leq \sigma$, it follows by Equation 7im that: $pc_2' = \Gamma_{22}(r_{21}, @fscope) \leq \sigma$, $r_1' \sim_{\beta'} r_2'$, and

$\lambda x_1.s_1 = \lambda x_2.s_2$. Hence, by noting that Equation 7io guarantees that either $\sigma_{12}, \sigma_{22} \leq \sigma \wedge v_{12} \sim_{\beta'} v_{22}$, or $\sigma_{12}, \sigma_{22} \not\leq \sigma$, we conclude that:

$$o_1|_{\sigma}^{r_1'', \Gamma_1'} \sim_{\beta'} o_2|_{\sigma}^{r_2'', \Gamma_2'} \tag{7ip}$$

Applying Lemma 19 to Equations 7ih, 7ii, 7ik, 7il, 7im, and 7ip, we conclude that:

$$\mu_1'', \Gamma_1'' \approx_{\beta'', \sigma} \mu_2'', \Gamma_2'' \tag{7iq}$$

where $\beta' \leq \beta''$ and $\beta''(r_1'') = r_2''$. Hence, recalling that $pc_1' = pc_2'$ and that $\sigma_{11} = \sigma_{21}$, we apply the induction hypothesis to Equations 7ic, 7if and 7iq to conclude the result.

If $\sigma_{11} \not\leq \sigma$ or $pc_1' \not\leq \sigma$, it follows that $pc_1' \sqcup \sigma_{11} \not\leq \sigma$ and $pc_2' \sqcup \sigma_{21} \not\leq \sigma$. Hence, applying Lemma 4 to Equations 7ic and 7if, it follows that:

$$\mu_1'', \Gamma_1'' \approx_{\mathsf{id}, \sigma} \mu_1', \Gamma_1' \tag{7ir}$$
$$\mu_2'', \Gamma_2'' \approx_{\mathsf{id}, \sigma} \mu_2', \Gamma_2' \tag{7is}$$

Observing that $r_1'' \notin \mathsf{dom}(\mu_{12})$ and $r_2'' \notin \mathsf{dom}(\mu_{22})$ and that there is no property either in μ_{12} or in μ_{22} pointing to r_1'' or r_2'' (respectively), we conclude that:

$$\mu_{12}, \Gamma_{12} \approx_{\mathsf{id}, \sigma} \mu_1'', \Gamma_1'' \tag{7it}$$
$$\mu_{22}, \Gamma_{22} \approx_{\mathsf{id}, \sigma} \mu_2'', \Gamma_2'' \tag{7iu}$$

Applying the transitivity of $\approx_{\mathsf{id}, \sigma}$ (Lemma 15) to Equations 7ir, 7is, 7it, and 7iu, we conlcude that:

$$\mu_{12}, \Gamma_{12} \approx_{\mathsf{id}, \sigma} \mu_1', \Gamma_1' \tag{7iv}$$
$$\mu_{22}, \Gamma_{22} \approx_{\mathsf{id}, \sigma} \mu_2', \Gamma_2' \tag{7iw}$$

Applying Lemma 20 to Equations 7im, 7iv, and 7iw, the first claim of the lemma follows. As for the second claim, applying Lemma 21 to Equations 7ic and 7if, we conclude that $pc_1' \sqcup \sigma_{11} \leq \sigma_1$ and $pc_2' \sqcup \sigma_{21} \leq \sigma_2$, entailing that $\sigma_1, \sigma_2 \not\leq \sigma$ and thus verifying the claim.

[Method Call] If the last rule to be applied in the derivation of 7c is [Method Call], then by hypothesis there are three expressions e_1, e_2, and e_3, such that $s = e_1[e_2](e_3)$. By inspection of [Method Call], it follows that there are eight transitions:

$$r_1, pc \vdash \langle \mu_1, e_1, \Gamma_1 \rangle \Downarrow_{IF} \langle \mu_{11}, r_{11}, \Gamma_{11}, \sigma_{11} \rangle \tag{7ja}$$
$$r_1, pc \vdash \langle \mu_{11}, e_2, \Gamma_{11} \rangle \Downarrow_{IF} \langle \mu_{12}, m_{12}, \Gamma_{12}, \sigma_{12} \rangle \tag{7jb}$$
$$r_1, pc \vdash \langle \mu_{12}, e_3, \Gamma_{12} \rangle \Downarrow_{IF} \langle \mu_{13}, v_{13}, \Gamma_{13}, \sigma_{13} \rangle \tag{7jc}$$
$$r_1'', pc_1' \sqcup \sigma_{11} \sqcup \sigma_{12} \sqcup \sigma_m^1 \vdash \langle \mu_1'', s_1, \Gamma_1'' \rangle \Downarrow_{IF} \langle \mu_1', v_1, \Gamma_1', \sigma_1 \rangle \tag{7jd}$$
$$r_2, pc \vdash \langle \mu_2, e_1, \Gamma_2 \rangle \Downarrow_{IF} \langle \mu_{21}, r_{21}, \Gamma_{21}, \sigma_{21} \rangle \tag{7je}$$
$$r_2, pc \vdash \langle \mu_{21}, e_2, \Gamma_{21} \rangle \Downarrow_{IF} \langle \mu_{22}, m_{22}, \Gamma_{22}, \sigma_{22} \rangle \tag{7jf}$$
$$r_2, pc \vdash \langle \mu_{22}, e_3, \Gamma_{22} \rangle \Downarrow_{IF} \langle \mu_{23}, v_{23}, \Gamma_{23}, \sigma_{23} \rangle \tag{7jg}$$
$$r_2'', pc_1' \sqcup \sigma_{11} \sqcup \sigma_{12} \sqcup \sigma_m^2 \vdash \langle \mu_2'', s_2, \Gamma_2'' \rangle \Downarrow_{IF} \langle \mu_2', v_2, \Gamma_2', \sigma_2 \rangle \tag{7jh}$$

Where:

$$\langle \mu_{13}, r_{11}, m_{12}, \Gamma_{13} \rangle \mathcal{R}_{Proto} \langle r_m^1, \sigma_m^1 \rangle \tag{7ji}$$
$$pc_1' = \Gamma_{13}(r_m^1, @fscope) \quad r_1' = \mu_{13}(r_m^1, m_1)(@fscope) \quad \lambda x_1.s_1 = \mu_{13}(r_m^1, m_1)(@code) \tag{7jj}$$
$$o_1 \doteq [x_1 \mapsto v_{13}, @scope \mapsto r_1', @this \mapsto r_{11}] \quad \mu_1'' = \mu_{13}\,[r_1'' \mapsto o_1] \tag{7jk}$$
$$\Gamma_1'' \doteq \Gamma_{13}\,[(r_1'', x_1) \mapsto \sigma_{13}, (r_1'', @scope) \mapsto \sigma_1', (r_1'', @this) \mapsto \sigma_{11}, r_1'' \mapsto \sigma_1'] \tag{7jl}$$
$$\langle \mu_{23}, r_{21}, m_{22}, \Gamma_{23} \rangle \mathcal{R}_{Proto} \langle r_m^2, \sigma_m^2 \rangle \tag{7jm}$$
$$pc_2' = \Gamma_{23}(r_m^2, @fscope) \quad r_2' = \mu_{23}(r_m^2, m_2)(@fscope) \quad \lambda x_2.s_2 = \mu_{23}(r_m^2, m_2)(@code) \tag{7jn}$$
$$o_2 \doteq [x_2 \mapsto v_{23}, @scope \mapsto r_2', @this \mapsto r_{21}] \quad \mu_2'' = \mu_{23}\,[r_2'' \mapsto o_2] \tag{7jo}$$
$$\Gamma_2'' \doteq \Gamma_{23}\,[(r_2'', x_2) \mapsto \sigma_{23}, (r_2'', @scope) \mapsto \sigma_2', (r_2'', @this) \mapsto \sigma_{21}, r_2'' \mapsto \sigma_2'] \tag{7jp}$$

and $r_1'' \notin \mathsf{dom}(\mu_{13})$, $r_2'' \notin \mathsf{dom}(\mu_{23})$, $\sigma_1' = pc_1' \sqcup \sigma_{11} \sqcup \sigma_{12} \sqcup \sigma_m^1$, and $\sigma_2' = pc_2' \sqcup \sigma_{21} \sqcup \sigma_{22} \sqcup \sigma_m^2$.

From Hypotheses 7a and 7b and Equations 7ja, 7jb, 7jc, 7je, 7jf, and 7jg, it follows, applying the induction hypothesis (three times), that there is a partial injective function β' on $\mathcal{R}ef$ that extends β such that:

$$\mu_{13}, \Gamma_{13} \approx_{\beta',\sigma} \mu_{23}, \Gamma_{23} \tag{7jq}$$
$$\big(\sigma_{11} \leq \sigma \vee \sigma_{21} \leq \sigma\big) \Rightarrow \big(r_{11} \sim_{\beta'} r_{21} \wedge \sigma_{11} = \sigma_{21}\big) \tag{7jr}$$
$$\big(\sigma_{12} \leq \sigma \vee \sigma_{22} \leq \sigma\big) \Rightarrow \big(m_{12} = m_{22} \wedge \sigma_{12} = \sigma_{22}\big) \tag{7js}$$
$$\big(\sigma_{13} \leq \sigma \vee \sigma_{23} \leq \sigma\big) \Rightarrow \big(v_{13} \sim_{\beta'} v_{23} \wedge \sigma_{13} = \sigma_{23}\big) \tag{7jt}$$

There are two cases to consider: $\sigma_1' \leq \sigma$ and $\sigma_1' \not\leq \sigma$.

If $\sigma_1' \leq \sigma$, it follows from Equations 7jr to 7jt that: $r_{11} \sim_{\beta'} r_{21}$, $m_{12} = m_{22}$, $r_1' \sim_{\beta'} r_2'$, $\lambda x_1.s_1 = \lambda x_2.s_2$. Remarking that $r_{11} \sim_{\beta'} r_{21}$, $m_{12} = m_{22}$, we can apply Lemma 6 to conclude that:

$$\big(\sigma_m^1 \leq \sigma \vee \sigma_m^2 \leq \sigma\big) \Rightarrow \big(r_m^1 \sim_{\beta'} r_m^2 \wedge \sigma_m^1 = \sigma_m^2\big) \tag{7jua}$$

Since, we are assuming that $\sigma_1' \leq \sigma$, it follows from the equation above that $r_m^1 \sim_{\beta'} r_m^2$ and $\sigma_m^1 = \sigma_m^2$. Hence, using Equation 7jt, we conclude that:

$$o_1|_\sigma^{r_1'',\Gamma_1'} \sim_{\beta'} o_2|_\sigma^{r_2'',\Gamma_2'} \tag{7jub}$$

Applying Lemma 19 to Equations 7jk, 7jk, 7jo, 7jp, 7jq, and 7jub, we conclude that:

$$\mu_1'', \Gamma_1'' \approx_{\beta'',\sigma} \mu_2'', \Gamma_2'' \tag{7juc}$$

where $\beta' \leq \beta''$ and $\beta''(r_1'') = r_2''$. Hence, observing that $\sigma_1' = \sigma_2'$, we apply the induction hypothesis to Equations 7jd, 7jh, and 7juc to conclude the result.

If $\sigma_1' \not\leq \sigma$, it follows by Equations 7jr to 7jt that $\sigma_2' \not\leq \sigma$. Hence, we can apply Lemma 4 to Equations 7jd and 7jh to conclude that:

$$\mu_1'', \Gamma_1'' \approx_{\beta',\sigma} \mu_1', \Gamma_1' \tag{7jva}$$
$$\mu_2'', \Gamma_2'' \approx_{\beta',\sigma} \mu_2', \Gamma_2' \tag{7jvb}$$

Observing that $r_1'' \notin \mathsf{dom}(\mu_{13})$ and $r_2'' \notin \mathsf{dom}(\mu_{23})$ and that there is no property either in μ_{13} or in μ_{23} pointing to r_1'' or r_2'' (respectively), we conclude that:

$$\mu_{13}, \Gamma_{13} \approx_{\mathsf{id},\sigma} \mu_1'', \Gamma_1'' \tag{7jvc}$$
$$\mu_{23}, \Gamma_{23} \approx_{\mathsf{id},\sigma} \mu_2'', \Gamma_2'' \tag{7jvd}$$

Applying the transitivity of $\approx_{\mathsf{id},\sigma}$ (Lemma 15) to Equations 7jva, 7jvb, 7jvc, and 7jvd, we conlcude that:

$$\mu_{13}, \Gamma_{13} \approx_{\mathsf{id},\sigma} \mu_1', \Gamma_1' \tag{7jve}$$
$$\mu_{23}, \Gamma_{23} \approx_{\mathsf{id},\sigma} \mu_2', \Gamma_2' \tag{7jvf}$$

Applying Lemma 20 to Equations 7jq, 7jve, and 7jvf, the first claim of the lemma follows. As for the second claim, applying Lemma 21 to Equations 7jd and 7jh, we conclude that $\sigma_1' \leq \sigma_1$ and $\sigma_2' \leq \sigma_2$, entailing that $\sigma_1, \sigma_2 \not\leq \sigma$ and thus verifying the claim.

[Constructor Call] If the last rule to be applied in the derivation of 7c is [Constructor Call], then by hypothesis there are two expressions e_1 and e_2 such that $s = \mathsf{new}\ e_1(e_2)$. By inspection of [Constructor

CALL], it follows that there are six transitions:

$$r_1, pc \vdash \langle \mu_1, e_1, \Gamma_1 \rangle \Downarrow_{IF} \langle \mu_{11}, r_{11}, \Gamma_{11}, \sigma_{11} \rangle \tag{7wa}$$
$$r_1, pc \vdash \langle \mu_{11}, e_2, \Gamma_{11} \rangle \Downarrow_{IF} \langle \mu_{12}, v_{12}, \Gamma_{12}, \sigma_{12} \rangle \tag{7wb}$$
$$r_1'', pc_1' \sqcup \sigma_{11} \vdash \langle \mu_1'', s_1, \Gamma_1'' \rangle \Downarrow_{IF} \langle \mu_1', v_1, \Gamma_1', \sigma_1 \rangle \tag{7wc}$$
$$r_2, pc \vdash \langle \mu_2, e_1, \Gamma_2 \rangle \Downarrow_{IF} \langle \mu_{21}, r_{21}, \Gamma_{21}, \sigma_{21} \rangle \tag{7wd}$$
$$r_2, pc \vdash \langle \mu_{21}, e_2, \Gamma_{21} \rangle \Downarrow_{IF} \langle \mu_{22}, v_{22}, \Gamma_{22}, \sigma_{22} \rangle \tag{7we}$$
$$r_2'', pc_2' \sqcup \sigma_{21} \vdash \langle \mu_2'', s_2, \Gamma_2'' \rangle \Downarrow_{IF} \langle \mu_2', v_2, \Gamma_2', \sigma_2 \rangle \tag{7wf}$$

where: and $r_o^1 \notin \mathsf{dom}(\mu_1)$, $r_1'' \notin \mathsf{dom}(\mu')$

$$r_1' = \mu_{12}(r_{11}, @fscope) \quad \lambda x_1.s_1 = \mu_{12}(r_{11}, @code) \quad r_p^1 = \mu_{12}(r_{11}, prototype) \tag{7wg}$$
$$pc_1' = \Gamma_{12}(r_{11}, @fscope) \quad \sigma_p^1 = \Gamma_{12}(r_p^1, prototype) \tag{7wh}$$
$$o_1 = [@proto \mapsto r_p^1] \quad o_s^1 = [x_1 \mapsto v_1, @scope \mapsto r_1', @this \mapsto r_o^1] \tag{7wi}$$
$$\mu_1'' = \mu_{12}\left[r_o^1 \mapsto o_1, r_1'' \mapsto o_s^1\right] \tag{7wj}$$
$$\Gamma_1'' = \Gamma_{12}\left\{\begin{array}{l}(r_o^1, @proto) \mapsto \sigma_{11} \sqcup \sigma_p^1, r_o^1 \mapsto \sigma_{11}, \\ (r_1'', x_1) \mapsto \sigma_{12}, (r_1'', @scope) \mapsto \sigma_{11} \sqcup pc_1', \\ (r_1'', @this) \mapsto \sigma_{11}, r_1'' \mapsto \sigma_{11} \sqcup pc_1'\end{array}\right\} \tag{7wk}$$
$$r_2' = \mu_{22}(r_{21}, @fscope) \quad \lambda x_2.s_2 = \mu_{22}(r_{21}, @code) \quad r_p^2 = \mu_{22}(r_{21}, prototype) \tag{7wl}$$
$$pc_2' = \Gamma_{22}(r_{21}, @fscope) \quad \sigma_p^2 = \Gamma_{22}(r_p^2, prototype) \tag{7wm}$$
$$o_2 = [@proto \mapsto r_p^2] \quad o_s^2 = [x_2 \mapsto v_2, @scope \mapsto r_2', @this \mapsto r_o^2] \tag{7wn}$$
$$\mu_2'' = \mu_{22}\left[r_o^2 \mapsto o_2, r_2'' \mapsto o_s^2\right] \tag{7wo}$$
$$\Gamma_2'' = \Gamma_{22}\left\{\begin{array}{l}(r_o^2, @proto) \mapsto \sigma_{21} \sqcup \sigma_p^2, r_o^2 \mapsto \sigma_{21}, \\ (r_2'', x_2) \mapsto \sigma_{22}, (r_2'', @scope) \mapsto \sigma_{21} \sqcup pc_2', \\ (r_2'', @this) \mapsto \sigma_{21}, r_2'' \mapsto \sigma_{21} \sqcup pc_2'\end{array}\right\} \tag{7wp}$$

and $r_1'', r_o^1 \notin \mathsf{dom}(\mu_{12})$, $r_2'', r_o^2 \notin \mathsf{dom}(\mu_{22})$.

From Hypotheses 7a and 7b and Equations 7wa, 7wb, 7wd, and 7we, it follows, applying the induction hypothesis (two times consecutively), that there is a partial injective function β' on $\mathcal{R}ef$ that extends β such that:

$$\mu_{12}, \Gamma_{12} \approx_{\beta', \sigma} \mu_{22}, \Gamma_{22} \tag{7wq}$$
$$\big(\sigma_{11} \leq \sigma \vee \sigma_{21} \leq \sigma\big) \Rightarrow \big(r_{11} \sim_{\beta'} r_{21} \wedge \sigma_{11} = \sigma_{21}\big) \tag{7wr}$$
$$\big(\sigma_{12} \leq \sigma \vee \sigma_{22} \leq \sigma\big) \Rightarrow \big(v_{12} = v_{22} \wedge \sigma_{12} = \sigma_{22}\big) \tag{7ws}$$

There are two cases to consider: $\sigma_{11} \sqcup pc_1' \leq \sigma$ and $\sigma_{11} \sqcup pc_1' \not\leq \sigma$.

If $\sigma_{11} \sqcup pc_1' \leq \sigma$, it follows from Equations 7jr and 7js that: $r_{11} \sim_{\beta'} r_{21}$, $\lambda x_1.s_1 = \lambda x_2.s_2$, $\sigma_{11} = \sigma_{21}$, and $pc_1' = pc_2'$. From $r_{11} \sim_{\beta'} r_{21}$ and Equation 7wq, it follows that:

$$\big(\sigma_p^1 \leq \sigma \vee \sigma_p^2 \leq \sigma\big) \Rightarrow \big(r_p^1 \sim_{\beta'} r_p^2 \wedge \sigma_p^1 = \sigma_p^2\big) \tag{7wta}$$

It therefore follows that:

$$o_1|_\sigma^{r_o^1, \Gamma_1'} \sim_{\beta'} o_2|_\sigma^{r_o^2, \Gamma_2'} \quad o_s^1|_\sigma^{r_1'', \Gamma_1'} \sim_{\beta'} o_s^2|_\sigma^{r_2'', \Gamma_2'} \tag{7wtb}$$

Applying Lemma 19 to Equations 7wi, 7wj, 7wk, 7wn, 7wo, and 7wp, 7wq, and 7wtb, we conclude that:

$$\mu_1'', \Gamma_1'' \approx_{\beta'', \sigma} \mu_2'', \Gamma_2'' \tag{7wtc}$$

where $\beta' \leq \beta''$, $\beta''(r_o^1) = r_o^2$ and $\beta''(r_1'') = r_2''$.

Observing that $\beta''(r_1'') = r_2''$ and $pc_1' \sqcup \sigma_{11} = pc_2' \sqcup \sigma_{21} \leq \sigma$, we apply the induction Hypothesis to Equations 7wc, 7wf, and 7wtc, the result follows.

If $\sigma_{11} \sqcup pc_1' \not\leq \sigma$, it follows (by Equations 7wq and 7wr) that $\sigma_{21} \sqcup pc_2' \not\leq \sigma$. Hence, we can apply Lemma 4 to Equations 7wc and 7wf and conclude that:

$$\mu_1'', \Gamma_1'' \approx_{\mathsf{id},\sigma} \mu_1', \Gamma_1' \tag{7wua}$$
$$\mu_2'', \Gamma_2'' \approx_{\mathsf{id},\sigma} \mu_2', \Gamma_2' \tag{7wub}$$

From Equations 7wi, 7wj, 7wk, 7wn, 7wo, and 7wp, we conclude:

$$\mu_{12}, \Gamma_{12} \approx_{\mathsf{id},\sigma} \mu_1'', \Gamma_1'' \tag{7wuc}$$
$$\mu_{22}, \Gamma_{22} \approx_{\mathsf{id},\sigma} \mu_2'', \Gamma_2'' \tag{7wud}$$

Applying the transivity of $\approx_{\mathsf{id},\sigma}$ (Lemma 15) to Equations 7wua, 7wub, 7wuc, and 7wud, it follows:

$$\mu_{12}, \Gamma_{12} \approx_{\mathsf{id},\sigma} \mu_1', \Gamma_1' \tag{7wue}$$
$$\mu_{22}, \Gamma_{22} \approx_{\mathsf{id},\sigma} \mu_2', \Gamma_2' \tag{7wuf}$$

Finally, applying Lemma 20 to Equations 7wue, 7wuf and 7wq, we conclude that: $\mu_1', \Gamma_2' \approx_{\beta'',\sigma} \mu_2', \Gamma_2'$ for some β'' that extends β.

As for the second claim of the lemma, applying Lemma 21, we conclude that $\sigma_{11} \sqcup pc_1' \leq \sigma_1$ and $\sigma_{21} \sqcup pc_2' \leq \sigma_2$. Thus, $\sigma_1, \sigma_2 \not\leq \sigma$ which proves the claim.

[Variable Upgrade] If the last rule to be applied in the derivation of 7c is [Variable Upgrade], then by hypothesis there is a variable x and a level σ' such that $s = \mathsf{upgVar}(x, \sigma')$. By inspection of [Variable Upgrade], we conclude that:

$$r_1, pc \vdash \langle \mu_1, \mathsf{upgVar}(x, \sigma'), \Gamma_1 \rangle \Downarrow_{IF} \langle \mu_1, undef, \Gamma_1 \left[(r_x^1, x) \mapsto \Gamma_1(r_x^1, x) \sqcup \sigma' \right], pc \rangle \tag{7va}$$
$$r_2, pc \vdash \langle \mu_2, \mathsf{upgVar}(x, \sigma'), \Gamma_2 \rangle \Downarrow_{IF} \langle \mu_2, undef, \Gamma_2 \left[(r_x^2, x) \mapsto \Gamma_2(r_x^2, x) \sqcup \sigma' \right], pc \rangle \tag{7vb}$$

where $\langle \mu_1, r_1, x \rangle \mathcal{R}_{Scope}\ r_x^1$, $\langle \mu_2, r_2, x \rangle \mathcal{R}_{Scope}\ r_x^2$, and $r_1', r_2' \neq null$. Hence, applying Lemma 5 to Hypotheses 7a and 7b and recalling that we are assuming that $pc \leq \sigma$, it follows that: $r_x^1 \sim_\beta r_x^2$. From Hypothesis 7a and the fact that $r_x^1 \sim_\beta r_x^2$, we conclude that:

$$\left(\Gamma_1(r_x^1, x) = \Gamma_2(r_x^2, x) \leq \sigma \wedge \mu_1(r_x^1, x) = \mu_2(r_x^2, x)\right) \vee \Gamma_1(r_x^1, x), \Gamma_2(r_x^2, x) \not\leq \sigma \tag{7vc}$$

which implies that:

$$\left(\sigma_1', \sigma_2' \leq \sigma \wedge \mu_1(r_x^1, x) = \mu_2(r_x^2, x)\right) \vee \sigma_1', \sigma_2' \not\leq \sigma \tag{7vd}$$

The first claim of the Lemma follows from Equation 7vd and Hypothesis 7a. By remarking that $\sigma_1 = \sigma_2 = pc$ and $v_1 = v_2 = undef$, the second claim follows.

[Property Upgrade] If the last rule to be applied in the derivation of 7c is [Property Upgrade], then by hypothesis there is a variable o, a string p and a level σ' such that $s = \mathsf{upgProp}(o, p, \sigma')$. By inspection of [Property Upgrade], we conclude that:

$$r_1, pc \vdash \langle \mu_1, \mathsf{upgProp}(o, p, \sigma'), \Gamma_1 \rangle \Downarrow_{IF} \langle \mu_1, undef, \Gamma_1 \left[(r_p^1, p) \mapsto \Gamma(r_p^1, p) \sqcup \sigma' \right], pc \rangle \tag{7wa}$$
$$r_2, pc \vdash \langle \mu_2, \mathsf{upgProp}(o, p, \sigma'), \Gamma_2 \rangle \Downarrow_{IF} \langle \mu_2, undef, \Gamma_2 \left[(r_p^2, p) \mapsto \Gamma(r_p^2, p) \sqcup \sigma' \right], pc \rangle \tag{7wb}$$

where $\langle \mu_1, r_1, o \rangle \mathcal{R}_{Scope}\ r_o^1$, $\langle \mu_1, \mu_1(r_o^1, o), p, \Gamma_1 \rangle \mathcal{R}_{Proto}\ \langle r_p^1, \sigma_1'' \rangle$, $\langle \mu_2, r_2, o \rangle \mathcal{R}_{Scope}\ r_o^2$, and $\langle \mu_2, \mu_2(r_o^2, o), p, \Gamma_2 \rangle \mathcal{R}_{Proto}\ \langle r_p^2, \sigma_2'' \rangle$.

Since $\langle \mu_1, r_1, o \rangle \mathcal{R}_{Scope}\ r_o^1$ and $\langle \mu_2, \mu_2(r_o^2, o), p, \Gamma_2 \rangle \mathcal{R}_{Proto}\ \langle r_p^2, \sigma_2'' \rangle$, we can apply Lemma 5 to Hypotheses 7a and 7b and (by noting that $pc \leq \sigma$) conclude that: $r_o^1 \sim_\beta r_o^2$. Given the fact that $r_o^1 \sim_\beta r_o^2$, there are two cases to consider, either: $\Gamma_1(r_o^1, o), \Gamma_2(r_o^2, o) \leq \sigma$ or $\Gamma_1(r_o^1), \Gamma_2(r_o^2) \not\leq \sigma$. If it is the case that $\Gamma_1(r_o^1, o), \Gamma_2(r_o^2, o) \not\leq \sigma$, the definition of $\Downarrow_{IF}$ guarantees that $\Gamma_1(r_p^1, p), \Gamma_2(r_p^2, p) \not\leq \sigma$. Upgrading the levels

of high properties preserves the low equality, therefore the result follows. It remains to prove that the result holds for the case in which $\Gamma_1(r_o^1, o), \Gamma_2(r_o^2, o) \leq \sigma$. For this case, it follows from the fact that $r_o^1 \sim_\beta r_o^2$ that $\mu_1(r_o^1, o) \sim_\beta \mu_2(r_o^2, o)$.

Since $r_o^1 \sim_\beta r_o^2$, $\langle \mu_1, \mu_1(r_o^1, o), p, \Gamma_1\rangle \ \mathcal{R}_{Proto}\ \langle r_p^1, \sigma_1''\rangle$, $\langle \mu_2, \mu_2(r_o^2, o), p, \Gamma_2\rangle \ \mathcal{R}_{Proto}\ \langle r_p^2, \sigma_2''\rangle$, we can apply Lemma 6 to the Hypothesis 7a and conclude that:

$$(\sigma_1'' \leq \sigma \vee \sigma_2'' \leq \sigma) \Rightarrow (r_p^1 \sim_\beta r_p^2 \wedge \sigma_1'', \sigma_2'' = \sigma) \tag{7wc}$$

Hence, there are two cases to consider: either $\sigma_1'', \sigma_2'' \leq \sigma$ or $\sigma_1'', \sigma_2'' \not\leq \sigma$.

If $\sigma_1'', \sigma_2'' \leq \sigma$, it follows that $r_p^1 \sim_\beta r_p^2$ and $\sigma_1'' = \sigma_2''$. Furthermore, it follows that property p is labeled with the same level in both memories ($\Gamma_1(r_p^1, p) = \Gamma_2(r_p^2, p)$) and that its corresponding values in the two memories are β-related ($\mu_1(r_p^1, p) \sim_\beta \mu_2(r_p^2)$). Since property p is updated to the same level in both executions, the low equality is preserved independently of the visibility of the new level of p.

If $\sigma_1'', \sigma_2'' \not\leq \sigma$, the definition of $\Downarrow_{IF}$ guarantees that $\Gamma_1(r_p^1, p), \Gamma_2(r_p^2, p) \not\leq \sigma$. Upgrading the levels of high properties preserves the low equality, therefore the result follows.

[IF-1] If the last rule to be applied in the derivation of 7c is [IF-1], then by hypothesis there is an expression e, and two statements s_1 and s_2 such that $s = \mathsf{if}(e)\{s_1\}\mathsf{else}\{s_2\}$. By inspection of [IF-1], we conclude that there are three transitions:

$$r_1, pc \vdash \langle \mu_1, e, \Gamma_1\rangle \Downarrow_{IF} \langle \mu_{10}, true, \Gamma_{10}, \sigma_{10}\rangle \tag{7xa}$$

$$r_1, pc \sqcup \sigma_{10} \vdash \langle \mu_{10}, s_1, \Gamma_{10}\rangle \Downarrow_{IF} \langle \mu_1', v_1, \Gamma_1', \sigma_1\rangle \tag{7xb}$$

$$r_2, pc \vdash \langle \mu_2, e, \Gamma_2\rangle \Downarrow_{IF} \langle \mu_{20}, v_{20}, \Gamma_{20}, \sigma_{20}\rangle \tag{7xc}$$

Applying the induction hypothesis to Hypotheses 7a and 7b and Equations 7xa and 7xc, we conclude that:

$$\mu_{10}, \Gamma_{10} \approx_{\beta', \sigma} \mu_{20}, \Gamma_{20} \tag{7xd}$$

$$\big(\sigma_{10} \leq \sigma \vee \sigma_{20} \leq \sigma\big) \Rightarrow \big(true = v_{20} \wedge \sigma_{10} = \sigma_{20}\big) \tag{7xe}$$

There are two cases to consider: $\sigma_{10}, \sigma_{20} \leq \sigma$ and $\sigma_{10}, \sigma_{20} \not\leq \sigma$.

If $\sigma_{10}, \sigma_{20} \leq \sigma$, it follows that $v_{20} = true$ and thus we conclude that the Rule [IF-1]was applied in the derivation of Hypothesis 7d. Hence:

$$r_2, pc \sqcup \sigma_{20} \vdash \langle \mu_{20}, s_1, \Gamma_{20}\rangle \Downarrow_{IF} \langle \mu_2', v_2, \Gamma_2', \sigma_2\rangle \tag{7xf}$$

Applying the induction hypothesis to Hypothesis 7b and Equations 7xb, 7xd, and 7xf, the result follows.

If $\sigma_{10}, \sigma_{20} \not\leq \sigma$, the rule used in the derivation of Hypothesis 7d can be either [IF-1]or [IF-2]. Hence, we have to consider two cases. Since they are similar, we will just consider one of them. Suppose that $v_{20} = false$, then:

$$r_2, pc \sqcup \sigma_{20} \vdash \langle \mu_{20}, s_2, \Gamma_{20}\rangle \Downarrow_{IF} \langle \mu_2', v_2, \Gamma_2', \sigma_2\rangle \tag{7xg}$$

Since $\sigma_{10}, \sigma_{20} \not\leq \sigma$, we can apply Lemma 4 to Equations 7xc and 7xg, to conclude that:

$$\mu_{10}, \Gamma_{10} \approx_{\mathsf{id}, \sigma} \mu_1', \Gamma_1' \tag{7xh}$$

$$\mu_{20}, \Gamma_{20} \approx_{\mathsf{id}, \sigma} \mu_2', \Gamma_2' \tag{7xi}$$

Applying Lemma 20 to Equations 7xd, 7xh and 7xi, the result follows.

Theorem 1 (Security). *For any proper β, program s, memory μ_0 well labeled by Γ, reference r, and level pc such that $r, pc \vdash \langle \mu_0, s, \Gamma\rangle \Downarrow_{IF} \langle \mu_0', v, \Gamma', \sigma\rangle$, then for all memories μ_1 either $\mathbf{NI}(s, \Gamma, \mu_0, \mu_1, \beta, pc)$, or $\beta(r), pc \vdash \langle \mu_1, s, \beta(\Gamma)\rangle \not\Downarrow_{IF}$.*

It is well-known that noninterference $\mathbf{NI}(s, \Gamma)$ is not a safety property [31] since it cannot be stated as a trace property. Monitors cannot enforce noninterference [22] but just over-approximations of the property [8].

This means that neither can we prove $\mathbf{NI}(s, \Gamma)$ for one monitored execution, nor can we prove that for all equivalent memory μ_1, $\mathbf{NI}(s, \Gamma, \mu_0, \mu_1, \beta)$ holds (recall that our noninterference notion does not depend on the monitor). Instead, our security theorem claims that a terminating program s in the monitor *either* holds the noninterference property for a given memory μ_1 *or* it ensures that s diverges with memory μ_1 when monitored. Proof is by structural induction and can be found in the full version of the paper [1].

Secure program rejected by the monitor. The following program is secure according to the noninterference property, but is rejected by the monitor with initial memory and labeling defined in Section 3, (2):

$$\mathsf{if}(h)\{o[\text{"}p\text{"}] = 0\}\mathsf{else}\{o[\text{"}p\text{"}] = 0\}$$

Discussion on Type System Extraction. Using the same constraints given by the monitor, we can extract a type system for a JavaScript subset limited to static variable references and statically determined scope. Hence, the labeling can be computed by an algorithm that statically determines a function Γ, $\Gamma : (\mathcal{S}tr \mapsto \mathcal{S}tr) \cup \mathcal{R}ef \mapsto \mathcal{L}$ for each program point in the program (for loops, the algorithm must compute a fix point of labeling functions obtained in several interactions). We assume that programs are desugared in such a way that property updates can only be written as $x.y = e$. Any command can be desugared to this form using additional local variables. Using the set of Γ functions, the typing rule for property update and creation has the following form:

$$\textsc{Property Update/Creation}\quad \frac{pc, \Gamma \vdash x : \sigma_1 \qquad pc, \Gamma \vdash y : \sigma_2 \qquad \sigma_1 \sqcup \sigma_2 \leq \Gamma(x, y)}{pc, \Gamma \vdash x.y = e}$$

6 Information Flow Instrumentation Compiler

Formal specification. Compilation is done by a function, $\mathcal{C}_{\mathcal{P}}\langle . \rangle$, from JavaScript programs to JavaScript programs. The compiler inlines the constraints defined by the monitor of Section 5. The compilation function $\mathcal{C}_{\mathcal{P}}\langle . \rangle$ is defined using a compilation function, $\mathcal{C}\langle . \rangle$, on statements and a compilation function, $\mathcal{C}_l\langle . \rangle$, on simple expressions (identifier, primitive values, and the *this* keyword). Their formal definitions are given in Figures 6, 6, and 6. The compilation function for statements parses JavaScript statements and the upgrade statements presented in Figure 1. Every object has a property *_lab* that maps its remaining properties to the corresponding security levels (this variable emulates Γ when fixed to the object's reference). Variable *_pc* holds the security level of the current context. The security lattice is a parameter of the compiler stored at runtime in variable *_lat*; it implements a property *bot* for the bottom of the lattice and a method *lub* for calculating the least upper bound of two security levels. Function *_setUpLab*() sets up the labeling property in the current scope object. Function *_GetPropLev*(.,.) looks up for the property level in the labeling of the object specified as first parameter. Function *_GetStructLev*(.) returns the structure level of the object given as parameter. Function *_GetVarLev*(.,.) returns the security level of the variable given as first parameter in the labeling given as second parameter. Function *_SetPropLev*(.,.,.,.,.) dynamically modifies the labeling *_lab* adding a property to the labeling if the property does not exist or modifying it, if it does. Finally, function *_Enforce*(.,.) receives two security levels l and l' as parameters; if $l \leq l'$ then the statement is equivalent to $undef$, otherwise it does not terminate. This is done to ensure that the execution of a compiled program is similar to the monitored execution of the original program. In practice, however, the compiler throws an exception. In the specification of the compiler we only consider normalized programs as established in Definition 8.

Definition 8 (Syntax of normalized programs).

$$
\begin{array}{lll}
s ::= s_1; s_2 & & \textit{sequence of statements} \\
\quad | \ \mathsf{if}^{\mathsf{i}}(\hat{e})\{s_1\}\mathsf{else}\{s_2\} & & \textit{conditional} \\
\quad | \ \mathsf{while}^{\mathsf{i}}(\hat{e})\{s\} & & \textit{while loop} \\
\hat{e} ::= x & & \textit{identifier} \\
\quad | \ v & & \textit{primitive values} \\
\quad | \ \textit{this} & & \textit{this keyword} \\
\bar{e} ::= x[\hat{e}] & & \textit{member selector} \\
\quad | \ (\hat{e}_1 \ \mathsf{op}_2 \ \hat{e}_2) & & \textit{binary operations} \\
\quad | \ \hat{e} & & \textit{very simple expression} \\
\bar{\bar{e}} ::= x(\hat{e})^i & & \textit{function call} \\
\quad | \ x[\hat{e}_1](\hat{e}_2)^i & & \textit{method call} \\
\quad | \ \mathsf{new} \ x(\hat{e})^i & & \textit{constructor call} \\
\quad | \ \mathsf{function}^{\mathsf{i}}(x)\{s\} & & \textit{function literal} \\
e ::= x[\hat{e}_1] = \hat{e}_2 & & \textit{property assignment} \\
\quad | \ x = \bar{e} & & \textit{variable assignment 1} \\
\quad | \ x = \bar{\bar{e}} & & \textit{variable assignment 2}
\end{array}
$$

VARIABLE LOOKUP
$$\mathcal{C}_l\langle x \rangle = \_lat.lub(\_pc, \_GetVarLev(\text{"}x\text{"}, \_lab))$$

VALUE
$$\mathcal{C}_l\langle v \rangle = \_pc$$

THIS
$$\mathcal{C}_l\langle this \rangle = \_lat.lub(\_pc, \_GetVarLev(\text{"}this\text{"}, \_lab))$$

PROPERTY LOOKUP
$$\mathcal{C}_l\langle x[\hat{e}] \rangle = \_lat.lub(\_pc, \mathcal{C}_l\langle x \rangle, \mathcal{C}_l\langle \hat{e} \rangle \_GetPropLev(x, \hat{e}))$$

BINARY OPERATION
$$\mathcal{C}_l\langle \hat{e}_1 \ \mathsf{op}_2 \ \hat{e}_2 \rangle = \_lat.lub(\mathcal{C}_l\langle \hat{e}_1 \rangle, \mathcal{C}_l\langle \hat{e}_2 \rangle)$$

Fig. 5. A compiler for the level of simple expressions

The theorem of correctness says that if a program terminates starting from a given configuration (original configuration) with the monitor semantics, then the compiled counterpart also terminates in a "similar" configuration. In order to state the theorem we need to formally define similarity between memories. Memory similarity requires that for all original references, labelings are the same and values are similar (using the β-equality defined in Section 4).

Definition 9 (Memory Similarity). *A memory μ labeled by Γ is similar to a memory μ' w.r.t. β, written $\mu, \Gamma \ \mathcal{R}_\beta \ \mu'$, iff $\mathsf{dom}(\beta) = \mathsf{dom}(\mu)$ and for all $r \in \mathsf{dom}(\beta)$ where $o = \mu(r)$ and $o' = \mu'(\beta(r))$ the following holds:*

$$
\begin{aligned}
\mathsf{dom}(o) = \mathsf{dom}(\mu'(\beta(r))(lab)) &\subseteq \mathsf{dom}(o') \\
\forall p \in \mathsf{dom}(o) \ o(p) &\sim_\beta o'(p) \\
\forall p \in \mathsf{dom}(o) \ \Gamma(r, p) &= \mu'(\beta(r))(lab)(p)
\end{aligned}
$$

In the formal specification and theorems we assume that variables and functions introduced by the compiler are fresh and cannot overlap with variables or functions in untrusted code. However, in the implementation we use randomization to satisfy this assumption (see Section 7).

Theorem 2 (Correctness). *Given a reference mapping β, a labeled configuration $\langle \mu, s, \Gamma \rangle$, a configuration $\langle \mu', \mathcal{C}_\mathcal{P}\langle s \rangle \rangle$, two scope references r and $\beta(r)$ in μ and μ' resp., and a security level $pc = \mu'(\beta(r))(\_pc)$, such*

FUNCTION LITERAL

$$\frac{\mathcal{C}\langle s\rangle = s' \qquad s'' = \left|\begin{array}{l}\mathsf{var}\ \_lab;\\ \_pc = \_lat.lub(\_pc, arguments.callee.\_pc);\\ \_lab = \_InitLab(arguments.callee.\_lab, "x", argLev, \mathsf{vars}(s), \_pc);\\ s'\end{array}\right.}{\mathcal{C}\langle \mathsf{function}^{\mathsf{i}}(x)\{s\}\rangle = \left|\begin{array}{l} z_i = \mathsf{function}(x, argLev, \_pc)\{s''\};\\ z_i.\_lab = \_lab;\\ z_i.\_pc = \_pc;\\ w_i.lev = \_pc\end{array}\right.}$$

FUNCTION CALL - 1

$$\mathcal{C}\langle x(\hat{e})^i\rangle = \left|\begin{array}{l} aux = x(\hat{e}, \mathcal{C}_l\langle\hat{e}\rangle, \mathcal{C}_l\langle x\rangle)\\ z_i = aux.val;\\ w_i = aux.lev;\end{array}\right.$$

METHOD CALL - 1

$$\mathcal{C}\langle x[\hat{e}_1](\hat{e}_2)^i\rangle = \left|\begin{array}{l} lev = \_lat.lub(\mathcal{C}_l\langle x\rangle, \mathcal{C}_l\langle\hat{e}_1\rangle, \_GetPropLev(x, \hat{e}_2));\\ aux = x[\hat{e}_1](\hat{e}_2, \mathcal{C}_l\langle\hat{e}_2\rangle, lev)\\ z_i = aux.val;\\ w_i = aux.lev;\end{array}\right.$$

CONSTRUCTOR CALL - 1

$$\mathcal{C}\langle \mathsf{new}\ x(\hat{e})^i\rangle = \left|\begin{array}{l} w_i = \_lat.lub(\mathcal{C}_l\langle x\rangle, x.\_pc);\\ z_i = \_InitObject(Object.create(x.prototype), \mathcal{C}_l\langle x\rangle);\\ aux = x.call(z_i, \hat{e}, \mathcal{C}_l\langle\hat{e}\rangle, \mathcal{C}_l\langle x\rangle);\\ \mathsf{if}(aux)\{\\ \quad z_i = aux.val;\\ \quad w_i = aux.lev\\ \}\end{array}\right.$$

Fig. 6. Compiling indexed expressions

PROPERTY UPDATE/CREATION

$$\mathcal{C}\langle x[\hat{e}_1] = \hat{e}_2\rangle = \left|\begin{array}{l}\mathsf{if}(\hat{e}_1\ \mathsf{in}\ x)\ \{\\ \quad \_Enforce(\_lat.lub(\mathcal{C}_l\langle x\rangle, \mathcal{C}_l\langle \hat{e}_1\rangle, \_pc), \_GetPropLev(x, \hat{e}_1))\\ \}\ \mathsf{else}\ \{\\ \quad \_Enforce(\_lat.lub(\mathcal{C}_l\langle x\rangle, \mathcal{C}_l\langle \hat{e}_1\rangle, \_pc), \_GetStructLev(x))\\ \}\\ \_SetPropLev(x, \hat{e}_1, \_lat.lub(\mathcal{C}_l\langle x\rangle, \mathcal{C}_l\langle \hat{e}_1\rangle, \mathcal{C}_l\langle \hat{e}_2\rangle, \_pc));\\ x[\hat{e}_1] = \hat{e}_2\end{array}\right.$$

SIMPLE EXPRESSION ASSIGNMENT

$$\mathcal{C}\langle x = \bar{e}\rangle = \left|\begin{array}{l}\_Enforce(\_pc, \mathcal{C}_l\langle x\rangle);\\ \_SetVarLev(\text{"}x\text{"}, \_lat.lub(\mathcal{C}_l\langle \bar{e}\rangle, \_pc), \_lab);\\ x = \bar{e}\end{array}\right.$$

CALL EXPRESSION ASSIGNMENT

$$\frac{\mathcal{C}\langle \bar{\bar{e}}\rangle = s \qquad i = \mathsf{index}(\bar{\bar{e}})}{\mathcal{C}\langle x = \bar{\bar{e}}\rangle = \left|\begin{array}{l}s;\\ \_Enforce(\_pc, \mathcal{C}_l\langle x\rangle);\\ \_SetVarLev(\text{"}x\text{"}, w_i, \_lab);\\ x = z_i\end{array}\right.}$$

SEQUENCE

$$\frac{\mathcal{C}\langle s_1\rangle = s_1' \qquad \mathcal{C}\langle s_2\rangle = s_2'}{\mathcal{C}\langle s_1; s_2\rangle = s_1'; s_2'}$$

IF

$$\frac{\mathcal{C}\langle s_1\rangle = s_1' \qquad \mathcal{C}\langle s_2\rangle = s_2'}{\mathcal{C}\langle \mathsf{if}^{\mathsf{i}}(\hat{e})\{s_1\}\mathsf{else}\{s_2\}\rangle = \left|\begin{array}{l}z_i = \_pc;\\ \_pc = \_lat.lub(\_pc, \mathcal{C}_l\langle \hat{e}\rangle);\\ \mathsf{if}(\hat{e})\{s_1'\}\mathsf{else}\{s_2'\}\\ \_pc = z_i\end{array}\right.}$$

WHILE

$$\frac{\mathcal{C}\langle s\rangle = s'}{\mathcal{C}\langle \mathsf{while}^{\mathsf{i}}(\hat{e})\{s\}\rangle = \left|\begin{array}{l}z_i = \_pc;\\ \_pc = \_lat.lub(\_pc, \mathcal{C}_l\langle \hat{e}\rangle);\\ \mathsf{while}(\hat{e})\{s'\}\\ \_pc = z_i\end{array}\right.}$$

UPGRADE VARIABLE

$$\mathcal{C}\langle \mathsf{upgVar}(x, l)\rangle = \_SetVarLev(\text{"}x\text{"}, \_lat.lub(\mathcal{C}_l\langle x\rangle, l), \_lab)$$

UPGRADE PROPERTY

$$\mathcal{C}\langle \mathsf{upgProp}(x, v, l)\rangle = \_SetPropLev(x, v, \_lat.lub(\_GetPropLev(x, v), l, \mathcal{C}_l\langle x\rangle))$$

UPGRADE STRUCTURE

$$\mathcal{C}\langle \mathsf{upgStruct}(x, l)\rangle = \_SetStructLev(x, \_lat.lub(\_GetStructLev(x), l, \mathcal{C}_l\langle x\rangle))$$

PROGRAM

$$\frac{\mathcal{C}\langle s\rangle = s'}{\mathcal{C}_{\mathcal{P}}\langle s\rangle = \left|\begin{array}{l}\mathsf{var}\ \_lab, \_pc;\\ \_pc = \_lat.bot;\\ \_lab = \_setUpLab(\mathsf{vars}(s), \_lat.bot);\\ s'\end{array}\right.}$$

Fig. 7. Compiler Specification

that $\mu, \Gamma \ \mathcal{R}_\beta \ \mu'$*; then:*

$$\exists \langle \mu_f, v_f, \Gamma_f, \sigma \rangle \ r, pc \vdash \langle \mu, s, \Gamma \rangle \Downarrow_{IF} \langle \mu_f, v_f, \Gamma_f, \sigma \rangle$$
$$\text{iff}$$
$$\exists \langle \mu'_f, v'_f \rangle \ \beta(r) \vdash \langle \mu', \mathcal{C}_\mathcal{P}\langle s \rangle \rangle \Downarrow_{JS} \langle \mu'_f, v'_f \rangle$$

If both configurations converge: $\mu_f, \Gamma_f \ \mathcal{R}_{\beta'} \ \mu'_f$ *with* $v_f \sim_{\beta'} v'_f$ *for some* $\beta \leq \beta'$*.*

Using the correctness theorem and Lemma 2, we prove the following lemma that relates monitored with unmonitored (but labeled) executions of compiled programs. In the following, we use the term *proper compiler labeling* for a labeling Γ that assigns the same level to all the compiler variables.

Lemma 8. *For any reference* r*, security level* pc*, and initial configuration* $\langle \mu, s, \Gamma \rangle$*, such that* $s = \mathcal{C}_\mathcal{P}\langle s' \rangle$ *for some program* s'*, then* $r, pc \vdash \langle \mu, s, \Gamma \rangle \Downarrow \langle \mu', v, \Gamma', \sigma \rangle$ iff $r, pc \vdash \langle \mu, s, \Gamma \rangle \Downarrow_{IF} \langle \mu', v, \Gamma', \sigma \rangle$*.*

We assume that Γ_0 is a labeling mapping the global object to the bottom of the security lattice considered. The following theorem shows that compiled code is secure w.r.t. the noninterference property given in Section 4.

Theorem 3 (Security). *For any program* s*,* $\mathsf{NI}(\mathcal{C}_\mathcal{P}\langle s \rangle, \Gamma_0)$ *holds.*

Proof. The result follows immediately by Theorem 2, Lemma 8, and Theorem 1.

Modular extensions for external interfaces Although JavaScript can be used as a general purpose language, most JavaScript programs are intended to be run in a browser in the context of a web page. Such programs often interact with the web page on which they are included through the DOM API. We provide a way to extend the enforcement functions used by our compiler in order to deal with the DOM API. We use the same approach to deal with other external APIs, namely the XMLHttpRequest API. For this reason, the term *external function* is used for all functions whose code is not given by the programmer and hence is not available for instrumentation. Managing external APIs consists in specifying for each one of them, an *IFlow signature*, that describes:

Its effect on the security levels of its arguments (*updtArgsLevels*), of the object on which it was called (*updtObjLevel*), and return value (*computeRetLevel*);
The circumstances in which the API can be successfully invoked (*enforce*);
A preprocessing function for its arguments (*processArg*) and for its return value (*computeRetValue*).

A call to an external API of the kind $o.m(arg)$ is compiled as shown in Listing 6 and denoted as $\mathcal{C}_\mathcal{P}\langle o.m(arg), ifs \rangle$ where ifs is its IFlow signature.

```
1 enforce(arg_level, ctxt_level);
2 val_i = o.m(processArg(arg));
3 val_i = computeRetValue(val_i);
4 lev_i = computeRetLevel(arg_level, ctxt_level, val_i);
5 updtArgsLevels(arg, args_level, ctxt_level);
6 updtObjLevel(o, args_level, ctxt_level);
```

Listing 1.4. Compilation of call to external API

In order to preserve the security results of Theorem 3, and due to the compositionality of our enforcement technique, the following lemma must be proved for each external API handled by the compiler. We assume in the lemma that JavaScript semantics rules $\Downarrow_{JS}$ are extended in order to handle external APIs. (See [24] for a JavaScript extension to HTML tags including the iframe tag and the Postmessage API, see [16] for a formal semantics of the DOM API).

Lemma 9. *Let* $o.m(e)$ *be an invocation to an external API and* ifs *its IFlow signature, then* $\mathsf{NI}(\mathcal{C}_\mathcal{P}\langle o.m(e), ips \rangle, \Gamma_0)$ *holds.*

IFlow Signatures. In order to show the effectiveness of our modular approach to handle external functions, we provide IFlow signatures for the following external APIs: *appendChild*, *createTextNode*, *createElement*, *eval*, *setTimeout*.

We give examples of IFlow signatures in order to illustrate the simplicity and scalability of the proposed mechanism. In the case of the ***eval*** external function a possible IFlow signature consists in using the *processArg* function to compile the argument that is passed to it at runtime. This is equivalent to on-the-fly inlining as proposed by Magazinius [27] for an imperative language extended with an *eval* statement.

```
function processArg(str) {  compile str  }
```

Listing 1.5. IFlow signature for the eval function

In the ***appendChild*** external method, the default IFlow signature prevents the execution of the method when trying to append an untrusted node to a trusted node. This is achieved through the *enforce* function:

```
function enforce(arg_level, ctxt_level) {
  if(_lat.leq(ctxt_level, arg_level)) {
     throw new Error('IFlow Error');
  } else {
     return true; }}
```

Listing 1.6. IFlow signature for the appendChild function

For the XMLHttpRequest ***send*** method, the default IFlow policy prevents the sending of high-level information to low-level servers. The *enforce* function to achieve this goal is similar to the one for the *appendChild* method.

Examples that make use of external interfaces. In this example, we consider a standard lattice for integrity [21] such that $H \leq L$, where H represents high integrity and L represents low integrity (untrusted). The program below creates a *text* node with untrusted content and tries to append it to a high integrity *div* node. According to the IFlow policy of Listing 6, this program is illegal. The DOM is regarded as an information sink, for this reason the properties of DOM objects cannot be automatically upgraded.

```
var low_integrity_string, text_node, div;
ugpVar(low_integrity_string, 'L');
low_integrity_string = readUntrustedSource();
text_node = document.createTextNode(low_integrity_string);
div = document.createElement('div');
div.appendChild(text_node);
```

Listing 1.7. An example concerning integrity of the DOM

The program below schedules the execution of a piece of code that updates the value of a low confidentiality variable, depending on the value of a high variable. This constitutes a sensitive upgrade. The IFlow signature for *setTimeout* must do more than compile the code that is to be executed, it must wrap it in a function literal and set the default level of its program counter to the current pc level. By doing this, when this code is finally executed, the pc level will be high and the assignment will be prevented.

```
var x = 0;
if (h) {
   setTimeout('x = 4', 2000);}
```

Listing 1.8. An example using setTimeout

The program below makes use of the XMLHttpRequest object to send a confidential cookie to an untrusted server through a POST http message. In the IFlow signature the *updtObjLevel* function of the *open* external method downgrades all the properties of the *xhr* object to L. The property *cookie* of document is by default labeled with H in the confidentiality lattice. Then, the *enforce* function prevents the illegal flow.

```
var xhr;
xhr = new XMLHttpRequest();
xhr.open("POST", untrusted_url);
xhr.send(document.cookie);
```

Listing 1.9. An example of cookie stealing

7 Implementation and Dealing with Untrusted Code

The compiler prototype is implemented in JavaScript and is available online at [1] together with a broad set of examples that includes those of the paper. We discuss here implementation details regarding implicit type coercions and randomization.

Untrusted code. The correctness of the instrumentation relies on the assumption that the internal variables and object properties used by the compiler do not intercept with those of the program to be compiled. Naturally, a malicious program may try to surpass the compiler by rewriting some of its internal variables. For example an attacker program $\_lat = permissive\_lattice$ redefines $\_lat$, which is assumed to be bound to the object representing the security lattice. Therefore, the attacker code is allowed to trigger information flows otherwise forbidden. In order to prevent this kind of malicious behaviour, variable names used by the compiler are randomly generated. In order to tamper with the compiler internal state, the attacker code must be able to guess the randomly generated names. The names of internal local variables (like $\_pc$) are randomly generated, whereas those of global variables (like $\_lat$) are assumed to be properties of a single object that is accessed through a global variable whose name is also randomly generated. In this way, the runtime libraries that are assumed to be available during the execution of compiled code do not need to be dynamically computed. *Type coercions.* Malicious code can exploit implicit type coercions to compromise the security of compiled code, as one can see in the example below.

```
o1.toString = function() { return 'p'; };
o2.p = secret;
public = o2[o1];
```

Listing 1.10. Malicious Code Example - Exploiting Implicit Type Coercions

Our instrumentation disallows any kind of implicit type coercion. Since relying in implicit type coercions is considered a bad programming practice that is error-prone and hinders maintainability, we do not find this restriction a serious shortcoming of the compiler. *Native functions.* The compiler correctness does not rely on any kind of function that is liable to malicious code, namely native functions.

```
o.p = 0;
upgStruct(o, H);
o.hasOwnProperty = function () { return false}
if(h) {
   o.p = 1;}
```

Listing 1.11. Malicious Code Example - Tampering with native functions

The example above is illegal, because updating the value of a low property in a high context constitutes a sensitive upgrade. Creating a new property in a high context is, however, allowed. Hence, the compiler must test if the object defines the property that is being set in order to decide which constraint to apply. To this end, one could use the object *hasOwnProperty* method directly, which would make the correctness of the compiler dependent on its semantics. This approach would entail a security violation, since malicious code can redefine the *hasOwnProperty* method, thus modifying its original semantics. Instead of using the object's *hasOwnProperty* method, the compiler uses a different one that is provided in the runtimes and thus accessed through a global variable whose name is randomly generated:

```
_runtime.hasOwnProperty = function(o, p) {
   var o = {};
```

```
3     return o.hasOwnProperty.call(o, p); }
```

Listing 1.12. Malicious Code Example - Internal *hasOwnProperty* method

8 Conclusion

We have presented a sound information flow monitor based on labeling of properties of objects instead of values of properties. The inlining of such a monitor generates a linear number of additional objects w.r.t. original code for compiled code (instrumented code). Having implemented a prototype [1] of the instrumentation, we show its effectiveness in a number of examples including examples that use external libraries such as the DOM API. The correctness and security of the instrumentation relies on the assumption that property names used by the compiler do not overlap with those of the original code. In the implementation, we remove this assumption by randomizing property names. It would be interesting to define a stronger noninterference property for active attackers [15] and use the techniques of [4] for the randomizing compiler to prove a stronger result w.r.t. untrusted code in mashups [23, 24]. The type system discussed in Section 5 can be used as an optimization within the compiler to type the portions of code that can be statically analyzed. This is left as future work.

References

1. Information flow instrumentation compiler. http://www-sop.inria.fr/indes/ifJS.
2. Thomas H. Austin and Cormac Flanagan. Efficient purely-dynamic information flow analysis. In Stephen Chong and David A. Naumann, editors, *PLAS*, pages 113–124. ACM, 2009.
3. Thomas H. Austin and Cormac Flanagan. Multiple Facets for Dynamic Information Flow. In *Proceedings of the ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, 2012.
4. Thomas H. Austin, Cormac Flanagan, and Martín Abadi. A functional view of imperative information flow. In Ranjit Jhala and Atsushi Igarashi, editors, *APLAS*, volume 7705 of *Lecture Notes in Computer Science*, pages 34–49. Springer, 2012.
5. Anindya Banerjee and David A. Naumann. Secure information flow and pointer confinement in a java-like language. In *CSFW*, pages 253–. IEEE Computer Society, 2002.
6. Nataliia Bielova, Dominique Devriese, Fabio Massacci, and Frank Piessens. Reactive non-interference for a browser model. In *NSS*, pages 97–104, 2011.
7. Aaron Bohannon, Benjamin C. Pierce, Vilhelm Sjöberg, Stephanie Weirich, and Steve Zdancewic. Reactive noninterference. In *ACM Conference on Computer and Communications Security*, pages 79–90, 2009.
8. Gérard Boudol. Formal aspects in security and trust. chapter Secure Information Flow as a Safety Property, pages 20–34. Springer-Verlag, Berlin, Heidelberg, 2009.
9. Cenzic Inc. Web application security trends report Q1, 2013. `http://www.cenzic.com/`, 2013.
10. Andrey Chudnov and David A. Naumann. Information flow monitor inlining. In *CSF*, pages 200–214. IEEE Computer Society, 2010.
11. D. Crockford. Adsafe. http://www.adsafe.org.
12. Dominique Devriese and Frank Piessens. Noninterference through secure multi-execution. In *IEEE Symposium on Security and Privacy*, pages 109–124, 2010.
13. ECMA. ECMAScript Language Specification. Technical report, ECMA, 2009. http://www.ecma-international.org/.
14. The FaceBook Team: FBJS. http://wiki.developers.facebook.com/index.php/FBJS.
15. Cédric Fournet and Tamara Rezk. Cryptographically sound implementations for typed information-flow security. In George C. Necula and Philip Wadler, editors, *POPL*, pages 323–335. ACM, 2008.
16. Philippa Gardner, Gareth Smith, Mark J. Wheelhouse, and Uri Zarfaty. Dom: Towards a formal specification. In *PLAN-X*, 2008.
17. Daniel Hedin and Andrei Sabelfeld. Information-Flow Security for a Core of JavaScript. In *Proceedings of the IEEE Computer Security Foundations Symposium*, 2012.
18. Arnaud Le Hors, Philippe Le Hegaret, Gavin Nicol, Jonathan Robie, Mike Champion, and Steve Byrne. Document Object Model (DOM) level 2 Core Specification. Technical report, W3C, November 2000.

19. Dongseok Jang, Ranjit Jhala, Sorin Lerner, and Hovav Shacham. An Empirical Study of Privacy-Violating Information Flows in JavaScript Web Applications. In *Proceedings of the ACM Conference on Computer and Communications Security*, pages 270–283, 2010.
20. Gurvan Le Guernic. *Confidentiality Enforcement Using Dynamic Information Flow Analyses*. PhD thesis, Kansas State University, 2007.
21. Peng Li, Yun Mao, and Steve Zdancewic. Information integrity policies. In *1st Workshop on Formal Aspects in Security and Trust (FAST)*, September 2003.
22. Jay Ligatti, Lujo Bauer, and David Walker. Enforcing non-safety security policies with program monitors. Technical Report TR-720-05, Princeton University, January 2005.
23. Mike Ter Louw, Karthik Thotta Ganesh, and V. N. Venkatakrishnan. Adjail: Practical enforcement of confidentiality and integrity policies on web advertisements. In *USENIX Security Symposium*, pages 371–388. USENIX Association, 2010.
24. Zhengqin Luo and Tamara Rezk. Mashic compiler: Mashup sandboxing based on inter-frame communication. In Stephen Chong, editor, *CSF*, pages 157–170. IEEE, 2012.
25. Sergio Maffeis, John C. Mitchell, and Ankur Taly. Object capabilities and isolation of untrusted web applications. In *IEEE Symposium on Security and Privacy*, pages 125–140. IEEE Computer Society, 2010.
26. Sergio Maffeis and Ankur Taly. Language-based isolation of untrusted javascript. In *CSF*, pages 77–91. IEEE Computer Society, 2009.
27. Jonas Magazinius, Alejandro Russo, and Andrei Sabelfeld. On-the-fly inlining of dynamic security monitors. In Kai Rannenberg, Vijay Varadharajan, and Christian Weber, editors, *SEC*, volume 330 of *IFIP Advances in Information and Communication Technology*, pages 173–186. Springer, 2010.
28. Jonas Magazinius, Alejandro Russo, and Andrei Sabelfeld. On-the-fly inlining of dynamic security monitors. *Computers & Security*, 31(7):827–843, 2012.
29. Joe Gibbs Politz, Spiridon Aristides Eliopoulos, Arjun Guha, and Shriram Krishnamurthi. Adsafety: Type-based verification of javascript sandboxing. In *USENIX Security Symposium*. USENIX Association, 2011.
30. Google Caja Team. Google-caja: A source-to-source translator for securing javascript-based web. http://code.google.com/p/google-caja/.
31. Tachio Terauchi and Alexander Aiken. Secure information flow as a safety problem. In Chris Hankin and Igor Siveroni, editors, *SAS*, volume 3672 of *Lecture Notes in Computer Science*, pages 352–367. Springer, 2005.
32. V. N. Venkatakrishnan, Wei Xu, Daniel C. DuVarney, and R. Sekar. Provably correct runtime enforcement of non-interference properties. In Peng Ning, Sihan Qing, and Ninghui Li, editors, *ICICS*, volume 4307 of *Lecture Notes in Computer Science*, pages 332–351. Springer, 2006.

A Information Flow Monitor Semantics

Figures 8, 9, and 10 give the rules for $\Downarrow_{IF}$.

$$\textsc{Property Update}$$
$$\frac{\begin{array}{c} r_s, pc \vdash \langle \mu, e_1, \Gamma\rangle \Downarrow_{IF} \langle \mu_1, r_1, \Gamma_1, \sigma_1\rangle \qquad r_s, pc \vdash \langle \mu_1, e_2, \Gamma_1\rangle \Downarrow_{IF} \langle \mu_2, m_2, \Gamma_2, \sigma_2\rangle \\ r_s, pc \vdash \langle \mu_2, e_3, \Gamma_2\rangle \Downarrow_{IF} \langle \mu_3, v_3, \Gamma_3, \sigma_3\rangle \qquad m_2 \in \mu_3(r_1) \qquad \sigma_1 \sqcup \sigma_2 \leq \Gamma_3(r_1, m_2) \\ \Gamma_4 \doteq \Gamma_3\,[(r_1, m_2) \mapsto \sigma_1 \sqcup \sigma_2 \sqcup \sigma_3] \qquad \mu_4 \doteq \mu_3\,[(r_1, m_2) \mapsto v_3] \end{array}}{r_s, pc \vdash \langle \mu, e_1[e_2] = e_3, \Gamma\rangle \Downarrow_{IF} \langle \mu_4, v_3, \Gamma_4, \sigma_3\rangle}$$

$$\textsc{Property Creation}$$
$$\frac{\begin{array}{c} r_s, pc \vdash \langle \mu, e_1, \Gamma\rangle \Downarrow_{IF} \langle \mu_1, r_1, \Gamma_1, \sigma_1\rangle \qquad r_s, pc \vdash \langle \mu_1, e_2, \Gamma_1\rangle \Downarrow_{IF} \langle \mu_2, m_2, \Gamma_2, \sigma_2\rangle \\ r_s, pc \vdash \langle \mu_2, e_3, \Gamma_2\rangle \Downarrow_{IF} \langle \mu_3, v_3, \Gamma_3, \sigma_3\rangle \qquad m_2 \notin \mu_3(r_1) \qquad \mu_4 \doteq \mu_3\,[(r_1, m_2) \mapsto v_3] \\ \sigma_1 \sqcup \sigma_2 \leq \Gamma_3(r_1) \qquad \Gamma_4 \doteq \Gamma_3\,[(r_1, m_2) \mapsto \sigma_1 \sqcup \sigma_2 \sqcup \sigma_3, r_1 \mapsto \Gamma_3(r_1) \sqcup \sigma_1 \sqcup \sigma_2] \end{array}}{r_s, pc \vdash \langle \mu, e_1[e_2] = e_3, \Gamma\rangle \Downarrow_{IF} \langle \mu_4, v_3, \Gamma_4, \sigma_3\rangle}$$

$$\textsc{Property Lookup}$$
$$\frac{\begin{array}{c} r_s, pc \vdash \langle \mu, e_1, \Gamma\rangle \Downarrow_{IF} \langle \mu_1, r_1, \Gamma_1, \sigma_1\rangle \qquad r_s, pc \vdash \langle \mu_1, e_2, \Gamma_1\rangle \Downarrow_{IF} \langle \mu_2, m_2, \Gamma_2, \sigma_2\rangle \\ \langle \mu_2, r_1, m_2, \Gamma_2\rangle\ \mathcal{R}_{Proto}\ \langle r', \sigma'\rangle \end{array}}{r_s, pc \vdash \langle \mu, e_1[e_2], \Gamma\rangle \Downarrow_{IF} \langle \mu_2, \mu_2(r', m_2), \Gamma_2, \sigma_1 \sqcup \sigma_2 \sqcup \sigma'\rangle}$$

$$\textsc{Variable}$$
$$\frac{\langle \mu, r_s, x\rangle\ \mathcal{R}_{Scope}\ r_x \qquad r_x \neq null}{r_s, pc \vdash \langle \mu, x, \Gamma\rangle \Downarrow_{IF} \langle \mu, \mu(r_x, x), \Gamma, \Gamma(r_x, x) \sqcup pc\rangle}$$

$$\textsc{Assignment - 1}$$
$$\frac{\begin{array}{c} r_s, pc \vdash \langle \mu, e, \Gamma\rangle \Downarrow_{IF} \langle \mu_1, v_1, \Gamma_1, \sigma_1\rangle \qquad \langle \mu, r_s, x\rangle\ \mathcal{R}_{Scope}\ r_x \qquad r_x \neq null \\ pc \leq \Gamma_1(r_x, x) \qquad \Gamma_2 \doteq \Gamma_1\,[(r_x, x) \mapsto \sigma_1] \qquad \mu_2 \doteq \mu_1\,[(r_x, x) \mapsto v_1] \end{array}}{r_s, pc \vdash \langle \mu, x = e, \Gamma\rangle \Downarrow_{IF} \langle \mu_2, v_1, \Gamma_2, \sigma_1\rangle}$$

$$\textsc{Assignment - 2}$$
$$\frac{\begin{array}{c} r_s, pc \vdash \langle \mu, e, \Gamma\rangle \Downarrow_{IF} \langle \mu_1, v_1, \Gamma_1, \sigma_1\rangle \qquad \langle \mu, r_s, x\rangle\ \mathcal{R}_{Scope}\ null \qquad pc \leq \Gamma_1(\#global) \\ \Gamma_2 \doteq \Gamma_1\,[(\#global, x) \mapsto \sigma_1] \qquad \mu_2 \doteq \mu_1\,[(\#global, x) \mapsto v_1] \end{array}}{r_s, pc \vdash \langle \mu, x = e, \Gamma\rangle \Downarrow_{IF} \langle \mu_2, v_1, \Gamma_2, \sigma_1\rangle}$$

$$\textsc{Function Literal}$$
$$\frac{\begin{array}{c} o_f \doteq [@fscope \mapsto r_s, @code \mapsto \lambda x.s, prototype \mapsto \#objProt] \qquad r_f \notin \mathsf{dom}(\mu) \qquad \mu' \doteq \mu\,[r_f \mapsto o_f] \\ \Gamma' \doteq \Gamma\,[(r_f, @fscope) \mapsto pc, (r_f, @code) \mapsto pc, (r_f, prototype) \mapsto pc] \end{array}}{r_s, pc \vdash \langle \mu, \mathsf{function}(x)\{s\}, \Gamma\rangle \Downarrow_{IF} \langle \mu', r_f, \Gamma', pc\rangle}$$

Fig. 8. Monitored Semantics of JavaScript Expressions - 1

B Auxiliary Lemmas for Theorem 1

Properties of the β-Equatility. In the following, $\mathcal{O}bj|_R$ corresponds to the set of objects that only use references in R (where $R \subseteq \mathcal{R}ef$). Denoting by $\mathcal{P}se$, the set $\mathcal{O}bj \cup \mathcal{V}al$, we use $\mathcal{P}se|_R$ for $\mathcal{O}bj|_R \cup \mathcal{V}al$. Lemmas 25, 11, and 27 establish that for every set of references $R \subseteq \mathcal{R}ef$, the β-Equatility is an *equivalence relation* on $\mathcal{P}se|_R$.

FUNCTION CALL

$$\frac{\begin{array}{c} r_s, pc \vdash \langle \mu, e_0, \Gamma\rangle \Downarrow_{IF} \langle \mu_0, r_0, \Gamma_0, \sigma_0\rangle \qquad r_s, pc \vdash \langle \mu_0, e_1, \Gamma_0\rangle \Downarrow_{IF} \langle \mu_1, v_1, \Gamma_1, \sigma_1\rangle \\ pc' \doteq \Gamma_1(r_0, @fscope) \qquad r'_s \doteq \mu_1(r_0, @fscope) \qquad \lambda x.s \doteq \mu_1(r_0, @code) \\ r''_s \notin \mathsf{dom}(\mu_1) \qquad \Gamma' \doteq \Gamma_1\,[(r''_s, x) \mapsto \sigma_1, (r''_s, @scope) \mapsto pc' \sqcup \sigma_0, r''_s \mapsto pc' \sqcup \sigma_0] \\ o_s \doteq [x \mapsto v_1, @scope \mapsto r'_s] \qquad \mu' \doteq \mu_1\,[r''_s \mapsto o_s] \qquad r''_s, pc' \sqcup \sigma_0 \vdash \langle \mu', s, \Gamma'\rangle \Downarrow_{IF} \langle \mu'', v, \Gamma'', \sigma'\rangle \end{array}}{r_s, pc \vdash \langle \mu, e_0(e_1), \Gamma\rangle \Downarrow_{IF} \langle \mu'', v, \Gamma'', \sigma'\rangle}$$

METHOD CALL

$$\frac{\begin{array}{c} r_s, pc \vdash \langle \mu, e_0, \Gamma\rangle \Downarrow_{IF} \langle \mu_0, r_0, \Gamma_0, \sigma_0\rangle \qquad r_s, pc \vdash \langle \mu_0, e_1, \Gamma_0\rangle \Downarrow_{IF} \langle \mu_1, m_1, \Gamma_1, \sigma_1\rangle \\ r_s, pc \vdash \langle \mu_1, e_2, \Gamma_1\rangle \Downarrow_{IF} \langle \mu_2, v_2, \Gamma_2, \sigma_2\rangle \qquad \langle \mu_2, r_0, m_1, \Gamma_2\rangle\ \mathcal{R}_{Proto}\ \langle r_m, \sigma_m\rangle \\ pc' \doteq \Gamma_2(r_m, @fscope) \qquad r'_s \doteq \mu_2(r_m, m_1)(@fscope) \qquad \lambda x.s \doteq \mu_2(r_m, m_1)(@code) \\ o_s \doteq [x \mapsto v_2, @scope \mapsto r'_s, @this \mapsto r_0] \qquad r''_s \notin \mathsf{dom}(\mu_2) \qquad \mu' \doteq \mu_2\,[r''_s \mapsto o_s] \\ \sigma' \doteq pc' \sqcup \sigma_0 \sqcup \sigma_1 \sqcup \sigma_m \qquad \Gamma'' \doteq \Gamma'\,[(r''_s, x) \mapsto \sigma_2, (r''_s, @scope) \mapsto \sigma', (r''_s, @this) \mapsto \sigma_0, r''_s \mapsto \sigma'] \\ r''_s, \sigma' \vdash \langle \mu', s, \Gamma''\rangle \Downarrow_{IF} \langle \mu'', v, \Gamma''', \sigma''\rangle \end{array}}{r_s, pc \vdash \langle \mu, e_0[e_1](e_2), \Gamma\rangle \Downarrow_{IF} \langle \mu'', v, \Gamma''', \sigma''\rangle}$$

CONSTRUCTOR CALL

$$\frac{\begin{array}{c} r_s, pc \vdash \langle \mu, e_0, \Gamma\rangle \Downarrow_{IF} \langle \mu_0, r_0, \Gamma_0, \sigma_0\rangle \qquad r_s, pc \vdash \langle \mu_0, e_1, \Gamma_0\rangle \Downarrow_{IF} \langle \mu_1, v_1, \Gamma_1, \sigma_1\rangle \\ pc' \doteq \Gamma_1(r_0, @fscope) \qquad r'_s \doteq \mu_1(r_0, @fscope) \qquad \lambda x.s \doteq \mu_1(r_0, @code) \qquad r_p \doteq \mu_1(r_0, prototype) \\ \sigma_p \doteq \Gamma_1(r_0, prototype) \qquad o \doteq [@proto \mapsto r_p] \qquad r_o \notin \mathsf{dom}(\mu_1) \qquad r''_s \notin \mathsf{dom}(\mu') \\ o_s \doteq [x \mapsto v_1, @scope \mapsto r'_s, @this \mapsto r_o] \qquad \mu' \doteq \mu_1\,[r_o \mapsto o, r''_s \mapsto o_s] \\ \Gamma' \doteq \Gamma_1\,[(r_o, @proto) \mapsto pc \sqcup \sigma_p, r_o \mapsto \sigma_0, (r''_s, x) \mapsto \sigma_1, (r''_s, @scope) \mapsto \sigma_0, (r''_s, @this) \mapsto \sigma_0, r''_s \mapsto \sigma_0] \\ r''_s, pc' \sqcup \sigma_0 \vdash \langle \mu', s, \Gamma'\rangle \Downarrow_{IF} \langle \mu'', v, \Gamma'', \sigma'\rangle \end{array}}{r_s, pc \vdash \langle \mu, \mathsf{new}\ e_0(e_1), \Gamma\rangle \Downarrow_{IF} \langle \mu'', v, \Gamma'', \sigma'\rangle}$$

THIS - 1

$$\frac{@this \in \mu(r_s) \qquad r_{this} \doteq \mu(r_s, @this) \qquad \sigma_{this} \doteq \Gamma(r_s, @this)}{r_s, pc \vdash \langle \mu, this, \Gamma\rangle \Downarrow_{IF} \langle \mu, r_{this}, \Gamma, pc \sqcup \sigma_{this}\rangle}$$

THIS - 2

$$\frac{@this \notin \mu(r_s)}{pc, r_s \vdash \langle \mu, this, \Gamma\rangle \Downarrow \langle \mu, \#global, \Gamma, pc\rangle}$$

BIN OPERATOR

$$\frac{r_s, pc \vdash \langle \mu, e_1, \Gamma\rangle \Downarrow_{IF} \langle \mu_1, v_1, \Gamma_1, \sigma_1\rangle \qquad r_s, pc \vdash \langle \mu, e_2, \Gamma\rangle \Downarrow_{IF} \langle \mu_2, v_2, \Gamma_2, \sigma_2\rangle}{r_s, pc \vdash \langle \mu, e_1\ \mathsf{op}_2\ e_2, \Gamma\rangle \Downarrow_{IF} \langle \mu_2, \Downarrow_{\mathsf{op}_2}(v_1, v_2), \Gamma_2, \sigma_1 \sqcup \sigma_2\rangle}$$

VALUE

$$r_s, pc \vdash \langle \mu, v, \Gamma\rangle \Downarrow_{IF} \langle \mu, v, \Gamma, pc\rangle$$

Fig. 9. Monitored Semantics of JavaScript Expressions - 2

VARIABLE UPGRADE

$$\frac{\langle \mu, r_s, x\rangle \ \mathcal{R}_{Scope}\ r_x \neq undef \qquad pc \leq \Gamma(r_x, x)}{r_s, pc \vdash \langle \mu, \mathsf{upgVar}(x, \sigma), \Gamma\rangle \Downarrow_{IF} \langle \mu, undef, \Gamma\left[(r_x, x) \mapsto \Gamma(r_x, x) \sqcup \sigma\right], pc\rangle}$$

PROPERTY UPGRADE

$$\frac{\langle \mu, r_s, o\rangle \ \mathcal{R}_{Scope}\ r_s^o \neq undef \qquad \langle \mu, \mu(r_s^o, o), p, \Gamma\rangle \ \mathcal{R}_{Proto}\ \langle r_p, \sigma'\rangle \qquad pc \sqcup \Gamma(r_s^o, o) \sqcup \sigma' \leq \Gamma(r_p, p)}{r_s, pc \vdash \langle \mu, \mathsf{upgProp}(o, p, \sigma), \Gamma\rangle \Downarrow_{IF} \langle \mu, undef, \Gamma\left[(r_p, p) \mapsto \Gamma(r_p, p) \sqcup \sigma\right], pc\rangle}$$

STRUCTURE UPGRADE

$$\frac{\langle \mu, r_s, o\rangle \ \mathcal{R}_{Scope}\ r_s^o \neq undef \qquad r_o \doteq \mu(r_s^o, o) \qquad pc \sqcup \Gamma(r_s^o, o) \leq \Gamma(r_o)}{r_s, pc \vdash \langle \mu, \mathsf{upgStruct}(o, \sigma), \Gamma\rangle \Downarrow_{IF} \langle \mu, undef, \Gamma\left[r_o \mapsto \Gamma(r_o) \sqcup \sigma\right], pc\rangle}$$

SEQ

$$\frac{r_s, pc \vdash \langle \mu, s_1, \Gamma\rangle \Downarrow_{IF} \langle \mu_1, v_1, \Gamma_1, \sigma_1\rangle \qquad r_s, pc \vdash \langle \mu_1, s_2, \Gamma_1\rangle \Downarrow_{IF} \langle \mu_2, v_2, \Gamma_2, \sigma_2\rangle}{r_s, pc \vdash \langle \mu, s_1; s_2, \Gamma\rangle \Downarrow_{IF} \langle \mu_2, v_2, \Gamma_2, \sigma_2\rangle}$$

IF - 1

$$\frac{\begin{array}{c} r_s, pc \vdash \langle \mu, e, \Gamma\rangle \Downarrow_{IF} \langle \mu, v', \Gamma', \sigma'\rangle \qquad v' \notin \{0, false, undef, null\} \\ r_s, pc \sqcup \sigma' \vdash \langle \mu', s_1, \Gamma'\rangle \Downarrow_{IF} \langle \mu'', v'', \Gamma'', \sigma''\rangle \end{array}}{r_s, pc \vdash \langle \mu, \mathsf{if}(e)\{s_1\}\mathsf{else}\{s_2\}, \Gamma\rangle \Downarrow_{IF} \langle \mu'', v'', \Gamma'', \sigma''\rangle}$$

IF - 2

$$\frac{\begin{array}{c} r_s, pc \vdash \langle \mu, e, \Gamma\rangle \Downarrow_{IF} \langle \mu, v', \Gamma', \sigma'\rangle \qquad v' \in \{0, false, undef, null\} \\ r_s, pc \sqcup \sigma' \vdash \langle \mu', s_2, \Gamma'\rangle \Downarrow_{IF} \langle \mu'', v'', \Gamma'', \sigma''\rangle \end{array}}{r_s, pc \vdash \langle \mu, \mathsf{if}(e)\{s_1\}\mathsf{else}\{s_2\}, \Gamma\rangle \Downarrow_{IF} \langle \mu'', v'', \Gamma'', \sigma''\rangle}$$

WHILE-1

$$\frac{r_s, pc \vdash \langle \mu, e, \Gamma\rangle \Downarrow_{IF} \langle \mu', v', \Gamma', \sigma'\rangle \qquad v' \in \{0, false, undef, null\}}{r_s, pc \vdash \langle \mu, \mathsf{while}(e)\{s\}, \Gamma\rangle \Downarrow_{IF} \langle \mu', undef, \Gamma', pc\rangle}$$

WHILE-2

$$\frac{\begin{array}{c} r_s, pc \vdash \langle \mu, e, \Gamma\rangle \Downarrow_{IF} \langle \mu', v', \Gamma', \sigma'\rangle \qquad v' \notin \{0, false, undef, null\} \\ r_s, pc \sqcup \sigma' \vdash \langle \mu', s, \Gamma'\rangle \Downarrow_{IF} \langle \mu'', v'', \Gamma'', \sigma''\rangle \qquad r_s, pc \vdash \langle \mu'', \mathsf{while}(e)\{s\}, \Gamma''\rangle \Downarrow_{IF} \langle \mu''', v''', \Gamma''', \sigma'''\rangle \end{array}}{r_s, pc \vdash \langle \mu, \mathsf{while}(e)\{s\}, \Gamma\rangle \Downarrow_{IF} \langle \mu''', v''', \Gamma''', \sigma'''\rangle}$$

Fig. 10. Monitored Semantics of JavaScript Statements

Lemma 10 (Reflexivity of $\sim_{\mathsf{id}}$). *Given a set of references $R \subseteq \mathcal{R}ef$, then for every pseudo value $v \in \mathcal{P}se|_R$, it follows that $v \sim_{\mathsf{id}} v$, where id is the identity function defined on R.*

Proof. The proof proceeds by case analysis.

- $v \in \mathcal{P}rim$. Applying the Rule PRIM, the result immediately follows.
- $v \in \Lambda$. Applying the Rule FUN, the result immediately follows.
- $v \in R$. Since id is defined on R, we conclude that $v = \mathsf{id}(v)$. Hence applying the Rule REFERENCE, the result immediately follows.
- $v \in \mathcal{O}bj|_R$. For every property $p \in \mathsf{dom}(v)$, $v(p) \in \mathcal{P}rim \cup R \cup \Lambda$. Hence, we can use the three first cases to conclude that $v(p) \sim_{\mathsf{id}} v(p)$. Thus, using Rule OBJECT, the result follows.

Lemma 11 (Symmetry of $\sim_\beta$). *Given an injective function β and two pseudo values $v_1, v_2 \in \mathcal{P}se$ such that: $v_1 \sim_\beta v_2$, then $v_2 \sim_{\beta^{-1}} v_1$.*

Proof. The proof proceeds by case analysis.

- $v_1, v_2 \in \mathcal{P}rim$. By the Rule VALUE, we conclude that $v_1 = v_2$, and hence $v_2 = v_1$. Thus, using the same rule, the result follows.
- $v_1, v_2 \in \Lambda$. By the Rule FUN, we conclude that $v_1 = v_2$, and hence $v_2 = v_1$. Thus, using the same rule, the result follows.
- $v_1, v_2 \in \mathcal{R}ef$. The Rule REFERENCE guarantees that $v_2 = \beta(v_1)$. Since β is injective, β^{-1} is defined on the range of β and particularly, $\beta^{-1}(v_2) = v_1$, from which the result follows (applying the same rule).
- $v_1, v_2 \in \mathcal{O}bj$. The Rule OBJECT guarantees that $\mathsf{dom}(v_1) = \mathsf{dom}(v_2) = P$ and that for every property $p \in P$, $v_1(p) \sim_\beta v_2(p)$. Since $v_1(p)$ and $v_2(p)$ are either in $\mathcal{P}rim$, in $\mathcal{R}ef$, or in Λ, we use the three first cases to conclude that: $v_2(p) \sim_{\beta^{-1}} v_1(p)$ for every property $p \in P$, thus proving the result.

Lemma 12 (Transitivity of $\sim_\beta$). *Given two injective functions $\beta_1, \beta_2 : \mathcal{R}ef \hookrightarrow \mathcal{R}ef$ and three pseudo-values $v_1, v_2, v_3 \in \mathcal{P}se$ such that: $v_1 \sim_{\beta_1} v_2$ and $v_2 \sim_{\beta_2} v_3$, then $v_1 \sim_{\beta_1 \circ \beta_2} v_3$.*

Proof. There are three cases to consider:

- $v_1, v_2, v_3 \in \mathcal{P}rim$. Rule VALUE guarantees that $v_1 = v_2$ and $v_2 = v_3$, which means that $v_1 = v_3$ and therefore: $v_1 \sim_{\beta_1 \circ \beta_2} v_3$.
- $v_1, v_2, v_3 \in \Lambda$. Rule FUN guarantees that $v_1 = v_2$ and $v_2 = v_3$, which means that $v_1 = v_3$ and therefore: $v_1 \sim_{\beta_1 \circ \beta_2} v_3$.
- $v_1, v_2, v_3 \in \mathcal{R}ef$. Rule REFERENCE guarantees that $v_2 = \beta_1(v_1)$ and $v_3 = \beta_2(v_2)$. Hence, $v_3 = \beta_2(\beta_1(v_1)) = \beta_1 \circ \beta_2(v_1)$, from which follows that: $v_1 \sim_{\beta_1 \circ \beta_2} v_3$.
- $v_1, v_2, v_3 \in \mathcal{O}bj$. Rule OBJECT guarantees that $\mathsf{dom}(v_1) = \mathsf{dom}(v_2) = \mathsf{dom}(v_3) = P$. For every property $p \in P$, $v_1(p) \sim_{\beta_1} v_2(p)$ and $v_2(p) \sim_{\beta_2} v_3(p)$. Since $v_1(p)$, $v_2(p)$ and $v_3(p)$ are either in $\mathcal{P}rim$, in $\mathcal{R}ef$, or in Λ, we use the two first cases to conclude that: $v_1(p) \sim_{\beta_1 \circ \beta_2} v_3(p)$ for every property $p \in P$, thus proving the result.

Lemma 13. *Given four pseudo values $v_1, v_1', v_2, v_2' \in \mathcal{P}se$ such that $v_1 \sim_{\mathsf{id}_1} v_1'$, $v_2 \sim_{\mathsf{id}_2} v_2'$ and $v_1 \sim_\beta v_2$, where id_1 and id_2 correspond to the identity mapping on references defined on two (possibly different) arbitrary domains. Then, it follows that $v_1' \sim_\beta v_2'$.*

Proof. The proof proceeds by case analysis.

- $v_1, v_1', v_2, v_2' \in \mathcal{P}rim$. Rule VALUE guarantees that $v_1 = v_2$, $v_1 = v_1'$, and $v_2 = v_2'$, from which follows that $v_1' = v_2'$ and therefore: $v_1' \sim_\beta v_2'$.
- $v_1, v_1', v_2, v_2' \in \mathcal{R}ef$. Rule REFERENCE guarantees that $v_2 = \beta(v_1)$, $v_1 = v_1'$, and $v_2 = v_2'$. Hence, $v_2' = \beta(v_1')$ and therefore $v_1' \sim_\beta v_2'$.
- $v_1, v_1', v_2, v_2' \in \mathcal{O}bj$. Rule OBJECT guarantees that $\mathsf{dom}(v_1) = \mathsf{dom}(v_1') = \mathsf{dom}(v_2) = \mathsf{dom}(v_2') = P$. For every property $p \in P$, $v_1(p) \sim_\beta v_2(p)$, $v_1(p) \sim_{\mathsf{id}_1} v_1'(p)$, and $v_1(p) \sim_\beta v_2(p)$. Since $v_1(p)$, $v_1'(p)$, $v_2(p)$, and $v_2'(p)$ are either in $\mathcal{P}rim$, in $\mathcal{R}ef$, or in Λ, we can apply the current lemma to conclude that: $v_1'(p) \sim_\beta v_2'(p)$ for every property $p \in P$, thus proving the result.

Low Equality Properties The following lemma is used in the proof of the transitivity of the low equality relation.

Lemma 14. *Given four security levels $\sigma_1, \sigma_2, \sigma_3, \sigma \in \mathcal{L}$ and three sets of references $R_1, R_2, R_3 \subseteq \mathcal{R}ef$, such that:*

$$\big((\sigma_1 \leq \sigma) \vee (\sigma_2 \leq \sigma)\big) \Rightarrow \big(\sigma_1, \sigma_2 \leq \sigma \wedge R_1 = R_2\big) \tag{29a}$$
$$\big((\sigma_2 \leq \sigma) \vee (\sigma_3 \leq \sigma)\big) \Rightarrow \big(\sigma_2, \sigma_3 \leq \sigma \wedge R_2 = R_3\big) \tag{29b}$$

Then it follows that:

$$\big((\sigma_1 \leq \sigma) \vee (\sigma_3 \leq \sigma)\big) \Rightarrow \big(\sigma_1, \sigma_3 \leq \sigma \wedge R_1 = R_3\big) \tag{29c}$$

Proof. Assume that $\sigma_1 \leq \sigma$, it follows from Hypothesis 29a that $\sigma_2 \leq \sigma$ and $R_1 = R_2$. From the fact that $\sigma_2 \leq \sigma$, we apply Hypothesis 29b to conclude that $\sigma_3 \leq \sigma$ and $R_2 = R_3$. Hence, assuming that $\sigma_1 \leq \sigma$, we conclude that $\big(\sigma_1, \sigma_3 \leq \sigma \wedge R_1 = R_3\big)$. In a similar way, assuming that $\sigma_3 \leq \sigma$, we conclude that $\big(\sigma_1, \sigma_3 \leq \sigma \wedge R_1 = R_3\big)$. Therefore, the claim of the lemma holds.

The following two lemmas establish the transitivity and the reflexivity (when β corresponds to the identity function) of the low equality relation.

Lemma 15 (Transitivity). *Given memories μ_1, μ_2 and μ_3, labelings Γ_1, Γ_2 and Γ_3, a security level σ and two functions β_{12} and β_{23} such that:*

$$\mu_1, \Gamma_1 \approx_{\beta_{12},\sigma} \mu_2, \Gamma_2 \tag{30a}$$
$$\mu_2, \Gamma_2 \approx_{\beta_{23},\sigma} \mu_3, \Gamma_3 \tag{30b}$$

Then it follows that: $\mu_1, \Gamma_1 \approx_{\beta_{23} \circ \beta_{12},\sigma} \mu_3, \Gamma_3$

Proof. For every reference $r \in \mathsf{dom}(\beta_{23} \circ \beta_{12})$, we have to prove that:

$$\{p \in \mathsf{dom}(\mu_1(r))) \mid \Gamma_1(r,p) \leq \sigma\} = \{p \in \mathsf{dom}(\mu_3(r'')) \mid \Gamma_3(r'',p) \leq \sigma\} = P \tag{30c}$$
$$\mu_1(r)|_P \sim_{\beta_{23} \circ \beta_{12}} \mu_3(r'')|_P \tag{30d}$$
$$\Gamma_1|_{r,P} = \Gamma_3|_{r'',P} \tag{30e}$$
$$\big(\Gamma_1(r) \leq \sigma \vee \Gamma_3(r'') \leq \sigma\big) \Rightarrow \big(\mathsf{dom}(\mu_1(r)) = \mathsf{dom}(\mu_3(r'')) \wedge \Gamma_1(r), \Gamma_3(r'') \leq \sigma\big) \tag{30f}$$

where: $r'' = \beta_{23} \circ \beta_{12}(r)$. Suppose that $r \in \mathsf{dom}(\beta_{23} \circ \beta_{12})$, it follows that:

$$r \in \mathsf{dom}(\beta_{12}) \tag{30g}$$
$$r' = \beta_{12}(r) \in \mathsf{dom}(\beta_{23}) \tag{30h}$$

From the Hypothesis 30a and Equation 30g, we conclude that:

$$\{p \in \mathsf{dom}(\mu_1(r))) \mid \Gamma_1(r,p) \leq \sigma\} = \{p \in \mathsf{dom}(\mu_3(\beta_{12}(r)))) \mid \Gamma_2(\beta_{12}(r),p) \leq \sigma\} = P' \tag{30i}$$
$$\mu_1(r)|_{P'} \sim_{\beta_{12}} \mu_2(\beta_{12}(r))|_{P'} \tag{30j}$$
$$\Gamma_1|_{r,P'} = \Gamma_2|_{\beta_{12}(r),P'} \tag{30k}$$
$$\big(\Gamma_1(r) \leq \sigma \vee \Gamma_2(\beta_{12}(r)) \leq \sigma\big) \Rightarrow \big(\mathsf{dom}(\mu_1(r)) = \mathsf{dom}(\mu_3(r')) \wedge \Gamma_1(r), \Gamma_2(\beta_{12}(r)) \leq \sigma\big) \tag{30l}$$

From the Hypothesis 30b and Equation 30h, we conclude that:

$$\{p \in \mathsf{dom}(\mu_2(\beta_{12}(r)))) \mid \Gamma_2(\beta_{12}(r),p) \leq \sigma\} = \{p \in \mathsf{dom}(\mu_3(r'')) \mid \Gamma_3(r'',p) \leq \sigma\} = P'' \tag{30m}$$
$$\mu_2(\beta_{12}(r))|_{P''} \sim_{\beta_{23}} \mu_3(r'')|_{P''} \tag{30n}$$
$$\Gamma_2|_{\beta_{12}(r),P''} = \Gamma_3|_{r'',P''} \tag{30o}$$
$$\big(\Gamma_2(\beta_{12}(r)) \leq \sigma \vee \Gamma_3(r'') \leq \sigma\big) \Rightarrow \big(\mathsf{dom}(\mu_2(\beta_{12}(r))) = \mathsf{dom}(\mu_3(r')) \wedge \Gamma_2(\beta_{12}(r)), \Gamma_3(r') \leq \sigma\big) \tag{30p}$$

From Equations 30i and 30m, Claim 30c follows by noting that $P = P' = P''$. Thus, applying Lemma 12 to Equations 30j and 30n, Claim 30d follows. Claim 30e is an immediate consequence of the fact that $P = P' = P''$ and of Equations 30k and 30o. Applying Lemma 14 to Equations 30l and 30p, Claim 30f follows.

Lemma 16 (Reflexivity of $\approx_{\mathsf{id},\sigma}$). *For any security level $\sigma \in \mathcal{L}$, two consistent memories μ and μ' and two labelings Γ and Γ', such that $\mu \leq \mu'$ and $\Gamma \leq \Gamma'$, then: $\mu, \Gamma \approx_{\mathsf{id},\sigma} \mu', \Gamma'$, where id is the identity function defined on the domain of μ.*

Proof. One has to prove for every reference $r \in \mathsf{dom}(\mu)$ that:

$$\{p \in \mathsf{dom}(\mu(r))) \mid \Gamma(r,p) \leq \sigma\} = \{p \in \mathsf{dom}(\mu'(r)) \mid \Gamma'(r,p) \leq \sigma\} = P \tag{31a}$$

$$\mu(r)|_P \sim_{\mathsf{id}} \mu'(r)|_P \tag{31b}$$

$$\Gamma|_{r,P} = \Gamma'|_{r,P} \tag{31c}$$

$$\big(\Gamma(r) \leq \sigma \vee \Gamma'(r) \leq \sigma\big) \Rightarrow \big(\mathsf{dom}(\mu(r)) = \mathsf{dom}(\mu'(r)) \wedge \Gamma(r), \Gamma'(r) \leq \sigma\big) \tag{31d}$$

Since, by hypothesis, $\mu \leq \mu'$ and $r \in \mathsf{dom}(\mu)$, it follows that $\mu(r) = \mu'(r)$ and, therefore, $\mathsf{dom}(\mu(r)) = \mathsf{dom}(\mu'(r))$. Thus, considering the hypothesis that $\Gamma \leq \Gamma'$, Claim 31a immediately follows. Since P is defined and $\mu(r) = \mu'(r)$, it follows that $\mu(r)|_P = \mu'(r)|_P$. Moreover, since μ is consistent, we get that $\mu(r)|_P \in \mathcal{O}bj|_{\mathsf{dom}(\mu)} = \mathcal{O}bj|_{\mathsf{dom}(\mathsf{id})}$. Hence, applying Lemma 25, Claim 31b immediately follows. Claim 31c follows directly from the fact that P is defined and $\Gamma \leq \Gamma'$. To prove claim 31d, we note that since $\mu \leq \mu'$ and $\Gamma \leq \Gamma'$, it follows that $\mathsf{dom}(\mu(r)) = \mathsf{dom}(\mu'(r))$ and $\Gamma(r) = \Gamma'(r)$. Hence, either $\Gamma(r), \Gamma'(r) \leq \sigma$ and $\mathsf{dom}(\mu(r)) = \mathsf{dom}(\mu'(r))$, or $\Gamma(r), \Gamma'(r) \not\leq \sigma$.

The following three lemmas establish three different sets of conditions that if verified when updating two memories that are initially low equal, ensure that final memories are also low equal.

Lemma 17 (High Property Update). *Given a security level $\sigma \in \mathcal{L}$ and two consistent memories μ and μ' respectively labeled by Γ and Γ' such that μ coincides with μ' everywhere except for some reference r and property p for which $\sigma \not\leq \Gamma(r,p)$ and $\sigma \not\leq \Gamma'(r,p)$; then: $\mu, \Gamma \approx_{\beta,\sigma} \mu', \Gamma'$.*

Lemma 18 (Low-Equality Preserving Property Update/Creation). *Given three security levels $\sigma, \sigma_1, \sigma_2 \in \mathcal{L}$, two consistent memories μ_1 and μ_2 respectively labeled by Γ_1 and Γ_2, two references $r_1, r_2 \in \mathcal{R}ef$, two values $v_1, v_2 \in \mathcal{V}al$ and partial injective function on references β, such that: $r_1 \sim_\beta r_2$, $\mu_1, \Gamma_1 \approx_{\beta,\sigma} \mu_2, \Gamma_2$, and $(v_1 \sim_\beta v_2 \wedge \sigma_1 = \sigma_2) \vee \sigma_1, \sigma_2 \leq \sigma$. Then, it follows that $\mu_1', \Gamma_1' \approx_{\beta,\sigma} \mu_2', \Gamma_2'$, for $\mu_1' = \mu_1\,[(r_1, m) \mapsto v_1]$, $\mu_2' = \mu_2\,[(r_2, m) \mapsto v_2]$, $\Gamma_1' = \Gamma_1\,[(r_1, m) \mapsto \sigma_1]$, and $\Gamma_2' = \mu_2\,[(r_2, m) \mapsto v_2]$.*

Given an object $o \in \mathcal{O}bj$ pointed by a reference r in a memory μ labeled by Γ and a security level $\sigma \in \mathbf{L}$, the notation $o|_\sigma^{r,\Gamma}$ is used for the restriction of o to its low properties. Formally: $o|_\sigma^{r,\Gamma} \stackrel{def}{=} o|_P$, where $P = \{p \in \mathsf{dom}(o) | \Gamma(r,p) \leq \sigma\}$.

Lemma 19 (Low-Equality Preserving Object Creation). *Given a security level $\sigma \in \mathcal{L}$, four consistent memories μ_1, μ_1', μ_2, and μ_2' respectively well-labeled by Γ_1, Γ_1', Γ_2, and Γ_2', two references $r_1, r_2 \in \mathcal{R}ef$ respectively free in μ_1 and μ_2, , two objects $o_1, o_2 \in \mathcal{O}bj$, and a partial injective function β defined on $\mathcal{R}ef$, such that: $\mu_1, \Gamma_1 \approx_{\beta,\sigma} \mu_2, \Gamma_2$, $\mu_1' = \mu_1\,[r_1 \mapsto o_1]$, $\mu_2' = \mu_2\,[r_1 \mapsto o_1]$, and $o_1|_\sigma^{r_1,\Gamma_1} \sim_\beta o_2|_\sigma^{r_2,\Gamma_2}$. Then: $\mu_1', \Gamma_1' \approx_{\beta',\sigma} \mu_2', \Gamma_2'$, where $\beta' = \beta \cup \{(r_1, r_2)\}$.*

Lemma 20. *Given a security level $\sigma \in \mathcal{L}$ and four consistent memories μ_1, μ_1', μ_2, and μ_2' respectively labeled by Γ_1, Γ_1', Γ_2, and Γ' and a partial injective function β defined on $\mathcal{R}ef$, such that:*

$$\mu_1, \Gamma_1 \approx_{\mathsf{id}_1,\sigma} \mu_1', \Gamma_1'$$
$$\mu_2, \Gamma_2 \approx_{\mathsf{id}_2,\sigma} \mu_2', \Gamma_2'$$
$$\mu_1, \Gamma_1 \approx_{\beta,\sigma} \mu_2, \Gamma_2$$

Where id_1 and id_2 correspond to the identity function defined on the domain of μ_1 and the identity function defined on the domain of μ_2 respectively. Then, it follows that: $\mu_1', \Gamma_1' \approx_{\beta,\sigma} \mu_2', \Gamma_2'$.

Auxiliar lemmas concerning monitored execution The following lemma states that the *reading effect* of an expression is always higher the program counter.

Lemma 21. *Given a program s, a memory μ, a labeling Γ, a security level pc and a reference r_s such that:*

$$r_s, pc \vdash \langle \mu, s, \Gamma \rangle \Downarrow_{IF} \langle \mu', v, \Gamma', \sigma \rangle$$

for some memory μ', value v, labeling Γ' and security level σ; then: $pc \leq \sigma$.

Proof. Straightforward by induction on the derivation of $r_s, pc \vdash \langle \mu, s, \Gamma \rangle \Downarrow_{IF} \langle \mu', v, \Gamma', \sigma \rangle$.