From Static to Hybrid Information Flow Control in a Core of JavaScript

José Fragoso Santos, Thomas Jensen, Tamara Rezk, Alan Schmitt

Inria
firstname.lastname@inria.fr

Abstract. We propose and prove sound a novel type system for securing information flow in a core of JavaScript. This core takes into account the defining features of the language, such as prototypical inheritance, extensible objects, and constructs that check the existence of object properties. We design a hybrid version of the proposed type system. This version infers a set of assertions under which a program can be securely accepted and instruments it so as to dynamically check whether these assertions hold. By deferring rejection to runtime, the hybrid version can typecheck secure programs that purely static type systems cannot accept.

1 Introduction

Client-side JavaScript programs often include untrusted code dynamically loaded from third-party code providers, such as online advertisers. This issue raises the need for enforcement mechanisms that isolate trusted code from that coming from other sources. For instance, trusted programs must not be allowed to leak secret values to untrusted parties. The absence of security leaks can be formally expressed using one of the many flavors of noninterference (NI) [12], which is a class of properties that have been classically used to define secure information flow. Here, we aim at enforcing termination-insensitive NI, meaning that programs are assumed not to leak information through their termination behavior.

When designing an information flow enforcement mechanism, it is vital that the "abstractions made in the attacker model be adequate with respect to potential attacks" [22]. In the context of this work, this means that security policies should be rich enough to capture the main attacks coming from JavaScript code. Hence, we present a type language for expressing security policies in JavaScript that takes into account the defining features of the language, such as prototypical inheritance, extensible objects, and constructs to check the existence of object properties. We then design a type system (TS) for statically verifying that a program abides by the specified policy.

One of the major issues in developing static analyses for JavaScript is the fact that "property names can be computed using string operations" [16], which renders intractable the problem of deciding at the static level which property is actually being accessed in a given property look-up. Consider the following program `o` = {}, `o.prop_A` = 0, `o.prop_B` = 1, `o["prop_"+ f()]` (where the + stands

for string concatenation) that creates an object `o` with two properties `prop_A` and `prop_B` that it assigns to 0 and 1, and then tries to read one of them depending on the output of `f`. In this example, deciding which property is being accessed is equivalent to predicting the dynamic behavior of function `f`, which is, in general, undecidable. In order to overcome this issue, previous analyses for enforcing confinement properties in JavaScript (such as that of [16]) have chosen to restrict the targeted language subset, excluding property look-ups with arbitrary expressions.

We propose a new approach, exploiting the connections between static and runtime analysis to avoid rejecting programs that are in fact secure. The key insight of our approach is that, since we aim at enforcing **termination insensitive** noninterference, the analysis may infer a set of assertions under which a program can be securely accepted and then dynamically verify whether or not these assertions hold. The original program is instrumented in such a way that if the assertions on which it is *conditionally accepted* fail to hold, its instrumentation diverges.

2 Core JavaScript

Objects are the central datatype of JavaScript. In contrast to class-based languages where the fields of an object are restricted by the class to which it belongs (which is statically specified), a JavaScript object is an unrestricted partial mapping from strings to values. The strings in the domain of an object are called its *properties*. There are no classes, but every (non-native) object has a prototype from which it can *inherit* properties. Prototypes are also objects. Hence, prototypical inheritance is a form of delegation. In order to look-up the value of a property p of an object o, the JavaScript engine first checks whether p belongs to the set of properties of o. If so, the property look-up yields $o(p)$, otherwise the engine checks whether the prototype of o defines a property named p, and so forth. The sequence of objects that can be accessed from a given object through the inspection of the resp. prototypes is called a *prototype-chain*.

JavaScript features first-class functions. Functions can be invoked in the standard way or they can be used as *methods*. When assigning a function to a property of an object, the function becomes a *method* of the object. When calling a function as a method, the keyword `this` is bound to the receiver object. Every method accessible to an object through its prototype-chain can be called as a method of that object. For instance, if the method m is accessible to object o through its prototype-chain, when calling $o.m(\cdots)$, the keyword `this` is bound to o and not to the object that actually defines m in the prototype-chain of o. Hence, prototypical inheritance can be seen as a device for method sharing.

Another important feature of JavaScript is that programs are not only allowed to dynamically add new properties to the domain of an object, but they can also delete existing ones. A program can check whether a property is accessible from an object through its prototype-chain using the keyword `in`. Interestingly, the property look-up construct can also be used to check the existence of prop-

$e ::=$	v	value	$\mid$	$\mathsf{function}^{\dot{\tau},i}(x)\{\mathsf{var}^{\dot{\tau}_1,\cdots,\dot{\tau}_n}\ y_1,\cdots,y_n;\ e\}$	function literal
$\mid$	this	this keyword	$\mid$	$\{\}^{\dot{\tau},i}$	object literal
$\mid$	$e_0\ \mathsf{op}^i\ e_1$	binary operation	$\mid$	$e_0(e_1)^i$	function call
$\mid$	x^i	variable	$\mid$	$e_0[e_1,P](e_2)^i$	method call
$\mid$	$x = e$	variable assignment	$\mid$	$e_0,\ e_1$	sequence
$\mid$	$e_0[e_1,P]^i$	property look-up	$\mid$	$e_0\ ?^i\ (e_1) : (e_2)$	conditional
$\mid$	$e_0[e_1,P] = e_2$	property assignment	$\mid$	$\mathsf{delete}^i\ e.p$	property deletion
$\mid$	$e_0\ \mathsf{in}_i^P\ e_1$	membership testing			

e, e_0, e_1 and e_2 represent expressions, i and j represent program indexes, x, y_1, ..., y_n represent variable names, and op represents binary operators.

Fig. 1. Syntax of Core JavaScript

erties, since the looking-up of a property that is not defined in the prototype chain of an object does not yield an error but instead `undefined`.

We define a JavaScript-like language, called Core JavaScript, whose syntax is given in Figure 1. As in [24], property look-ups, method calls, and property assignments are annotated with a set P of the properties to which the corresponding expression may evaluate, which we call a *look-up annotation*. For instance, in the expression `o[e, {"foo", "bar", "baz"}]`, the look-up annotation means that `e` always evaluates to a string equal to `"foo"`, `"bar"`, or `"baz"`. We similarly annotate the occurrences of the `in` expression with the set of properties that may be checked. A look-up annotation P is *correct* if the expression to which it applies always evaluates to a string in P. Moreover, we say that P is *minimal* if there is no other correct P' such that $P' \subset P$. It is trivial to instrument a program so that it diverges if its look-up annotations are not correct. For instance, one could easily modify the specification of the hybrid TS to ensure the correctness of look-up annotations. This would, however, clutter up the presentation. Hence, we leave it implicit and in the rest of the paper assume that look-up annotations are correct. But they do not have to be minimal – the look-up annotation corresponding to the set $\mathcal{S}tr$ of all strings is always correct. Additionally, some program constructs are annotated with unique indexes to be used by the instrumentation, and object and function literals are annotated with the corresponding security type. We say that two expressions e and e' are *equal up to look-up annotations*, written $e \equiv e'$, if they only differ in look-up annotations. Whenever a look-up annotation is omitted, it is assumed to be $\mathcal{S}tr$, and the notation `o.p` is used as an abbreviation for `o["p",{"p"}]`.

Core JavaScript is intended to model a realistic subset of the JavaScript specification [2]. However, in order to simplify the presentation, we do not model the `return` statement—functions are assumed to return the value to which their body evaluates. Furthermore, given that most implementations do allow explicit prototype mutation, we depart from [2] and include this feature through a special property `_prot_`. For instance, `o._prot_ = o_p` sets the prototype of `o` to `o_p`, and `o._prot_` evaluates to the prototype of `o`.

Figure 2 presents the running example that is used throughout the paper. It consists of a fragment of the code for a simple contact management online application. The `CM` variable holds the *Contact Manager* object. The contact manager stores contacts in an object bound to its property `contact_list`, which is used as a table whose entries are the last names of the contacts (extended with

```
CM = {}, CM.proto_contact = {}, CM.contact_list = {},
CM.proto_contact.printContact = function() { this.lst + "," + this.fst },
CM.proto_contact.makeFavorite = function() { this.favorite = null },
CM.proto_contact.isFavorite = function() { "favorite" in this },
CM.proto_contact.unFavorite = function() {
   "favorite" in this ? delete this.favorite : true },
CM.createContact = function(fst_name, lst_name, email) { var contact;
  contact = {}, contact.__proto__ = proto_contact, contact.fst = fst_name,
  contact.lst = lst_name, contact.email = email, contact },
CM.storeContact = function(contact, i) {
   var list, key; list = this.contact_list, key = contact.lst+i,
   key in list ? CM.storeContact(contact, i+1) : list[key] = contact }
```

Fig. 2. A Simple Contact Manager

unique integers to avoid collisions) and whose values are the actual contacts. A contact is simply an object containing a first name (`fst`), a last name (`lst`), an e-mail address (`email`), and a flag `favorite`. This example illustrates the typical use of prototypical inheritance in JavaScript. We create a "fixed" object bound to the property `proto_contact` of `CM` that stores all the methods contact objects are assumed to implement and every time a contact object is created, its prototype is set to `CM.proto_contact`. Hence, every contact object implements the methods: (1) `printContact` (that generates a string with a description of the contact), (2) `makeFavorite` (that marks the contact as favorite), (3) `isFavorite` (that checks whether the contact is marked as favorite), and (4) `unFavorite` (that deletes the property that marks the contact as favorite).

Formal Semantics. We model objects as partial functions mapping strings to values in the set $\mathcal{P}rim \cup \mathcal{R}ef \cup \mathcal{F}_\lambda$ containing all primitive values, references, and parsed function literals. The set $\mathcal{P}rim$ includes strings, numbers, and booleans, and two special values: `null` and `undefined`. References can be viewed as pointers to objects, in the sense that every expression that creates an object yields a new reference that points to it. As in [7], we assume a *parametric object allocator*. The properties reserved for the internal use of the semantics are prefixed with an "@". We use $dom(o)$ for the set of properties of o (excluding internal properties). A memory $\mu : \mathcal{R}ef \mapsto \mathcal{S}tr \mapsto \mathcal{P}rim \cup \mathcal{R}ef \cup \mathcal{F}_\lambda$ is a mapping from references to objects [2]. In the following, we assume that the binding of variables is modeled using *scope objects* [15] and that the evaluation of a function literal triggers the creation of a *function object* storing its parsed counterpart (in the property `@code`) and a reference to the scope object that was *active* at the time of its evaluation (in the property `@fscope`). We use a relation $\mathcal{R}_{Scope}$ for modeling the variable look-up procedure. If $\langle \mu, r, x \rangle \ \mathcal{R}_{Scope} \ r'$ then r' is the reference of the scope object that is closest to $\mu(r)$ in its scope chain and that defines a binding for x. Thus, $\mu(r')(x)$ is the value associated with x in that scope. We assume that memories include a reference to a special object called *global object*, pointed to by a fixed reference $\#glob$ that binds global variables. We make use of a big-step semantics for Core JavaScript $\Downarrow$ (given in Appendix A) with the following shape: $r \vdash \langle \mu, \Sigma, e \rangle \Downarrow \langle \mu', \Sigma', v \rangle$, where: (1) r is the reference of the active scope object, (2) μ and μ' are the initial and final memories, (3) Σ and Σ' are initial and final labelings, (4) e is the expression to evaluate, and (5) v is the computed value. A *labeling* $\Sigma : \mathcal{R}ef \to \mathcal{T}$ is a mapping from references to security types. Upon the evaluation of a function/object literal of type $\dot{\tau}$, the semantics extends the current labeling Σ with a new mapping from the newly created reference to the corresponding type.

The specification of security policies usually relies on two key elements: a lattice of security levels and a labeling that maps resources to security levels. In the examples, we use $\mathcal{L} = \{H, L\}$ with LH, meaning that resources labeled with L (*low*) are less confidential than those labeled with H (*high*). Hence, H-labeled resources may depend on L-labeled resources, but not the contrary, as that would entail a *security leak*. We use , , $\bot$, and $\top$ for the greatest lower bound (*glb*), the least upper bound (*lub*), the *bottom* level, and the *top* level, respectively.

3 Challenges for IFC in Core JavaScript

Before proceeding to the formal definition of secure information flow in Core JavaScript, we review the main challenges imposed by the particular features of the language.

Extensible Objects. In JavaScript, the programmer can dynamically add and remove properties from objects. In fact, objects are commonly used as tables whose keys are computed at runtime. Hence, in many contexts, it is not realistic to expect the programmer to statically know the properties of the objects that are created at runtime. However, security-wise, the programmer often knows the security level of the contents of an object even when its actual properties are unknown. For instance, in the Contact Manager example, the precise structure of `contact_list` cannot be statically known because the last names in its domain are dynamic inputs. Nevertheless, the programmer should be allowed to specify that the value associated with every property of `contact_list` is of type Contact.

Leaks via Prototype Mutations. The fact that a prototype of an object is allowed to change at runtime may be exploited to encode security leaks. For instance, suppose that the first and last names of a contact are of level L and that we create a new object, bound to `CM.proto_contact_new`, to be used as the prototype of contact objects, that prints contacts in a different way:

```
CM.proto_contact_new.printContact = function(){this.fst +" "+ this.lst}
```

The output of `printContact` is *low* for the original and new methods, since, in both cases, it only discloses information at level L. However, the expression:

```
h ? (c._proto_ = CM.proto_contact_new) : (null), l = c.printContact()
```

is illegal because, depending on the value of a *high* variable `h`, it changes the prototype of `c` and therefore the behavior of `printContact`, which is supposed to generate a *low* output. Concretely, depending on the value of `h`, the attacker sees the contact printed $last\_name$, $first\_name$ or $first\_name\ last\_name$. To tackle this problem, we forbid the prototype of a *low* object to change in a *high* context and therefore the assignment `c._prot_` = `CM.proto_contact_new` is considered illegal by the enforcement mechanism.

Leaks via the Checking of the Existence of Properties. A program can dynamically add and remove properties from objects. Furthermore, a program can check whether a property is defined in the prototype-chain of an object using the keyword `in`. Thus, the mere existence of a property in the domain of an object

may disclose confidential information. As in [14], we associate every property in the domain of an object with an *existence level*. For instance, suppose that the user of the contact manager does not want to disclose which are his favorite contacts. In this case, the existence level of property `favorite` must be set to H. However, the fact that a property is confidential does not imply that its existence is confidential. Suppose that the e-mail address associated with each contact is of level H. This does not mean that the existence level of the property `email` should be set to H.

Leaks via the Global Object. During the execution of a function call, the keyword `this` is bound to the global object, whose properties are the global variables of the program. Hence, it is possible to encode illegal information flows regarding confidential global variables using the keyword `this` inside a function. For instance, the program `function(){ l = this.cookie}()` produces the same effect as `l = cookie`. In order to detect this type of leaks, the security type of a function that may be called directly (as opposed to as a method) must use the type of the global object as the type of `this` and the type of the global object should be used as the global typing environment. For instance, in the Contact Manager example, the global object defines a property `CM` bound to the contact manager. Hence, the global typing environment should map the global variable `CM` to the type of the property `CM` of the global object.

Pre-methods and Recursive Types. In contrast to class-based languages, where method types are specified inside their classes, JavaScript functions are first-class values which can be defined anywhere in the code and later assigned to properties of arbitrary objects. This creates a dependency between types for functions and types for objects, because object types include the types of their methods and function types include the type of the objects to which the keyword `this` is bound during execution. To break this circularity, we make use of equi-recursive types. However, to keep the presentation fairly simple, we restrict the occurrence of type variables to the type of `this` in function types.

3.1 Security Types for Core JavaScript

Every security type $\dot{\tau} = \tau^{\sigma}$ is obtained by pairing up a *raw* type τ with a security level σ, that gives an upper bound on the levels of the resources on which the values of that type may depend.[1] Let p, σ, and κ range over the sets of strings, security levels, and type variables. The syntax of raw types is as follows.

$$\tau ::= \textsc{prim} \mid \langle \dot{\tau}.\dot{\tau} \xrightarrow{\sigma} \dot{\tau} \rangle \mid \langle \kappa.\dot{\tau} \xrightarrow{\sigma} \dot{\tau} \rangle \mid \mu\kappa.\langle p^{\sigma} : \dot{\tau}, \cdots, p^{\sigma} : \dot{\tau}, *^{\sigma} : \dot{\tau} \rangle \mid \mu\kappa.\langle p^{\sigma} : \dot{\tau}, \cdots, p^{\sigma} : \dot{\tau} \rangle$$

We denote by $\mathcal{T}$ the set of all security types. Given a security type $\dot{\tau}$, $lev(\dot{\tau})$ denotes its level and $\lfloor \dot{\tau} \rfloor$ its raw type. For instance, $lev(\textsc{prim}^{L}) = L$ and $\lfloor \textsc{prim}^{L} \rfloor = \textsc{prim}$. We define $\dot{\tau}^{\sigma}$ as $\lfloor \dot{\tau} \rfloor^{lev(\dot{\tau}) \sqcup \sigma}$. Hence, $(\textsc{prim}^{L})^{H} = \textsc{prim}^{H}$. A typing environment Γ is a mapping from variables to types.

[1] For instance, a primitive value of type $\textsc{prim}^{L}$ may only depend on *low* resources. The same applies to an object o of type $\mu\kappa.\langle p^{L} : \textsc{prim}^{H} \rangle^{L}$. However, the value associated with o's property p may depend on *high* resources.

$$\dot{\tau}_{contact} = \mu\kappa.\left\langle \begin{array}{c} fst^L : \textsf{PRIM}^L, lst^L : \textsf{PRIM}^L, id^L : \textsf{PRIM}^H, printContact^L : \langle \kappa.\textsf{PRIM}^L \xrightarrow{H} \textsf{PRIM}^L \rangle^L, \\ makeFavorite^L : \langle \kappa.\textsf{PRIM}^L \xrightarrow{H} \textsf{PRIM}^L \rangle^L, isFavorite^L : \langle \kappa.\textsf{PRIM}^L \xrightarrow{H} \textsf{PRIM}^H \rangle^L, \\ unFavorite^L : \langle \kappa.\textsf{PRIM}^L \xrightarrow{H} \textsf{PRIM}^H \rangle^L, favorite^H : \textsf{PRIM}^H, \_proto\_^L : \dot{\tau}_{proto\_contact} \end{array} \right\rangle^L$$

$$\dot{\tau}_{CM} = \mu\kappa.\left\langle \begin{array}{c} proto\_contact^L : \dot{\tau}_{contact}, contact\_list^L : \mu\kappa.\langle *^L : \dot{\tau}_{contact}, L \rangle^L, \\ create\_contact^L : \langle \kappa.(\textsf{PRIM}^L, \textsf{PRIM}^L, \textsf{PRIM}^H) \xrightarrow{L} \dot{\tau}_{contact} \rangle^L, \\ store\_contact^L : \langle \kappa.(\dot{\tau}_{contact}, \textsf{PRIM}^L) \xrightarrow{L} \dot{\tau}_{contact} \rangle^L, \_proto\_^L : \textsf{PRIM}^L \end{array} \right\rangle^L$$

Fig. 3. Typing Environment for the Contact Manager - $\Gamma_{CM} = [CM \mapsto \dot{\tau}_{CM}]$

The type PRIM is the type of all primitive values. The type $\langle \dot{\tau}_0.\dot{\tau}_1 \xrightarrow{\sigma} \dot{\tau}_2 \rangle$ is the type of all functions that map values of type $\dot{\tau}_1$ to values of type $\dot{\tau}_2$ and during the executiton of which the keyword `this` is bound to an object of type $\dot{\tau}_0$. The level σ is the *writing effect* [22] of the function, i.e., a lower bound on the levels of the resources created/updated during its execution. The type $\mu\kappa.\langle p_0^{\sigma_0} : \dot{\tau}_0, \cdots, p_n^{\sigma_n} : \dot{\tau}_n, *^{\sigma_*} : \dot{\tau}_* \rangle$ is the type of all objects that **potentially** define properties $p_0, \cdots, p_n$, mapping each property p_i to a value of type $\dot{\tau}_i$. The type assigned to the $*$ is the *default type*. Every property p_i is additionally associated with an *existence level* σ_i. The level σ_* is the *default existence level*. We use the notation $dom(\dot{\tau})$ for the set containing the properties that appear in $\dot{\tau}$ (including $*$ if it is present), and the notation $*(\dot{\tau})$ for the pair $(\sigma_*, \dot{\tau}_*)$ consisting of the default existence level and security type of $\dot{\tau}$. The fact that an object has type $\dot{\tau}$ does not mean that it defines all properties in $dom(\dot{\tau})$, but rather that it **potentially** defines the properties in $dom(\dot{\tau})$. Moreover, if $* \notin dom(\dot{\tau})$, then o is assumed to be *non-extensible*, meaning that only properties in $dom(\dot{\tau})$ can be added to o. This is enforced in Rule [Property Assignment] in Figure 5 that checks whether the look-up annotation is contained in the domain of the corresponding object type, whenever that type does not include the $*$. Figure 3 presents a typing environment for the Contact Manager example. We omit the specification of the type $\dot{\tau}_{proto\_contact}$ that coincides with $\dot{\tau}_{contact}$ in every property except in `_prot_` for which it does not define a mapping, since objects of that type are not supposed to have a prototype.[2]

Another important aspect of object types is that they must reflect the whole prototype-chain accessible through the corresponding objects. Hence, in the Contact Manager example, the security type assigned to contact objects also includes the methods that the corresponding prototype implements.

3.2 The Attacker Model and the Meaning of Security Types

In order to formally characterize the "observational power" of an attacker, we take the standard approach of defining a notion of *low projection* of a memory at a given level σ [17], which corresponds to the part of the memory that an attacker at level σ can observe. This defintion makes use of a function $\Rsh$ that receives as input an object security type $\dot{\tau}$ and a string p and outputs a pair consisting of the existence level and the security type with which $\dot{\tau}$ associates p:

$$\Rsh(\dot{\tau}, p) = \begin{cases} (\sigma_i, \{\dot{\tau}/\kappa\}\dot{\tau}_i) \text{ if } \dot{\tau} = \mu\kappa.\langle \cdots, p_i^{\sigma_i} : \dot{\tau}_i, \cdots \rangle^\sigma \wedge p = p_i \\ (\sigma_*, \{\dot{\tau}/\kappa\}\dot{\tau}_*) \text{ if } \dot{\tau} = \mu\kappa.\langle \cdots, *^{\sigma_*} : \dot{\tau}_*, \cdots \rangle^\sigma \wedge p \notin dom(\dot{\tau}) \end{cases}$$

[2] In the specification every object has an implicit prototype: `Object.prototype`.

where $\{\dot{\tau}_0/\kappa\}\dot{\tau}_1$ denotes the capture-avoiding substitution of κ for $\dot{\tau}_0$ in $\dot{\tau}_1$. Interestingly, given an object type $\dot{\tau}$, if we define $\Pi^{\prime}\,(\dot{\tau}) : \mathcal{S}tr \to \mathcal{T}$, as the function that maps every identifier p to the second element of $\Pi^{\prime}\,(\dot{\tau}, p)$, one can interpret an object type as a typing environment. Indeed, programs must be typed in a typing environment matching the type of the *global object* $\dot{\tau}_{glob}$, meaning that: if $\Gamma(x) = \dot{\tau}_x$, then $\Pi^{\prime}\,(\dot{\tau}_{glob}, x) = (\sigma', \dot{\tau}_x)$.

Informally, an attacker at level σ can see: **(1)** the references whose corresponding object types are annotated with levels $\sqsubseteq \sigma$, **(2)** all of the values that are reachable from visible properties in visible references and are annotated with levels $\sqsubseteq \sigma$, **(3)** the existence of visible properties in visible references, and **(4)** the code of visible function objects as well as the low-projections of their corresponding scope-chains. Since every function object in memory is associated with the scope object that was active at the time of its evaluation, the low-equality must take into account the scope-chains that are stored in memory. To this end, we make use of a function $\mathcal{S}Chain$ that given a memory and a reference to a function object outputs the corresponding scope-chain and we extend the definition of low-projection to scope-chains. However, due to lack of space and since this issue is mainly connected with technical details regarding the way we model the semantics of the language, we give these two definitions in Appendix B.

Definition 1 (Low-Projection and Low-Equality). *The low-projection of a memory μ w.r.t. a security level σ and a labeling Σ is given by:*

$$\begin{aligned}\mu \restriction^{\Sigma,\sigma} = \ & \{(r, \Sigma(r)) \mid lev(\Sigma(r)) \sqsubseteq \sigma\} \\ & \cup \{(r, p, v) \mid \Pi^{\prime}\,(\Sigma(r), p) = (\sigma', \dot{\tau}) \wedge \sigma' \sqcup lev(\dot{\tau}) \sqcup lev(\Sigma(r)) \sqsubseteq \sigma \wedge p \in dom(\mu(r))\} \\ & \cup \{(r, p) \mid \Pi^{\prime}\,(\Sigma(r), p) = (\sigma', \dot{\tau}) \wedge \sigma' \sqcup lev(\Sigma(r)) \sqsubseteq \sigma \wedge p \in dom(\mu(r))\} \\ & \cup \{(r, f, \mathcal{S}Chain(\mu, r) \restriction^{\sigma}) \mid lev(\Sigma(r)) \sqsubseteq \sigma \wedge f = \mu(r)(@code)\}\end{aligned}$$

Two memories μ_0 and μ_1, respectively labeled by Σ_0 and Σ_1 are said to be low-equal at security level σ, written $\mu_0, \Sigma_0 \sim_\sigma \mu_1, \Sigma_1$ if they coincide in their respective low-projections, $\mu_0 \restriction^{\Sigma_0,\sigma} = \mu_1 \restriction^{\Sigma_1,\sigma}$.

Figure 4 presents the memory resulting from the execution of the program below:

```
x = CM.createContact("Jane", "Doe", "jane.d@gmail.com"),
y = CM.createContact("John", "Doe", "john.d@gmail.com"),
CM.storeContact(x, 0), CM.storeContact(y, 0), makeFavorite(x)
```

together with its low-projections at level L for the typing environment given in Figure 3 (Γ_{CM}). Remark that, while the values of both e-mail addresses disappear, their existence remains visible. In contrast, the property `favorite` is removed from the contact object of Jane.

Informally, a program is *noninterferent* (NI) if its execution on two low-equal memories always produces two low-equal memories. Hence, an attacker cannot use a NI program as a means to disclose the confidential contents of a memory.

Definition 2 (Noninterference). *An expression e is said be noninterferent w.r.t. a typing environment Γ if for any two memories μ and μ' respectively well-labeled by Σ and Σ', if $\#glob \vdash \langle \mu, \Sigma, e\rangle \Downarrow \langle \mu_f, \Sigma_f, v\rangle$, $\#glob \vdash \langle \mu', \Sigma', e\rangle \Downarrow \langle \mu'_f, \Sigma'_f, v'\rangle$, $\mu, \Sigma \sim_\sigma \mu', \Sigma'$, and $\Gamma = \Pi^{\prime}\,(\Sigma(\#glob))$, then: $\mu_f, \Sigma_f \sim_\sigma \mu'_f, \Sigma'_f$.*

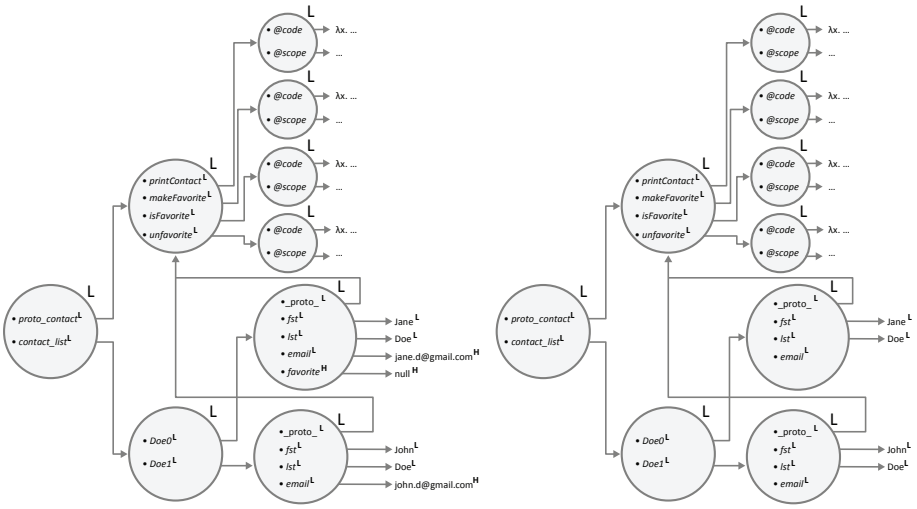

Fig. 4. A memory and its low-projection

The definition of noninterference is standard except for the requirement that the typing environment be consistent with the type of the global object. Furthermore, the initial memories are assumed to be *well-labeled*, meaning that the types of the references in memory must "match" their corresponding values. This definition is formally given in Appendix B. For simplicity, the definition of noninterference does not impose any restriction on the generated outputs. This does not constitute a problem, since any expression e that produces a *high* output can be trivially re-written as `h = e, null`.

3.3 A Type System for IFC in Core JavaScript

Figure 5 presents the proposed static TS for securing information flow in Core JavaScript. Typing judgements have the form: $\Gamma \vdash e : \dot{\tau}, \sigma$, where (1) Γ is the typing environment, (2) e the expression to be typed, (3) $\dot{\tau}$ the type that is assigned to it, and (4) σ its *writing effect*, that is, a lower bound on the levels of the resources that are updated/created when e is evaluated.

The type system assumes two basic restrictions on the syntax of security types. First, we require the existence level of a property to be lower than or equal to the level that annotates its corresponding security type. This restriction forbids the specification of an object type that associates an invisible property with a visible value. Second, we require the security level that annotates an object type to be higher than or equal to the level that annotates the type of its prototype. This constraint is meant to prevent leaks *via* prototype mutations. If the level of the prototype of an object o is *high*, then the prototype of o is allowed to change in a *high* context. However, such changes remain invisible to a *low* observer, because the level of o is itself *high*, meaning that a *low* observer can never see any of the contents of o.

In order to type expressions that either result from the combination of subexpressions with different types, or whose evaluation may yield values of different types (for instance, a property look-up with an imprecise look-up annotation), the TS makes use of an ordering on security types. The ordering $\sqsubseteq$ on security levels induces a simple ordering $\preceq$ on security types: $\dot{\tau}_0 \preceq \dot{\tau}_1$ *iff* $lev(\dot{\tau}_0) \sqsubseteq lev(\dot{\tau}_1)$ and $\lfloor \dot{\tau}_0 \rfloor \equiv \lfloor \dot{\tau}_1 \rfloor$, where $\equiv$ stands for syntactic equality up to arbitrary unfoldings

$$\textsc{Val}\quad \Gamma \vdash v : \textsf{PRIM}^{\bot}, \top \qquad \textsc{This}\quad \Gamma \vdash \textsf{this} : \Gamma(\textsf{this}), \top \qquad \textsc{Var}\quad \Gamma \vdash x : \Gamma(x), \top \qquad \textsc{Object Literal}\quad \Gamma \vdash \{\}^{\tau,i} : \dot{\tau}, lev(\tau)$$

$$\textsc{Binary Operation}\quad \frac{\forall_{i=0,1} \cdot \Gamma \vdash e_i : \dot{\tau}_i, \sigma_i}{\Gamma \vdash e_0 \ \textsf{op} \ e_1 : \dot{\tau}_0 \curlyvee \dot{\tau}_1, \sigma_0 \sqcap \sigma_1}$$

$$\textsc{Variable Assignment}\quad \frac{\Gamma \vdash e : \dot{\tau}, \sigma \qquad \dot{\tau} \preceq \Gamma(x) \qquad \sigma' = \sigma \sqcap lev(\Gamma(x))}{\Gamma \vdash x = e : \tau, \sigma'}$$

$$\textsc{Property Look-up}\quad \frac{\forall_{i=0,1} \cdot \Gamma \vdash e_i : \dot{\tau}_i, \sigma_i \qquad \Rsh_{\uparrow}(\dot{\tau}_0, P) = (\sigma, \dot{\tau}) \qquad \sigma = \sigma_0 \sqcap \sigma_1}{\Gamma \vdash e_0[e_1, P] : \dot{\tau}^{lev(\dot{\tau}_0) \sqcup lev(\dot{\tau}_1)}, \sigma}$$

$$\textsc{In Expression}\quad \frac{\forall_{i=0,1} \cdot \Gamma \vdash e_i : \dot{\tau}_i, \sigma_i \qquad \sigma = \sigma_0 \sqcap \sigma_1 \qquad \Rsh_{\uparrow}(\dot{\tau}_1, P) = (\sigma', \dot{\tau}) \qquad \sigma'' = lev(\dot{\tau}_0) \sqcup lev(\dot{\tau}_1)}{\Gamma \vdash e_0 \ \textsf{in}^P \ e_1 : \textsf{PRIM}^{\sigma' \sqcup \sigma''}, \sigma}$$

$$\textsc{Property Assignment}\quad \frac{\begin{array}{c}\forall_{i=0,1,2} \cdot \Gamma \vdash e_i : \dot{\tau}_i, \sigma_i \qquad \Rsh_{\downarrow}(\dot{\tau}_0, P) = (\sigma, \dot{\tau}) \qquad \dot{\tau}_2 \preceq \dot{\tau} \\ lev(\dot{\tau}_0) \sqcup lev(\dot{\tau}_1) \sqsubseteq \sigma \qquad * \notin dom(\dot{\tau}_0) \Rightarrow P \subseteq dom(\dot{\tau}_0)\end{array}}{\Gamma \vdash e_0[e_1, P] = e_2 : \dot{\tau}_2, \sigma_0 \sqcap \sigma_1 \sqcap \sigma_2 \sqcap \sigma}$$

$$\textsc{Function Call}\quad \frac{\Gamma \vdash e_0 : \langle \dot{\tau}_0'.\dot{\tau}_1' \xrightarrow{\hat{\sigma}} \dot{\tau}_2' \rangle^{\hat{\sigma}'}, \sigma_0 \qquad \Gamma \vdash e_1 : \dot{\tau}_1, \sigma_1 \qquad \dot{\tau}_{global} \preceq \dot{\tau}_0' \qquad \dot{\tau}_1 \preceq \dot{\tau}_1' \qquad \hat{\sigma}' \sqsubseteq \hat{\sigma}}{\Gamma \vdash e_0(e_1) : (\dot{\tau}_2')^{\hat{\sigma}'}, \sigma_0 \sqcap \sigma_1 \sqcap \hat{\sigma}}$$

$$\textsc{Method Call}\quad \frac{\begin{array}{c}\forall_{i=0,1,2} \cdot \Gamma \vdash e_i : \dot{\tau}_i, \sigma_i \qquad \Rsh_{\uparrow}(\dot{\tau}_0, P) = (\sigma, \langle \dot{\tau}_0'.\dot{\tau}_1' \xrightarrow{\hat{\sigma}} \dot{\tau}_2' \rangle^{\hat{\sigma}'}) \\ \sigma' = \hat{\sigma}' \sqcup lev(\dot{\tau}_0) \sqcup lev(\dot{\tau}_1) \qquad \dot{\tau}_0 \preceq \dot{\tau}_0' \qquad \dot{\tau}_2 \preceq \dot{\tau}_1' \qquad \sigma' \sqsubseteq \hat{\sigma}\end{array}}{\Gamma \vdash e_0[e_1, P](e_2) : (\dot{\tau}_2')^{\sigma'}, \sigma_0 \sqcap \sigma_1 \sqcap \sigma_2 \sqcap \hat{\sigma}}$$

$$\textsc{Property Deletion}\quad \frac{\Gamma \vdash e : \dot{\tau}, \sigma \qquad \Rsh(\dot{\tau}, p) = (\sigma', \dot{\tau}') \qquad lev(\dot{\tau}) \sqsubseteq \sigma'}{\Gamma \vdash \textsf{delete} \ e.p : \textsf{PRIM}^{\bot}, \sigma \sqcap \sigma'}$$

$$\textsc{Sequence}\quad \frac{\forall_{i=0,1} \cdot \Gamma \vdash e_i : \dot{\tau}_i, \sigma_i}{\Gamma \vdash e_0, e_1 : \dot{\tau}_1, \sigma_0 \sqcap \sigma_1}$$

$$\textsc{Conditional Expression}\quad \frac{\forall_{i=0,1,2} \cdot \Gamma \vdash e_i : \dot{\tau}_i, \sigma_i \qquad \sigma = \sigma_0 \sqcap \sigma_1 \sqcap \sigma_2 \qquad lev(\dot{\tau}_0) \sqsubseteq \sigma_1 \sqcap \sigma_2}{\Gamma \vdash e_0 \ ? \ (e_1) : (e_2) : (\dot{\tau}_1 \curlyvee \dot{\tau}_2)^{lev(\dot{\tau}_0)}, \sigma}$$

$$\textsc{Function Literal}\quad \frac{\tau = \langle \dot{\tau}_0'.\dot{\tau}_1' \xrightarrow{\hat{\sigma}} \dot{\tau}_2' \rangle^{\sigma} \qquad \Gamma\left[\textsf{this} \mapsto \dot{\tau}_0', x \mapsto \dot{\tau}_1', y_1 \mapsto \dot{\tau}_1, \cdots, y_n \mapsto \dot{\tau}_n\right] \vdash e : \dot{\tau}_2', \hat{\sigma}}{\Gamma \vdash \textsf{function}^{\dot{\tau}}(x)\{\textsf{var}^{\dot{\tau}_1, \cdots, \dot{\tau}_n} \ y_1, \cdots, y_n; \ e\} : \dot{\tau}, \sigma}$$

Fig. 5. Typing Secure Information Flow in Core JavaScript

of raw types [3]. Every two object security types in the subtyping relation need to have the same corresponding raw type, because, while property look-ups are *covariant* with the type of the property, property assignments are *contravariant*. Concretely, given an obect of type $\dot{\tau}_0 = \mu\kappa.\langle p^L : \textsf{PRIM}^L \rangle^L$ bound to `x` and an object of type $\dot{\tau}_1 = \mu\kappa.\langle p^L : \textsf{PRIM}^H \rangle^L$ bound to `y`, if we let $\dot{\tau}_0 \preceq \dot{\tau}_1$, the expression `y = x, y.p = h`, which is **not** noninterferent, would be typable. Given a raw type τ, the set $\{\dot{\tau} \mid \lfloor \dot{\tau} \rfloor \equiv \tau\}$ of its corresponding security types (ordered by $\preceq$) forms a lattice. The corresponding *lub* and *glb* $\curlyvee, \curlywedge : \mathcal{T} \times \mathcal{T} \to \mathcal{T}$ are defined as follows: $\dot{\tau}_0 \underset{\curlywedge}{\curlyvee} \dot{\tau}_1 = \dot{\tau} \Leftrightarrow \lfloor \dot{\tau} \rfloor \equiv \lfloor \dot{\tau}_0 \rfloor \equiv \lfloor \dot{\tau}_1 \rfloor \wedge lev(\dot{\tau}) = lev(\dot{\tau}_0) \underset{\sqcap}{\sqcup} lev(\dot{\tau}_1)$. Using the notions of *lub* and *glb* between security types, we extend function $\Rsh$ to arbitrary sets of properties in the two following ways:

$$\Rsh_{\uparrow}(\dot{\tau}, P) = (\sqcup\{\hat{\sigma} \mid p \in P \wedge \Rsh(\dot{\tau}, p) = (\hat{\sigma}, \dot{\tau}')\}, \curlyvee\{\dot{\tau}' \mid p \in P \wedge \Rsh(\dot{\tau}, p) = (\sigma, \dot{\tau}')\})$$
$$\Rsh_{\downarrow}(\dot{\tau}, P) = (\sqcap\{\hat{\sigma} \mid p \in P \wedge \Rsh(\dot{\tau}, p) = (\hat{\sigma}, \dot{\tau}')\}, \curlywedge\{\dot{\tau}' \mid p \in P \wedge \Rsh(\dot{\tau}, p) = (\sigma, \dot{\tau}')\})$$

While $\Rsh_{\uparrow}$ is used for the typing of property look-ups, in expressions, and method calls (which are covariant with the type of the corresponding property), $\Rsh_{\downarrow}$ is used for the typing of property assignments and property deletions (which are contravariant with the type of the corresponding property).

In the following, we give a brief description of the rules that better illustrate the information flows specific to Core JavaScript and refer to [22] for a comprehensive presentation of a classical TS for IFC. In the Rule [Property Assignment], the raw type of the property that is being assigned ($\lfloor \dot{\tau} \rfloor$) must coincide with the raw type of the expression to which it is being assigned ($\lfloor \dot{\tau}_2 \rfloor$). The constraint $lev(\dot{\tau}) \sqsubseteq lev(\dot{\tau}_2)$ prevents the *explicit flow* resulting from the assignment of a *high* value to a *low* property, whereas the constraint $lev(\dot{\tau}_0) \sqcup lev(\dot{\tau}_1) \sqsubseteq \sigma$

prevents the so-called *implicit flows* – one cannot create a property with a *low* existence level depending on *high* values. Moreover, one cannot update a property associated with a *low* value depending on *high* values. However, the former constraint subsumes the latter. In the Rule [Property Deletion], the security level that annotates the type of the object whose property is being deleted ($lev(\dot{\tau})$) must be lower than or equal to the existence level of that property (σ'). The reason is that one cannot delete a visible property depending on secret information. In both rules, the existence level of the property being assigned/deleted is included in the writing effect of the respective expression in order to prevent the creation/deletion of visible properties in invisible contexts. Finally, the Rule [Method Call] checks whether the types of the object ($\dot{\tau}_0$) and the argument ($\dot{\tau}_2$) match the types of the keyword `this` ($\dot{\tau}'_0$) and the formal parameter ($\dot{\tau}'_1$) of the method being invoked. The constraint $\sigma' \sqsubseteq \hat{\sigma}$ prevents the calling of a method that creates/updates *low* memory depending on *high* values. The soundness of the proposed TS is established in Theorem 1.

Theorem 1 (Noninterference). *For any expression e and typing environment Γ such that $\Gamma \vdash e : \tau, \sigma$, it holds that e is noninterferent w.r.t. Γ.*

4 A Hybrid Approach for IFC in Core JavaScript

The precision of the purely static TS heavily depends on the precision of look-up annotations. For instance, a property look-up is typable only if all properties in the corresponding look-up annotation are associated with the same raw type. In this section, we modify this TS so as to make its precision independent of the precision of look-up annotations. The key insight is that, since our goal is to verify **termination insensitive** noninterference, we can defer failure to execution time. Hence, instead of rejecting a program based on imprecise look-up annotations, the hybrid TS infers a set of assertions under which a program can be securely accepted and instruments it so as to dynamically check whether these assertions hold. The instrumented version diverges if the assertions on which the original version was *conditionally accepted* fail to hold at runtime.

In order to be able to reason about intermediate states of the execution, the TS makes use of an indexed set of variables $\mathcal{V}_{TS}$. These variables are used for bookkeeping the values of intermediate expressions and are not available for the programmer. Since one can easily instrument a program so that it diverges when trying to read/write reserved variables, we can assume that program variables do not overlap with those in $\mathcal{V}_{TS}$. The runtime assertions generated by the TS are described by the following grammar:

$$\omega ::= \hat{x}_i \in V \mid v \in V \mid \texttt{tt} \mid \omega \vee \omega \mid \omega \wedge \omega \mid \neg\omega$$

where $\hat{x}_i$ is the i-th variable of $\mathcal{V}_{TS}$ and V an arbitrary set of primitive values. We consider two types of *elementary assertions*. An elementary assertion $v \in V$ holds if the value v is contained in V. An *elementary assertion* $\hat{x}_i \in V$ holds in a memory μ in the scope-chain starting from r, written $\mu, r \vDash \hat{x}_i \in V$, if $\hat{x}_i$ is

bound to a value in V in that scope.[3] The remaining assertions are interpretated as in classical propositional logic.

In this section, we use as a running example the program $\mathtt{x[y]} = \mathtt{u[v]} + \mathtt{z}$, to be typed using the following typing environment:

$$\begin{gathered}\Gamma(x) = \mu\kappa.\langle p_0^L : \mathsf{PRIM}^H, p_1^L : \mathsf{PRIM}^L, *^L : \mathsf{PRIM}^L\rangle^L \\ \Gamma(u) = \mu\kappa.\langle q_0^L : \mathsf{PRIM}^H, q_1^L : \mathsf{PRIM}^L, *^L : \mathsf{PRIM}^H\rangle^L \\ \Gamma(z) = \Gamma(y) = \Gamma(v) = \mathsf{PRIM}^L\end{gathered}$$

This program is not typable using the static TS, because the left-hand side expression is typed with PRIM^L (since the TS uses $\Rsh_\downarrow$ to determine its type), while the right-hand side expression is typed with PRIM^H (since the TS uses $\Rsh_\uparrow$ to determine its type).[4] However, since the look-up annotations of this program are very imprecise, it can be the case that the potential illegal flows, which cause the static TS to reject it, never actually happen. Hence, instead of assigning a single security type and a single writing effect to each expression, the hybrid TS assigns it a set T of *possible* security types and a set L of *possible* writing effects. Each type $\dot{\tau}$ in T and each security level σ in L is paired up with an assertion ω that describes "when" the expression is correctly typed by $\dot{\tau}$ or has writing effect σ. For instance, the look-up expressions $\mathtt{x[y]}$ and $\mathtt{u[v]}$ are respectively typed with $T_{x[y]} = \{(\mathsf{PRIM}^H, \hat{x}_i \in \{p_0\}), (\mathsf{PRIM}^L, \hat{x}_i \in \{p_1\}), (\mathsf{PRIM}^L, \neg(\hat{x}_i \in \{p_0, p_1\}))\}$ and $T_{u[v]} = \{(\mathsf{PRIM}^L, \hat{x}_j \in \{q_1\}), (\mathsf{PRIM}^H, \hat{x}_j \in \{q_0\}), (\mathsf{PRIM}^H, \neg(\hat{x}_j \in \{q_0, q_1\}))\}$, where $\hat{x}_i$ and $\hat{x}_j$ are the variables of the TS that hold the values to which $\mathtt{y}$ and $\mathtt{v}$ evaluate in that context.

It is therefore useful to define a function $\Rsh^?$ that **expects** as input an object type $\dot{\tau}$, a set P of properties to inspect, and an expression e that evaluates to the actual property being inspected[5] and **generates** a set of triples of the form $(\sigma, \dot{\tau}', \omega)$. Each of these triples consists of a security level σ, a security type $\dot{\tau}'$, and the assertion ω that must hold so that the actual property being looked-up has existence level σ and security type $\dot{\tau}'$. Formally, letting $LT^{\dot{\tau},P,e} = \{(\sigma, \dot{\tau}', (e \in \{p\}) \mid p \in P \cap dom(\dot{\tau}) \wedge \Rsh\,(\dot{\tau}, p) = (\sigma, \dot{\tau}')\}$, $\Rsh^?$ is defined as follows:

$$\Rsh^?\,(\dot{\tau}, P, e) = \begin{cases} LT^{\dot{\tau},P,e} & \text{if } P \subseteq dom(\dot{\tau}) \\ LT^{\dot{\tau},P,e} \cup \{(\sigma_*, \tau_*, \neg(e \in dom(\dot{\tau}) \cap P))\} & \text{if } P \not\subseteq dom(\dot{\tau}) \end{cases}$$

We extend $\Rsh^?$ to sets of object security types paired up with runtime assertions in the following way: $\Rsh^?\,(T, P, e) = \{(\sigma, \dot{\tau}', \omega \wedge \omega') \mid (\dot{\tau}, \omega) \in T \ \wedge\ (\sigma, \dot{\tau}', \omega') \in \Rsh^?\,(\dot{\tau}, P, e)\}$. Given a set LT of triples of the form $(\sigma, \dot{\tau}, \omega)$, we denote by $\pi_{\mathtt{lev}}\,(LT)$ ($\pi_{\mathtt{type}}\,(LT)$, resp.) the set of pairs obtained from LT by removing from each triple the security type (the security level, resp.). Observe that $\pi_{\mathtt{type}}\left(\Rsh^?\,(\dot{\tau}_x, \mathcal{S}tr, \hat{x}_i)\right) = T_{x[y]}$ and $\pi_{\mathtt{type}}\left(\Rsh^?\,(\dot{\tau}_u, \mathcal{S}tr, \hat{x}_j)\right) = T_{u[v]}$.

Since an expression is typed with a set of security types and a set of writing effects (instead of a single type and a single writing effect), the constraints as

[3] Formally, $\langle \mu, r, \hat{x}_i\rangle\ \mathcal{R}_{Scope}\ r'$ and $\mu(r')(\hat{x}_i) \in P$.

[4] Recall that the implicit look-up annotation is $\mathcal{S}tr$.

[5] Observe that e must either be a variable of the TS or a primitive value.

well as the *lub's* and *glb's* operations of the old TS must be rewritten in order to account for this change. For instance, in the current running example, the hybrid TS types `u[v]` with $T_{u[v]}$ and `z` with $T_z = \{(\mathsf{PRIM}^L, \mathtt{tt})\}$. Therefore, in order to type `u[v] + z`, the TS needs to combine two sets of security types paired up with runtime assertions. To this end, we make use of a function $\oplus_{\Cup}$, parameterized with a generic binary function $\Cup$, that given two sets of elements paired up with runtime assertions, S_0 and S_1, generates a new set $S_0 \oplus_{\Cup} S_1$. If $(s, \omega) \in S_0 \oplus_{\Cup} S_1$, then there are two pairs $(s_0, \omega_0) \in S_0$ and $(s_1, \omega_1) \in S_1$ such that for every memory μ and reference r: $\mu, r \vDash \omega \Leftrightarrow \mu, r \vDash (\omega_0 \wedge \omega_1) \ \wedge \ s = s_0 \Cup s_1$. Concretely, $T_{u[v]} \oplus_{\curlyvee} T_z = T_{u[v]}$. However, if we let $T'_z = \{(\mathsf{PRIM}^H, \mathtt{tt})\}$, $T_{u[v]} \oplus_{\curlyvee} T'_z = \{(\mathsf{PRIM}^H, \mathtt{tt})\}$.

In the rules that feature constraints, the hybrid TS tries to infer a dynamic assertion under which the corresponding expression is legal. For instance, when trying to type `x[y] = u[v] + z`, the hybrid TS tries to infer an assertion that is verified only if the level of the property that is being assigned is higher than or equal to the level of the left-hand side expression. Thus, we assume the existence of a function $When^?_{\Subset}$, parameterized with a generic order relation $\Subset$, that given two sets of elements paired up with runtime assertions, S_0 and S_1, generates an assertion $\omega = When^?_{\Subset}(S_0, S_1)$. The generated assertion describes the conditions under which there are two pairs $(s_0, \omega_0) \in S_0$ and $(s_1, \omega_1) \in S_1$ such that $s_0 \Subset s_1$ and $\omega_0 \wedge \omega_1$ holds. Formally, if $\omega = When^?_{\Subset}(S_0, S_1)$, then:

$$\forall_{\mu,r} \exists_{(s_0,\omega_0)\in S_0,(s_1,\omega_1)\in S_1} \ \mu, r \vDash \omega \Leftrightarrow \mu, r \vDash (\omega_0 \wedge \omega_1) \ \wedge \ s_0 \Subset s_1$$

For instance, in the current example: $When^?_{\preceq}(T_{x[y]}, T_{u[v]}) = (\hat{x}_i \in \{p_0\}) || (\hat{x}_j \in \{q_1\})$. If $\hat{x}_i \in \{p_0\}$ then the property being assigned is *high* and the assignment is legal. If $\hat{x}_j \in \{q_1\}$, then the value that is being assigned is *low* and, again, the assignment is legal.

The hybrid TS rewrites the program to be typed in order to dynamically check the assertions on which it is conditionally accepted. To this end, every conditionally typed expression is wrapped in a conditional expression that checks whether the assertion under which it was accepted holds. In order to simplify the specification, we make use of a syntactic function *wrap* that given an assertion ω and an expression e generates the expression $\omega \ ? \ (e) : (\_diverge())$, where $\_diverge()$ is a runtime function that always diverges. For instance, the program used as the running example is rewritten as follows: `_x_i = y, _x_j = v, (_x_i == "p_0" || _x_j == "q_1")? (x[y] = u[v]): (_diverge())`.

In Figure 6, we present the hybrid TS for the imperative fragment of Core JavaScript. Typing judgements have the form: $\Gamma \vdash e \rightsquigarrow e'/e'' : T, L$, where (1) Γ is the typing environment, (2) e the expression to be typed, (3) e' a new expression semantically equivalent to e except for the executions that are considered illegal, (4) e'' an expression that bookkeeps the value to which e' evaluates, (5) T a set of security types paired up with runtime assertions, and (6) L a set of security levels paired up with runtime assertions. In the specification of the hybrid TS, we make use of a new function *lev* that given a set T of security types paired up with runtime assertions produces the set $\{(\sigma, \omega) \mid (\tau^\sigma, \omega) \in T\}$.

$$\textsc{Val}\quad \frac{T = \{(\mathsf{PRIM}^{\perp}, \mathtt{tt})\} \quad L = \{(\top, \mathtt{tt})\}}{\Gamma \vdash v \rightsquigarrow v/v : T, L}$$

$$\textsc{This}\quad \frac{T = \{(\Gamma(\mathsf{this}), \mathtt{tt})\} \quad L = \{(\top, \mathtt{tt})\} \quad e = \hat{x}_i = \mathsf{this}}{\Gamma \vdash \mathsf{this}^k \rightsquigarrow e/\hat{x}_i : T, L}$$

$$\textsc{Var}\quad \frac{T = \{(\Gamma(x), \mathtt{tt})\} \quad L = \{(\top, \mathtt{tt})\}}{\Gamma \vdash x^i \rightsquigarrow \hat{x}_i = x/\hat{x}_i : T, L}$$

$$\textsc{Binary Operation}\quad \frac{\forall_{i=0,1} \cdot \Gamma \vdash e_i \rightsquigarrow e_i'/e_i'' : T_i, L_i \quad e' = e_0', e_1', \hat{x}_j = e_0'' \ \mathsf{op}\ e_1'', \hat{x}_j}{\Gamma \vdash e_0 \ \mathsf{op}^j \ e_1 \rightsquigarrow e'/\hat{x}_j : T_0 \oplus_{\gamma} T_1, L_0 \oplus_{\sqcap} L_1}$$

$$\textsc{Object Literal}\quad \frac{T = \{(\tau, \mathtt{tt})\} \ L = \{(\top, \mathtt{tt})\} \quad e' = \hat{x}_i = \{\}^{\tau}}{\Gamma \vdash \{\}^{\tau, i} \rightsquigarrow e'/\hat{x}_i : T, L}$$

$$\textsc{Variable Assignment}\quad \frac{\Gamma \vdash e \rightsquigarrow e'/e'' : T, L \qquad L' = \{(lev(\Gamma(x)), \mathtt{tt})\} \quad L'' = L \oplus_{\sqcap} L' \quad When^{?}_{\preceq}(T, \{(\Gamma(x), \mathtt{tt})\}) = \omega}{\Gamma \vdash x = e \rightsquigarrow e', wrap(\omega, x = e'')/e'' : T, L''}$$

$$\textsc{Property Look-up}\quad \frac{\forall_{i=0,1} \cdot \Gamma \vdash e_i \rightsquigarrow e_i'/e_i'' : T_i, L_i \quad L = L_0 \oplus_{\sqcap} L_1 \quad T = \pi_{\mathtt{type}}\left(\Gamma^{?}\ (T_0, P, e_1'')\right) \quad e = e_0', e_1', \hat{x}_j = e_0''[e_1'']}{\Gamma \vdash e_0[e_1, P]^j \rightsquigarrow e'/\hat{x}_j : T^{lev(T_0) \oplus_{\sqcup} lev(T_1)}, L}$$

$$\textsc{Property Assignment}\quad \frac{\forall_{i=0,1,2} \cdot \Gamma \vdash e_i \rightsquigarrow e_i'/e_i'' : T_i, L_i \qquad LT = \Gamma^{?}\ (T_0, P, i) \qquad L = \pi_{\mathtt{lev}}\,(LT) \qquad T = \pi_{\mathtt{type}}\,(LT) \quad When^{?}_{\preceq}(lev(T_0) \oplus lev(T_1), L) = \omega_0 \qquad When^{?}_{\preceq}(T_2, T) = \omega_1 \qquad \omega' = \vee\{\omega'' \wedge (\hat{x}_i \in dom(\dot{\tau})) \mid \exists \dot{\tau} \cdot (\dot{\tau}, \omega'') \in T_0\}}{\Gamma \vdash e_0[e_1, P]^i = e_2 \rightsquigarrow e_0', e_1', e_2', \hat{x}_i = e_1'', wrap(\omega_0 \wedge \omega_1 \wedge \omega', e_0''[e_1''] = e_2'')/e_2'' : T_2, L_0 \oplus_{\sqcap} L_1 \oplus_{\sqcap} L_2 \oplus_{\sqcap} L}$$

$$\textsc{In Expression}\quad \frac{\forall_{i=0,1} \cdot \Gamma \vdash e_i \rightsquigarrow e_i'/e_i'' : T_i, L_i \qquad L_P = \pi_{\mathtt{lev}}\left(\Gamma^{?}\ (T_0, P, e_0'')\right) \quad T = \{(\mathsf{PRIM}^{\perp}, \mathtt{tt})\}^{L_P \oplus_{\sqcup} lev(T_0) \oplus_{\sqcup} lev(T_1)}}{\Gamma \vdash e_0 \ \mathsf{in}^P_j \ e_1 \rightsquigarrow e_0', e_1', \hat{x}_i = e_1'', \hat{x}_j = e_0'' \ \mathsf{in}\ e_1''/\hat{x}_j : T, L_0 \oplus_{\sqcap} L_1}$$

$$\textsc{Property Deletion}\quad \frac{\Gamma \vdash e_0 \rightsquigarrow e_0'/e_0'' : T_0, L_0 \qquad L = \pi_{\mathtt{lev}}\left(\Gamma^{?}\ (T_0, \{p\}, e_0'')\right) \quad When^{?}_{\sqsubseteq}(lev(T_0), L) = \omega \qquad e' = e_0', wrap(\omega, \hat{x}_i = \mathsf{delete}\ e_0''.p)}{\Gamma \vdash \mathsf{delete}^i \ e_0.p \rightsquigarrow e'/\hat{x}_i : \{(\mathsf{PRIM}^{\perp}, \mathtt{tt})\}, L_0 \oplus_{\sqcap} L}$$

$$\textsc{Conditional Expression}\quad \frac{\forall_{i=0,1,2} \cdot \Gamma \vdash e_i \rightsquigarrow e_i'/e_i'' : T_i, L_i \qquad When^{?}_{\sqsubseteq}(lev(T_0), L_1 \oplus_{\sqcap} L_2) = \omega \quad e' = e_0', wrap(\omega, e_0''\ ?\ (e_1', \hat{x}_j = e_1'') : (e_2', \hat{x}_j = e_2'')) \qquad \omega_{\mathtt{tt}} = \neg(e_0'' \in V_F) \qquad \omega_{\mathtt{ff}} = (e_0'' \in V_F)}{\Gamma \vdash e_0\ ?^j\ (e_1) : (e_2) \rightsquigarrow e'/\hat{x}_j : T_1^{\omega_{\mathtt{tt}}} \cup T_2^{\omega_{\mathtt{ff}}}, L_0 \oplus_{\sqcap} (L_1^{\omega_{\mathtt{tt}}} \cup L_2^{\omega_{\mathtt{ff}}})}$$

$$\textsc{Sequence}\quad \frac{\forall_{i=0,1} \cdot \Gamma \vdash e_i \rightsquigarrow e_i'/e_i'' : T_i, L_i}{\Gamma \vdash e_0, e_1 \rightsquigarrow e_0', e_1'/e_1'' : T_1, L_0 \oplus_{\sqcap} L_1}$$

Fig. 6. Hybrid Typing Secure Information Flow in Core JavaScript

Furthermore, given a set L of security levels paired up with runtime assertions, we use T^L for the set $\{(\dot{\tau}', \omega) \mid (\dot{\tau}, \omega_t) \in T \wedge (\sigma, \omega_l) \in L \wedge \omega = \omega_t \wedge \omega_l \wedge \dot{\tau}' = \dot{\tau}^{\sigma}\}$. Finally, we use T^{ω} for the set $\{(\dot{\tau}, \omega \wedge \omega') \mid (\dot{\tau}, \omega') \in T\}$ and V_F for the set of *falsy* values: $\{false, 0, undefined, null\}$.

In order to illustrate the difference in functioning between the static and the hybrid TSs, let us consider the Rule [Property Assignment]. In the typing of a property assignment, all of the three subexpressions e_0, e_1, and e_2 are typed with three sets of possible types T_0, T_1, and T_2 and three sets of possible writing effects L_0, L_1, and L_2. The runtime assertion ω_0 guarantees that the existence level of the actual property being assigned is higher than or equal to the levels of the resources on which the computation of e_0 and e_1 depends (thereby avoiding implicit flows), while ω_1 guarantees that its security level is higher than or equal to the level of the value that is assigned to it (thereby avoiding explicit flows). Finally, the constraint ω' ensures that if the type of the receiver object does not include the $*$, then the property that is being assigned is in its domain. The instrumentation wraps the property assignment in a conditional expression that checks whether all of the three conditions hold.

The soundness of the hybrid TS is established by Theorems 2 and 3. The former states that the semantics of the instrumented program is contained in the semantics of the original one, while the latter states that the instrumented program is noninterferent. In the following, we use $\mu \simeq \mu'$ whenever μ and μ' coincide in all variables/properties available for the programmer and μ does not define mappings for variables/properties in $\mathcal{V}_{TS}$.[6]

[6] We give the formal definition in Appendix B.

Theorem 2 (Transparency). *For any expression e, typing environment Γ, memory μ well-labeled by Σ, and reference r such that $\Gamma \vdash e \rightsquigarrow e'/e'' : T, L$, and $r \vdash \langle \mu, \Sigma, e'' \rangle \Downarrow \langle \mu'_f, \Sigma_f, v \rangle$, it holds that $r \vdash \langle \mu, \Sigma, e \rangle \Downarrow \langle \mu_f, \Sigma_f, v \rangle$, where $\mu_f \simeq \mu'_f$.*

Theorem 3 (Noninterference). *For any expression e and typing environment Γ, if $\Gamma \vdash e \rightsquigarrow e', e'' : T, L$, then e'' is noninterferent w.r.t. Γ.*

Theorem 4 characterizes the precision of the hybrid TS. It shows that, given two expressions $\hat{e}$ and e that only differ in look-up annotations, whenever $\hat{e}$ is typable using the purely static TS and it converges, then e is typable using the hybrid TS and its instrumentation also converges. Hence, the theorem shows that the changing of look-up annotations of an expression $\hat{e}$, typable using the purely static TS, always yields an expression e, typable using the hybrid TS, whose evaluation never diverges due to the failure of runtime assertions.

Theorem 4 (Precision). *For any two expressions $\hat{e}$ and e, typing environment Γ, and memory μ well-labeled by Σ such that: $\hat{e} \equiv e$, $\Gamma \vdash \hat{e} : \tau, \sigma$ and $\#glob \vdash \langle \mu, \Sigma, e \rangle \Downarrow \langle \mu_f, \Sigma_f, v \rangle$; it holds that: $\Gamma \vdash e \rightsquigarrow e'/e'' : T, L$ and $\#glob \vdash \langle \mu, \Sigma, e \rangle \Downarrow \langle \mu'_f, \Sigma_f, v \rangle$.*

5 Related Work and Conclusions

Since the seminal work of Volpano *et al* [27] on typing secure information flow in a simple imperative language, TSs for IFC have been proposed for a wide variety of languages, ranging from functional [20] to Java-like object-oriented languages [7]. To the best of our knowledge, our TS is the first one that addresses the particular features of JavaScript in the context of IFC.

The increasing popularity of dynamic languages has motivated further research on runtime mechanisms for IFC, such as *information flow monitors*. In contrast to *purely dynamic monitors* [4–6] that do not rely on any kind of static analysis, *hybrid monitors* [13, 23, 26], use static analysis to reason about the implicit flows that arise due to untaken execution paths. Furthermore, some hybrid monitors also use static analyses to boost performance. For instance, Moore *et al* [18] show how to combine monitoring and static analysis so as to reduce the number of variables whose levels are tracked at runtime. Interestingly, Russo *et al* [21] prove that hybrid monitors are more permissive than both purely dynamic and purely static enforcement mechanisms. Their result supports the need for mechanisms which combine static and dynamic analysis like our own. However, unlike hybrid monitors, the hybrid TS we propose does **not** require any kind of runtime tracking of security levels, since the inlined conditions feature the actual values that are computed by the program rather than their levels.

Recently, *gradual security typing* [9, 11] has been proposed as a way to combine runtime monitoring and static analysis in order to cater for controlled forms of *polymorphism*. Concretely, the programmer is expected to introduce runtime casts in points where values of a pre-determined security type are expected.

"The type system statically guarantees adherence to the [security] policies on the static side of a cast, whereas the runtime system checks the policies on the dynamic side" [11]. This approach could be used for the typing of arbitrary property look-ups. However, this would necessarily imply partial tracking of security levels, which our solution does not require.

Due to the complexity of JavaScript semantics, most mechanisms for preventing security violations spawned by client-side JavaScript code have focused on isolation properties [8, 10, 16, 19], which are easier to enforce than noninterference [12]. The analyses presented in [16] and [19] deal in different ways with the issue of property look-ups featuring arbitrary expressions. While the authors of [16] consider a subset of the language that does not include this kind of look-up expression, the TS presented in [19] overapproximates the set of properties to which these arbitrary expressions may evaluate. We believe that the idea illustrated by the hybrid TS could be applied both to [16] and [19] in order to increase their permissiveness.

Hedin *et al* [14] have been the first to propose an information flow monitor for a realistic core of JavaScript. They introduce the notion of *existence levels* to deal with the constructs for the checking of the existence of properties. They further introduce the notion of *structure security level* (SSL), which corresponds to an upper bound on the existence levels of the properties of an object. Hence, if an object o has a *low* SSL, one can only change its structure[7] in *low* contexts. It is important to emphasize that the SSL is not a key element for the characterization of the attacker model inherent to JavaScript, but rather a device of the authors' enforcement mechanism. The need for the SSL arises from the fact that existence levels are not established *a priori*. Hence, the SSL plays the role of the existence level of the properties that do not exist yet.

Thiemann [25] has proposed a TS that guarantees *termination* and *progress* for a fragment of JavaScript, which does not account for objects whose domain may change at runtime. To overcome this issue, Anderson *et al* [3] have proposed a type inference algorithm that allows objects "to evolve in a controlled manner" by classifying their properties as *definite* or *potential*. This additional information could be used by the static TS to distinguish *property creations* from *property updates*, thereby relaxing the constraints imposed on property updates, which would not need to take into account the existence level of the updated property.

In summary, we have presented a new TS for enforcing secure information flow in a core of JavaScript that takes into account the defining features of the language: prototypical inheritance, constructs for checking the existence of properties, extensible objects, and unusual interactions between the binding of variables and the binding of properties. In order to boost its permissiveness, we have designed a hybrid version of the proposed TS that explores a novel way of combining static and dynamic analysis in the context of information flow control. A prototype of the TS as well as a long version of the paper that includes the proofs of the theorems are available online [1].

[7] Either by adding properties to o or removing properties from o.

References

1. Code + Proofs. http://www-sop.inria.fr/members/Jose.Santos/.
2. The 3rd edition of ECMA 262. ECMAScript Language Specification. Technical report, ECMA, 1999.
3. C. Anderson, P. Giannini, and S. Drossopoulou. Towards type inference for javascript. In *ECOOP*, 2005.
4. T. H. Austin and C. Flanagan. Efficient purely-dynamic information flow analysis. In *PLAS*, 2009.
5. T. H. Austin and C. Flanagan. Permissive dynamic information flow analysis. In *PLAS*, 2010.
6. T. H. Austin and C. Flanagan. Multiple facets for dynamic information flow. In *POPL*, 2012.
7. A. Banerjee and D. A. Naumann. Secure information flow and pointer confinement in a java-like language. In *CSFW*, 2002.
8. D. Crockford. Adsafe. http://www.adsafe.org.
9. T. Disney and C. Flanagan. Gradual information flow typing. In *STOP*, 2011.
10. The FaceBook Team: FBJS. http://wiki.developers.facebook.com/index.php/FBJS.
11. L. Fennell and P. Thiemann. Gradual security typing with references. In *CSF*, 2013.
12. J. A. Goguen and J. Meseguer. Security policies and security models. In *S&P*, 1982.
13. G. Le Guernic. *Confidentiality Enforcement Using Dynamic Information Flow Analyses*. PhD thesis, Kansas State University, 2007.
14. D. Hedin and A. Sabelfeld. Information-flow security for a core of javascript. In *CSF*, 2012.
15. S. Maffeis, J. C. Mitchell, and A. Taly. An operational semantics for javascript. In *APLAS*, 2008.
16. S. Maffeis and A. Taly. Language-based isolation of untrusted javascript. In *CSF*, 2009.
17. A. Almeida Matos and G. Boudol. On declassification and the non-disclosure policy. In *CSFW*, 2005.
18. Scott Moore and Stephen Chong. Static analysis for efficient hybrid information-flow control. In *CSF*, 2011.
19. J. Gibbs Politz, S. A. Eliopoulos, A. Guha, and S. Krishnamurthi. Adsafety: Type-based verification of javascript sandboxing. In *USENIX Security Symposium*, 2011.
20. F. Pottier, , and V. Simonet. Information flow inference for ML. *ACM Trans. Program. Lang. Syst.*, 2003.
21. A. Russo and A. Sabelfeld. Dynamic vs. static flow-sensitive security analysis. In *CSF*, 2010.
22. A. Sabelfeld and A. C. Myers. Language-based information-flow security. *IEEE Journal on Selected Areas in Communications*, 2003.
23. P. Shroff, S. F. Smith, and M. Thober. Dynamic dependency monitoring to secure information flow. In *CSF*, 2007.
24. A. Taly, U. Erlingsson, J. C. Mitchell, M. S. Miller, and J. Nagra. Automated analysis of security-critical javascript apis. In *SP*, 2011.
25. P. Thiemann. Towards a type system for analyzing javascript programs. In *ESOP*, 2005.
26. V. N. Venkatakrishnan, W. Xu, D. C. DuVarney, and R. Sekar. Provably correct runtime enforcement of non-interference properties. In *ICICS*, 2006.
27. D. M. Volpano, C. E. Irvine, and G. Smith. A sound type system for secure flow analysis. *Journal of Computer Security*, 1996.

A Formal Semantics of Core JavaScript

The semantics of Core JavaScript is given in Figure 7. It makes use of three semantic relations $\mathcal{R}_{Scope}$, $\mathcal{R}_{Proto}$, and $\mathcal{R}_{NewScope}$, respectively given in Definitions 3, 4, and 6. It further uses a function $fresh$, given in Definition 5, which corresponds to the *parametric object allocator*.

Notation. We use the notation: (1) $[p_0 \mapsto v_0, \cdots, p_n \mapsto v_n]$ for the partial function that maps p_0 to v_0, ..., and p_n to v_n resp., (2) $f\,[p_0 \mapsto v_0, \cdots, p_n \mapsto v_n]$ for the function that coincides with f everywhere except in p_0, ..., p_n, which are otherwise mapped to v_0, ..., v_n resp., (3) $f|_P$ for the restriction of f to P (provided it is included in its domain), and (4) $f(r)(p)$ for $(f(r))(p)$, that is, the application of the image of r by f (which is assumed to be a function) to p.

Definition 3 (Scope-Chain Inspection – $\mathcal{R}_{Scope}$). *The relation $\mathcal{R}_{Scope}$ is recursively defined as follows:*

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

Definition 4 (Prototype-Chain Inspection – $\mathcal{R}_{Proto}$). *The relation $\mathcal{R}_{Proto}$ is recursively defined as follows:*

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

Definition 5 (Parametric Allocator). *A parametric allocator $fresh$ is a function that maps triples of the form (μ, Σ, σ) where μ is a memory, Σ a labeling, and σ a security level to a free reference in μ and which verifies the following property: $\mu_0 \restriction^{\Sigma_0,\sigma} = \mu_1 \restriction^{\Sigma_1,\sigma} \Rightarrow fresh(\mu_0, \Sigma_0, \sigma) = fresh(\mu_1, \Sigma_1, \sigma)$.*

Definition 6 (Scope Creation – $\mathcal{R}_{NewScope}$). *For any two memories μ and μ', labeling Σ, three references r_f, r_{this}, and r, value v, and expression e: $\langle \mu, \Sigma, r_f, v, r_{this}\rangle \mathcal{R}_{NewScope} \langle \mu', e, r\rangle$ holds if and only if $\lambda^{(\dot\tau, i)} x.\,\{\mathsf{var}^{\dot\tau_1,\cdots,\dot\tau_n}\ y_1, \cdots, y_n;\ e\}$ $= \mu(r_f)(@code)$, $r' = \mu(r_f)(@fscope)$, $r = fresh(\mu, \Sigma, \top)$, and:*

$\mu' = \mu[r \mapsto [@scope \mapsto r', x \mapsto v, @this \mapsto r_{this}, y_1 \mapsto undefined, \cdots, y_n \mapsto undefined]]$

for some variables $x, y_1, \cdots, y_n$.

B Auxiliary Definitions Used in the Proofs

In this section we present several definitions that are either referred to in the paper or required for the proofs.

Admissible Prototypes. Not all types can be used as the *type of the prototype* for the objects of a given type. Consider, for instance, an object o_0 of type $\dot\tau_0 = \mu\kappa.\langle p^L : \textsf{PRIM}^L, \_proto\_^L : \_\rangle$ and an object o_1 of type $\dot\tau_1 = \mu\kappa.\langle p^L : \mu\kappa.\langle *^L :$

VALUE
$$r \vdash \langle \mu, \Sigma, v \rangle \Downarrow \langle \mu, \Sigma, v \rangle$$

THIS
$$r \vdash \langle \mu, \Sigma, \mathsf{this} \rangle \Downarrow \langle \mu, \Sigma, \mu(r)(@this) \rangle$$

VARIABLE
$$\frac{\langle \mu, r, x \rangle \ \mathcal{R}_{Scope} \ r_x \qquad r_x \neq null \qquad v = \mu(r_x)(x)}{r \vdash \langle \mu, \Sigma, x \rangle \Downarrow \langle \mu, \Sigma, v \rangle}$$

BINARY OPERATION
$$\frac{\begin{array}{c} r \vdash \langle \mu, \Sigma, e_0 \rangle \Downarrow \langle \mu_0, \Sigma_0, v_0 \rangle \\ r \vdash \langle \mu_0, \Sigma_0, e_1 \rangle \Downarrow \langle \mu_1, \Sigma_1, v_1 \rangle \end{array}}{r \vdash \langle \mu, \Sigma, e_0 \ \mathsf{op} \ e_1 \rangle \Downarrow \langle \mu_1, \Sigma_1, v_0 \ \mathsf{op} \ v_1 \rangle}$$

IN EXPRESSION
$$\frac{\begin{array}{c} r \vdash \langle \mu, \Sigma, e_0 \rangle \Downarrow \langle \mu_0, \Sigma_0, m_0 \rangle \qquad r \vdash \langle \mu_0, \Sigma_0, e_1 \rangle \Downarrow \langle \mu_1, \Sigma_1, r_1 \rangle \\ \langle \mu_1, r_1, m_0 \rangle \ \mathcal{R}_{Proto} \ r' \qquad r' \neq null \Rightarrow v = \mathtt{ff} \qquad r' = null \Rightarrow v = \mathtt{tt} \end{array}}{r \vdash \langle \mu, \Sigma, e_0 \ \mathsf{in} \ e_1 \rangle \Downarrow \langle \mu_1, \Sigma_1, v \rangle}$$

VARIABLE ASSIGNMENT
$$\frac{\begin{array}{c} r \vdash \langle \mu, \Sigma, e \rangle \Downarrow \langle \mu_0, \Sigma_0, v_0 \rangle \qquad \langle \mu_0, r, x \rangle \ \mathcal{R}_{Scope} \ r_x \\ r_x \neq null \qquad \mu' = \mu_0 \left[r_x \mapsto \mu_0(r_x) \left[x \mapsto v_0 \right] \right] \end{array}}{r \vdash \langle \mu, \Sigma, x = e \rangle \Downarrow \langle \mu', \Sigma_0, v_0 \rangle}$$

PROPERTY LOOK-UP
$$\frac{\begin{array}{c} r \vdash \langle \mu, \Sigma, e_0 \rangle \Downarrow \langle \mu_0, \Sigma_0, r_0 \rangle \qquad r \vdash \langle \mu_0, \Sigma_0, e_0 \rangle \Downarrow \langle \mu_1, \Sigma_1, m_1 \rangle \\ \langle \mu_1, r_0, m_1 \rangle \ \mathcal{R}_{Proto} \ r' \\ r' \neq null \Rightarrow v = \mu_1(r')(m_1) \qquad r' = null \Rightarrow v = undefined \end{array}}{r \vdash \langle \mu, \Sigma, e_0[e_1] \rangle \Downarrow \langle \mu_1, \Sigma_1, v \rangle}$$

PROPERTY ASSIGNMENT
$$\frac{\begin{array}{c} r \vdash \langle \mu, \Sigma, e_0 \rangle \Downarrow \langle \mu_0, \Sigma_0, r_0 \rangle \qquad r \vdash \langle \mu_0, \Sigma_0, e_1 \rangle \Downarrow \langle \mu_1, \Sigma_1, m_1 \rangle \\ r \vdash \langle \mu_1, \Sigma_1, e_2 \rangle \Downarrow \langle \mu_2, \Sigma_2, v_2 \rangle \qquad \mu' = \mu_2 \left[r_0 \mapsto \mu_2(r_0) \left[m_1 \mapsto v_2 \right] \right] \end{array}}{r \vdash \langle \mu, \Sigma, e_0[e_1] = e_2 \rangle \Downarrow \langle \mu', \Sigma_2, v_2 \rangle}$$

PROPERTY DELETION
$$\frac{\begin{array}{c} r \vdash \langle \mu, \Sigma, e_0 \rangle \Downarrow \langle \mu_0, \Sigma_0, r_0 \rangle \\ \mu' = \mu_0 \left[r_0 \mapsto \mu_0(r_0)|_{dom(\mu_0(r_0) - p)} \right] \end{array}}{r \vdash \langle \mu, \Sigma, \mathsf{delete} \ e_0.p \rangle \Downarrow \langle \mu', \Sigma_0, \mathtt{tt} \rangle}$$

FUNCTION LITERAL
$$\frac{\begin{array}{c} r' = fresh(\mu, \Sigma, lev(\dot{\tau})) \qquad \Sigma' = \Sigma \left[r' \mapsto \dot{\tau} \right] \\ \mu' = \mu \left[r' \mapsto \left[@fscope \mapsto r, @code \mapsto \lambda^{(\Gamma, \dot{\tau})} x. \left\{ \mathsf{var}^{\dot{\tau}_1, \cdots, \dot{\tau}_n} \ y_1, \cdots, y_n; \ e \right\} \right] \right] \end{array}}{r \vdash \langle \mu, \Sigma, \mathsf{function}^{\Gamma, \dot{\tau}, i}(x) \{ \mathsf{var}^{\dot{\tau}_1, \cdots, \dot{\tau}_n} \ y_1, \cdots, y_n; \ e \} \rangle \Downarrow \langle \mu', \Sigma', r' \rangle}$$

OBJECT LITERAL
$$\frac{\begin{array}{c} r' = fresh(\mu, \Sigma, lev(\Sigma(\dot{\tau}))) \qquad \Sigma' = \Sigma \left[r' \mapsto \dot{\tau} \right] \\ \mu' = \mu \left[r' \mapsto \left[\_proto\_ \mapsto null \right] \right] \end{array}}{r \vdash \langle \mu, \Sigma, \{\}^{\dot{\tau}, i} \rangle \Downarrow \langle \mu', \Sigma', r' \rangle}$$

FUNCTION CALL
$$\frac{\begin{array}{c} r \vdash \langle \mu, \Sigma, e_0 \rangle \Downarrow \langle \mu_0, \Sigma_0, r_0 \rangle \qquad r \vdash \langle \mu_0, \Sigma_0, e_1 \rangle \Downarrow \langle \mu_1, \Sigma_1, v_1 \rangle \\ \langle \mu_1, r_0, v_1, \#glob \rangle \ \mathcal{R}_{NewScope} \ \langle \hat{\mu}, \hat{e}, \hat{r} \rangle \\ \hat{r} \vdash \langle \hat{\mu}, \Sigma_1, \hat{e} \rangle \Downarrow \langle \mu', \Sigma', v \rangle \end{array}}{r \vdash \langle \mu, \Sigma, e_0(e_1) \rangle \Downarrow \langle \mu', \Sigma', v \rangle}$$

CONDITIONAL EXPRESSION
$$\frac{\begin{array}{c} r \vdash \langle \mu, \Sigma, \hat{e} \rangle \Downarrow \langle \hat{\mu}, \hat{\Sigma}, \hat{v} \rangle \\ \hat{v} \notin V_F \Rightarrow i = 0 \qquad \hat{v} \in V_F \Rightarrow i = 1 \\ r \vdash \langle \hat{\mu}, \hat{\Sigma}, e_i \rangle \Downarrow \langle \mu', \Sigma', v \rangle \end{array}}{r \vdash \langle \mu, \Sigma, \hat{e} \ ? \ (e_0) : (e_1) \rangle \Downarrow \langle \mu', \Sigma', v \rangle}$$

METHOD CALL
$$\frac{\begin{array}{c} r \vdash \langle \mu, \Sigma, e_0 \rangle \Downarrow \langle \mu_0, \Sigma_0, r_0 \rangle \qquad r \vdash \langle \mu_0, \Sigma_0, e_1 \rangle \Downarrow \langle \mu_1, \Sigma_1, m_1 \rangle \\ r \vdash \langle \mu_1, \Sigma_1, e_2 \rangle \Downarrow \langle \mu_2, \Sigma_2, v_2 \rangle \qquad \langle \mu_2, r_0, m_1 \rangle \ \mathcal{R}_{Proto} \ r_m \qquad r_f = \mu_2(r_m)(m_1) \\ \langle \mu_2, r_f, v_2, r_0 \rangle \ \mathcal{R}_{NewScope} \ \langle \hat{\mu}, \hat{e}, \hat{r} \rangle \qquad \hat{r} \vdash \langle \hat{\mu}, \Sigma_2, \hat{e} \rangle \Downarrow \langle \mu', \Sigma', v \rangle \end{array}}{r \vdash \langle \mu, \Sigma, e_0[e_1](e_2) \rangle \Downarrow \langle \mu', \Sigma', v \rangle}$$

SEQUENCE
$$\frac{\begin{array}{c} r \vdash \langle \mu, \Sigma, e_0 \rangle \Downarrow \langle \mu_0, \Sigma_0, v_0 \rangle \\ r \vdash \langle \mu_0, \Sigma_0, e_1 \rangle \Downarrow \langle \mu_1, \Sigma_1, v_1 \rangle \end{array}}{r \vdash \langle \mu, \Sigma, e_0, e_1 \rangle \Downarrow \langle \mu_1, \Sigma_1, v_1 \rangle}$$

Fig. 7. A Big-Step Semantics for Core JavaScript

$\mathsf{PRIM}^L \rangle^L \rangle$. Suppose we set $\dot{\tau}_1$ as the type of the prototype in $\dot{\tau}_0$. Then, the look-up of p in o_0 may yield two different types of values (besides `undefined`, if neither o_0 nor o_1 defines p). It yields a value of type PRIM^L when object o_0 defines p and an object of type $\mu\kappa.\langle *^L : \mathsf{PRIM}^L \rangle^L$ when o_0 does not define p and o_1 defines p. In order to overcome this problem, we restrict what types can be legally used for the prototype of a given object type. Informally, $\dot{\tau}_1$ is a *consistent prototype type* for $\dot{\tau}_0$ if $\dot{\tau}_1$ does not define a default type and it coincides with $\dot{\tau}_0$ for all properties in its domain.

Scope-Chains. In order to simplify the proofs, we assume that parsed function literals in memory are annotated with typing environment in which the corresponding function literals were typed. It is important to emphasize that this is

just a device for the proofs and not a feature of the enforcement mechanism. Concretely, the implementation does not need to associate with every parsed function literal the corresponding typing environment. We define a scope-chain as triple (μ, r, Γ) consisting of a memory, a reference to a scope object, and the typing environment associated with the variables defined in that scope. The function $\mathcal{S}Chain$ receives a memory and reference to a function object and outputs the corresponding scope-chain. Concretely, $\mathcal{S}Chain(\mu, r) = (\mu, r', \Gamma)$ *iff* $r' = \mu(r)(@fscope)$ and $\Gamma = tenv(\mu(r)(@code))$, where $tenv(f)$ corresponds to the typing environment that annotates f. Definition 7 establishes the notion of *low-projection* for scope chains.

Definition 7 (Low-Projection for Scope-Chains). *The low-projection of the scope-chain (μ, r, Γ) at security level σ is defined as follows:*

$$(\mu, r, \Gamma) \restriction^{\sigma} = \{(x, \mu(r_x)(x)) \mid x \in dom(\Gamma) \wedge lev(\Gamma(x)) \sqsubseteq \sigma \wedge \langle \mu, r, x \rangle \; \mathcal{R}_{Scope} \; r_x \wedge r_x \neq null\}$$

Well-labeled Memories. The definition of noninterference requires that the original memories to be *well-labeled*. This means that types declared in the typing environment must "match" their corresponding values. Definition 8 establishes the notion of *well-labeled scope-chain*, whereas Definition 9 gives the notion of *well-labeled memory*. In the following we make use of the notion of *extended labeling*. Given a labeling $\Sigma : \mathcal{R}ef \rightarrow \mathcal{T}$, we define its extension $\Sigma^* : \mathcal{R}ef \cup \mathcal{P}rim \rightarrow \mathcal{T}$ as follows: $\Sigma^*(v) = \Sigma(v)$ if $v \in \mathcal{R}ef$ and $\Sigma^*(v) = \mathsf{PRIM}^{\top}$ if $v \in \mathcal{P}rim$.

Definition 8 (Well-labeled Scope-Chain). *The scope-chain (μ, r', Γ) is well-labeled by well-labeled by Σ if for every variable $x \in dom(\Gamma)$ for which there is a reference r' such that $\langle \mu, r, x \rangle \; \mathcal{R}_{Scope} \; r'$ and $r' \neq null$, it follows that: $\Gamma(x) \preceq \Sigma^*(\mu(r_x)(x))$.*

Definition 9 (Well-labeled Memory). *A memory μ is well-labeled by Σ, if:* **(1)** *every reference pointing to a non-scope object in μ is in the domain of Σ,* **(2)** *every funtion object in μ is mapped to a function type $\dot{\tau}$ by Σ, which correctly types the corresponding function in its annotated environment, and the corresponding scope-chain is well-labeled Σ, and* **(3)** *all references to non-internal objects in $dom(\mu)$ are in $dom(\Sigma)$, and for every reference $r \in dom(\Sigma)$ and property $p \in dom(\mu(r))$ $\Sigma(\mu(r)(p)) \preceq \Sigma(r), p)$.*

Instrumentation. Definition 10 formalizes the similarity relation between the memory of an instrumented program and the memory of its original version. We use the notation $@dom$ for the domain of an object including internal properties.

Definition 10 (Memory Similarity). *Two memories μ and μ' are said to be similar, written $\mu \simeq \mu'$, if:* **(1)** *the domains coincide – $dom(\mu) = dom(\mu')$,* **(2)** *for every reference $r \in dom(\mu)$ and property $p \in @dom(\mu(r))$, $\mu(r)(p) = \mu'(r)(p)$ and $p \notin \mathcal{V}_{TS}$, and* **(3)** *for every reference $r \in dom(\mu')$ and property $p \in dom(\mu'(r)) \backslash dom(\mu(r))$, $p \in \mathcal{V}_{TS}$.*

C Proof for the Static Type System

Lemma 1 (Well-Labeled Prototype Chains). *Given a memory μ well-labeled (hyp.1) by Σ, a reference r, and property p, such that $\langle \mu, r, p\rangle \ \mathcal{R}_{Proto}\ r'$ (hyp.2), then $\Uparrow(\Sigma(r), p) = \Uparrow(\Sigma(r'), p)$, whenever $\Uparrow(\Sigma(r'), p)$ is defined.*

Proof. Suppose that $\Uparrow(\Sigma(r'), p) = (\sigma, \dot{\tau})$ (hyp.3). We need to show that: $\Uparrow(\Sigma(r), p) = (\sigma, \dot{\tau})$. We proceed by induction on the derivation of $\langle \mu, r, p\rangle \ \mathcal{R}_{Proto}\ r'$.

[Base] $p \in dom(\mu(r))$ (hyp.4). We conclude that:

- $r = r'$ — (1) - hyp.2 + hyp.4
- $\Uparrow(\Sigma(r), p) = (\sigma, \dot{\tau})$ — (2) - hyp.3 + (1)

[Look-up] $p \notin dom(\mu(r))$ (hyp.3). We conclude that:

- $\langle \mu, r'', p\rangle \ \mathcal{R}_{Proto}\ r'$ and $\mu(r)(\_proto\_) = r''$. — (1) - hyp.2 + hyp.4
- $\Uparrow(\Sigma(r''), p) = \Uparrow(\Sigma(r'), p)$ — (2) - hyp.1 + (1) + **ih**
- $\Sigma(r'') \preceq\ \pi_{\texttt{type}}(\Uparrow(\Sigma(r), \_proto\_))$ — (3) - hyp.1 + (1)
- $\lfloor \pi_{\texttt{type}}(\Uparrow(\Sigma(r), \_proto\_))\rfloor \equiv \lfloor \Sigma(r'')\rfloor$ — (4) - (3)
- $\Uparrow(\pi_{\texttt{type}}(\Uparrow(\Sigma(r), \_proto\_)), p) = \Uparrow(\Sigma(r''), p)$ — (5) - (4)
- $\Uparrow(\Sigma(r), p) = \Uparrow(\pi_{\texttt{type}}(\Uparrow(\Sigma(r), \_proto\_)), p)$ — (6) - *Consistent Prototype*
- $\Uparrow(\Sigma(r), p) = (\sigma, \dot{\tau})$ — (7) - hyp.3 + (2) + (5) + (6)

□

Lemma 2 (Prototype-Chain Indistinguishability). *For any two memories μ_0 and μ_1 respectively well-labeled by Σ_0 and Σ_1, reference r, and property p such that $\langle \mu_0, r, p\rangle \ \mathcal{R}_{Proto}\ r_0$ (hyp.1), $\langle \mu_1, r, p\rangle \ \mathcal{R}_{Proto}\ r_1$ (hyp.2), $\mu_0, \Sigma_0 \sim_\sigma \mu_1, \Sigma_1$ (hyp.3), and $\pi_{\texttt{lev}}(\Uparrow(\Sigma_0(r), p)) \sqcup lev(\Sigma_0(r)) \sqsubseteq \sigma$ (hyp.4), it holds that: $r_0 = r_1$.*

Proof. We proceed by induction on the derivation of hyp.1.

[Null] $r = null$ (hyp.5). We conclude that:

- $r_0 = r_1 = null$ — (1) - hyp.1 + hyp.2 + hyp.5

[Base] $p \in dom(\mu_0(r))$ (hyp.5). We conclude that:

- $r_0 = r$ — (1) - hyp.1 + hyp.5
- $\Sigma_0(r) = \Sigma_1(r)$ — (2) - hyp.3 + hyp.4
- $\pi_{\texttt{1ev}}(\Uparrow(\Sigma_0(r), p)) = \pi_{\texttt{1ev}}(\Uparrow(\Sigma_1(r), p)) \sqsubseteq \sigma$ — (3) - hyp.4 + (2)
- $p \in dom(\mu_1(r))$ — (4) - hyp.3 + hyp.5 + (3)
- $r_1 = r$ — (5) - hyp.2 + (4)
- $r_0 = r_1$ — (6) - (1) + (5)

[Look-up] $p \notin dom(\mu_0(r))$ (hyp.5) and $\langle \mu_0, r'_0, p\rangle \ \mathcal{R}_{Proto}\ r_0$ (hyp.6) where $r'_0 = \mu_0(r)(\_proto\_)$ (hyp.7). We conclude that:

- $\Sigma_0(r) = \Sigma_1(r)$ and $lev(\Sigma_0(r)) = lev(\Sigma_1(r)) \sqsubseteq \sigma$ — (1) - hyp.3 + hyp.4
- $\pi_{\texttt{1ev}}(\Uparrow(\Sigma_0(r), p)) = \pi_{\texttt{1ev}}(\Uparrow(\Sigma_1(r), p)) \sqsubseteq \sigma$ — (2) - hyp.4 + (1)
- $p \notin dom(\mu_1(r))$ — (3) - hyp.3 + hyp.5 + (1) + (2)
- $\langle \mu_1, r'_1, p\rangle \ \mathcal{R}_{Proto}\ r_1$, where $r'_1 = \mu_1(r)(\_proto\_)$ — (4) - hyp.2 + (3)

- $\pi_{\texttt{type}}\left(\Pi^{\star}\left(\Sigma_i(r), \_proto\_\right)\right) \preceq \Sigma_i(r)$ for $i = 0, 1$
  (5) - hyp.7 + (4) + *Syntax of Object Types* + *Well-labeled Memory*
- $lev(\pi_{\texttt{type}}\left(\Pi^{\star}\left(\Sigma_i(r), \_proto\_\right)\right)) \sqsubseteq lev(\Sigma_i(r)) \sqsubseteq \sigma$, for $i = 0, 1$ (6) - (1) + (5)
- $r'_0 = r'_1$ (7) - hyp.3 + hyp.7 + (1) + (6)
- $\Sigma_i(r'_i) \preceq \pi_{\texttt{type}}\left(\Pi^{\star}\left(\Sigma_i(r), \_proto\_\right)\right)$, for $i = 0, 1$
  (8) - hyp.7 + (5) + *Well-labeled Memory*
- $\Sigma_i(r'_i) \preceq \Sigma_i(r)$, for $i = 0, 1$ (9) - (5) + (9)
- $lev(\Sigma_i(r'_i)) \sqsubseteq lev(\Sigma_i(r)) \sqsubseteq \sigma$ and $\lfloor \Sigma_i(r'_i) \rfloor = \lfloor \Sigma_i(r) \rfloor$, for $i = 0, 1$ (10) - (1) + (9)
- $\pi_{\texttt{lev}}\left(\Pi^{\star}\left(\Sigma_0(r'_0), p\right)\right) \sqcup lev(\Sigma_0(r'_0)) \sqsubseteq \sigma$ (11) - hyp.4 + (10)
- $r_0 = r_1$ (12) - hyp.3 + hyp.4 +hyp.6 + (4) + (7) + (11) + **ih**

□

Lemma 3 (Indistinguishable Variable Assignment). *For any two memories μ_0 and μ_1, typing environment Γ, reference r, security level σ, variable x, and values v_0 and v_1 such that:*

- $\Gamma, r \Vdash \mu_0 \sim_\sigma \mu_1$ *(hyp.1),*
- $\langle \mu_0, r, x \rangle\ \mathcal{R}_{Scope}\ r_0$ *(hyp.2) for some reference r_0, $\mu'_0 = \mu_0\left[r_0 \mapsto \mu_0(r_0)\left[x \mapsto v_0\right]\right]$ (hyp.3),*
- $\langle \mu_1, r, x \rangle\ \mathcal{R}_{Scope}\ r_1$ *(hyp.4) for some reference r_1, $\mu'_1 = \mu_1\left[r_1 \mapsto \mu_1(r_1)\left[x \mapsto v_1\right]\right]$ (hyp.5),*
- $lev(\Gamma(x)) \sqsubseteq \sigma \Rightarrow v_0 = v_1$ *(hyp.6)*

It holds that: $\Gamma, r \Vdash \mu'_0 \sim_\sigma \mu'_1$.

Lemma 4 (Indistinguishable Property Assignment). *For any two memories μ_0 and μ_1 respectively labeled by Σ_0 and Σ_1, reference r, string p, security level σ, and values v_0 and v_1 such that:*

- $\mu_0, \Sigma_0 \sim_\sigma \mu_1, \Sigma_1$ *(hyp.1),*
- $\mu'_0 = \mu_0\left[r \mapsto \mu_0(r)\left[p \mapsto v_0\right]\right]$ *(hyp.2),*
- $\mu'_1 = \mu_1\left[r \mapsto \mu_1(r)\left[p \mapsto v_1\right]\right]$ *(hyp.3),*
- $lev(\Sigma_0(r)) \sqcup lev(\pi_{type}\left(\Pi^{\star}\left(\Sigma_0(r), p\right)\right)) \sqsubseteq \sigma \Rightarrow v_0 = v_1$ *(hyp.4)*

It holds that: $\mu'_0, \Sigma_0 \sim_\sigma \mu'_1, \Sigma_1$.

Lemma 5 (Indistinguishable Scope-Object Allocation). *For any two memories μ_0 and μ_1 respectively labeled by Σ_0 and Σ_1, references r_f, r_0, and r_1 (where r_f points to a function object), values v_0 and v_1, and security level σ such that:*

- $\mu_0, \Sigma_0 \sim_\sigma \mu_1, \Sigma_1$ *(hyp.1),*
- $\langle \mu_0, r_f, v_0, r_0 \rangle\ \mathcal{R}_{NewScope}\ \langle \mu'_0, \hat{e}_0, \hat{r}_0 \rangle$ *(hyp.2),*
- $\langle \mu_1, r_f, v_1, r_1 \rangle\ \mathcal{R}_{NewScope}\ \langle \mu'_1, \hat{e}_1, \hat{r}_1 \rangle$ *(hyp.3),*
- $lev(\Sigma_0(r_f)) \sqsubseteq \sigma$ *(hyp.4),*
- $lev(\pi_{arg}\left(\Sigma_0(r_f)\right)) \sqsubseteq \sigma \Rightarrow v_0 = v_1$ *and* $lev(\pi_{this}\left(\Sigma_0(r_f)\right)) \sqsubseteq \sigma \Rightarrow r_0 = r_1$ *(hyp.5).*

It holds that: $\hat{e}_0 = \hat{e}_1$, $\hat{r}_0 = \hat{r}_1$, and $\bar{\Gamma}, \hat{r}_0 \Vdash \mu'_0 \sim_\sigma \mu'_1$, where:

- $\mu_0(r_f)(@code) = \mu_1(r_f)(@code) = \lambda^{(\hat{\Gamma}, \Sigma_0(r_f))} x.\, \{\mathsf{var}\ y_1, \cdots, y_n;\ \hat{e}_0\}$ *for some typing environment $\hat{\Gamma}$ and variables $x, y_1, \cdots, y_n$,*
- $\bar{\Gamma} = \hat{\Gamma}\left[\begin{array}{l} \mathsf{this} \mapsto \pi_{this}\left(\Sigma_0(r_f)\right), x \mapsto \pi_{arg}\left(\Sigma_0(r_f)\right), \\ y_1 \mapsto \pi_{var}\left(\Sigma_0(r_f), 1\right), \cdots, y_n \mapsto \pi_{var}\left(\Sigma_0(r_f), n\right) \end{array}\right]$

Lemma 6 (Confinement). *For any memory μ, labeling Σ, reference r, expression e, typing environment Γ, and security level σ such that: $r \vdash \langle \mu, \Sigma, e\rangle \Downarrow \langle \mu', \Sigma', v\rangle$, $\Gamma \vdash e : \dot{\tau}, \hat{\sigma}$, and the scope-chain starting from the object pointed to by r in μ is well-labeled by Γ, for some security type $\dot{\tau}$ and security level $\hat{\sigma}$; it holds that: $\mu \restriction^{\Sigma,\sigma} = \mu' \restriction^{\Sigma',\sigma}$ and $(\mu, r) \restriction^{\Gamma,\sigma} = (\mu', r) \restriction^{\Gamma,\sigma}$.*

Lemma 7 (Noninterference). *Given two memories μ and μ' well-labeled by Σ and Σ' resp., an expression e, a reference r, a typing environment Γ, and a reference r such that: $\Gamma \vdash e : \dot{\tau}, \hat{\sigma}$ (hyp.1), $r \vdash \langle \mu, \Sigma, e\rangle \Downarrow \langle \mu_f, \Sigma_f, v\rangle$ (hyp.2), $r \vdash \langle \mu', \Sigma', e\rangle \Downarrow \langle \mu'_f, \Sigma'_f, v'\rangle$ (hyp.3), $\mu, \Sigma \sim_\sigma \mu', \Sigma'$ (hyp.4), and $\Gamma, r \Vdash \mu \sim_\sigma \mu'$(hyp.5); it holds that: (i) $\mu_f, \Sigma_f \sim_\sigma \mu'_f, \Sigma'_f$, (ii) $\Gamma, r \Vdash \mu_f \sim_\sigma \mu'_f$, and (iii) $lev(\dot{\tau}) \sqsubseteq \sigma \Rightarrow v = v'$.*

Proof. We proceed by induction on the derivation of *hyp.*2.

[Val] $e = v$ for some value v (hyp.6). We conclude that:

- $v_f = v'_f = v$ — (1) - hyp.2 + hyp.3 + hyp.6
- $lev(\dot{\tau}) \sqsubseteq \sigma \Rightarrow v_f = v'_f$ — **(2)** - (1)
- $\mu_f = \mu$, $\mu'_f = \mu'$, $\Sigma_f = \Sigma$, $\Sigma'_f = \Sigma'$ — (3) - hyp.2 + hyp.3 + hyp.6
- $\mu_f, \Sigma_f \sim_\sigma \mu'_f, \Sigma'_f$ — **(4)** - hyp.4 + (3)
- $\Gamma, r \Vdash \mu_f \sim_\sigma \mu'_f$ — **(5)** - hyp.5 + (3)

[This] $e = \texttt{this}$ (hyp.6). We conclude that:

- $v_f = \mu(r)(@this)$ and $v'_f = \mu'(r)(@this)$ — (1) - hyp.2 + hyp.3 + hyp.6
- $lev(\Gamma(\mathsf{this})) \sqsubseteq \sigma \Rightarrow v_f = v'_f$ — (2) - hyp.5 + (1)
- $\dot{\tau} = \Gamma(\mathsf{this})$ — (3) - hyp.1
- $lev(\dot{\tau}) \sqsubseteq \sigma \Rightarrow v_f = v'_f$ — **(4)** - (2) + (3)
- $\mu_f = \mu$, $\mu'_f = \mu'$, $\Sigma_f = \Sigma$, and $\Sigma'_f = \Sigma'$. — (5) - hyp.2 + hyp.3 + hyp.6
- $\mu_f, \Sigma_f \sim_\sigma \mu'_f, \Sigma'_f$ — **(6)** - hyp.4 + (5)
- $\Gamma, r \Vdash \mu_f \sim_\sigma \mu'_f$ — **(7)** - hyp.5 + (5)

[Variable] $e = x$, for some variable x (hyp.6). We conclude that:

- $v_f = \mu(r_x)(x)$ and $\langle \mu, r, x\rangle\ \mathcal{R}_{Scope}\ r_x$ for some reference r_x — (1) - hyp.2 + hyp.6
- $v'_f = \mu'(r'_x)(x)$ and $\langle \mu', r, x\rangle\ \mathcal{R}_{Scope}\ r'_x$ for some reference r'_x — (2) - hyp.3 + hyp.6
- $lev(\Gamma(x)) \sqsubseteq \sigma \Rightarrow v_f = v'_f$ — (3) - hyp.5 + (1) + (2)
- $\dot{\tau} = \Gamma(x)$ — (4) - hyp.1 + hyp.6
- $lev(\dot{\tau}) \sqsubseteq \sigma \Rightarrow v_f = v'_f$ — **(5)** - (3) + (4)
- $\mu = \mu_f$, $\mu' = \mu'_f$, $\Sigma = \Sigma_f$, and $\Sigma' = \Sigma'_f$. — (6) - hyp.2 + hyp.3 + hyp.6
- $\mu_f, \Sigma_f \sim_\sigma \mu'_f, \Sigma'_f$ — **(7)** - hyp.4 + (6)
- $\Gamma, r \Vdash \mu_f \sim_\sigma \mu'_f$ — **(8)** - hyp.5 + (6)

[Binary Operation] $e = e_0\ \mathsf{op}\ e_1$ for two exprs. e_0 and e_1 (hyp.6). We conclude that:

- $r \vdash \langle \mu, \Sigma, e_0\rangle \Downarrow \langle \mu_0, \Sigma_0, v_0\rangle$ and $r \vdash \langle \mu_0, \Sigma_0, e_1\rangle \Downarrow \langle \mu_f, \Sigma_f, v_1\rangle$ for some memory μ_0, labeling Σ_0, and two values v_0 and v_1 such that $v_f = v_0\ \mathsf{op}\ v_1$ — (1) - hyp.2 + hyp.6
- $r \vdash \langle \mu', \Sigma', e_0\rangle \Downarrow \langle \mu'_0, \Sigma'_0, v'_0\rangle$ and $r \vdash \langle \mu'_0, \Sigma'_0, e'_1\rangle \Downarrow \langle \mu'_f, \Sigma'_f, v'_1\rangle$ for some memory μ'_0, labeling Σ'_0, and two values v'_0 and v'_1 such that $v'_f = v_1\ \mathsf{op}\ v'_1$ — (2) - hyp.3 + hyp.6
- $\Gamma \vdash e_0 : \dot{\tau}_0, \sigma_0$ and $\Gamma \vdash e_1 : \dot{\tau}_1, \sigma_1$, where: $\dot{\tau} = \dot{\tau}_0 \curlyvee \dot{\tau}_1$ and $\hat{\sigma} = \sigma_0 \sqcap \sigma_1$ — (3) - hyp.1 + hyp.6

– $\mu_0, \Sigma_0 \sim_\sigma \mu'_0, \Sigma'_0, \Gamma, r \Vdash \mu_0 \sim_\sigma \mu'_0, lev(\dot{\tau}_0) \sqsubseteq \sigma \Rightarrow v_0 = v'_0$
(4) - **ih** + hyp.4 + hyp.5 + (1) + (2) + (3)
– $\mu_f, \Sigma_f \sim_\sigma \mu'_f, \Sigma'_f, \Gamma, r \Vdash \mu_f \sim_\sigma \mu'_f, lev(\dot{\tau}_1) \sqsubseteq \sigma \Rightarrow v_1 = v'_1$
(5) - **ih** + (1) + (2) + (3) + (4)
– $v_0 = v'_0 \wedge v_1 = v'_1 \Rightarrow v_f = v'_f$ (6) - (1) + (2)
– $lev(\dot{\tau}) \sqsubseteq \sigma \Rightarrow (lev(\dot{\tau}_0) \sqsubseteq \sigma) \wedge (lev(\dot{\tau}_1) \sqsubseteq \sigma)$ (7) - (3)
– $lev(\dot{\tau}) \sqsubseteq \sigma \Rightarrow v_f = v'_f$ **(8)** - (4)-(7)

[Variable Assignment] $e = x = e_0$ for some variable e and expression e_0 (hyp.6). We conclude that:

– $r \vdash \langle \mu, \Sigma, e_0 \rangle \Downarrow \langle \mu_0, \Sigma_f, v \rangle$, $\langle \mu_0, r, x \rangle \mathcal{R}_{Scope}\ r_x$ and $\mu_f = \mu_0\,[r_x \mapsto \mu_0(r_x)\,[x \mapsto v_f]]$ for some memory μ_0 and reference r_x. (1) - hyp.2 + hyp.6
– $r \vdash \langle \mu', \Sigma', e_0 \rangle \Downarrow \langle \mu'_0, \Sigma'_f, v' \rangle$, $\langle \mu'_0, r, x \rangle \mathcal{R}_{Scope}\ r'_x$ and $\mu'_f = \mu'_0\,[r'_x \mapsto \mu'_0(r'_x)\,[x \mapsto v'_f]]$ for some memory μ'_0 and reference r'_x. (2) - hyp.3 + hyp.6
– $\Gamma \vdash e_0 : \dot{\tau}, \sigma_0$, $\dot{\tau} \preceq \Gamma(x)$, and $\hat{\sigma} = \sigma \sqcap lev(\Gamma(x))$ (3) - hyp.1 + hyp.6
– $\mu_0, \Sigma_0 \sim_\sigma \mu'_0, \Sigma'_0, \Gamma, r \Vdash \mu_0 \sim_\sigma \mu'_0, lev(\dot{\tau}) \sqsubseteq \sigma \Rightarrow v_f = v'_f$
(4) - **ih** + hyp.4 + hyp.5 + (1) + (2) + (3)
– $\mu_f, \Sigma_f \sim_\sigma \mu'_f, \Sigma'_f$ **(5)** - (1) + (2) + (4)
– $lev(\Gamma(x)) \sqsubseteq \sigma \Rightarrow lev(\dot{\tau}) \sqsubseteq \sigma$ (6) - (3)
– $lev(\Gamma(x)) \sqsubseteq \sigma \Rightarrow v_f = v'_f$ (7) - (4) + (6)
– $\Gamma, r \Vdash \mu_f \sim_\sigma \mu'_f$ **(8)** - (1) + (2) + (4) + (7) + Lemma **??**

[Object Literal] $e = \{\}^{\dot{\tau}}$(hyp.6). We conclude that:

– $\hat{r} = fresh(\mu, \Sigma, lev(\dot{\tau}))$, $\mu_f = \mu\,[\hat{r} \mapsto [\_proto\_ \mapsto null]]$, $\Sigma_f = \Sigma\,[\hat{r} \mapsto \dot{\tau}]$, and $v_f = \hat{r}$. (1) - hyp.6
– $\hat{r}' = fresh(\mu', \Sigma', lev(\dot{\tau}))$, $\mu'_f = \mu'\,[r' \mapsto [\_proto\_ \mapsto null]]$, $\Sigma'_f = \Sigma'\,[r' \mapsto \dot{\tau}]$, and $v'_f = \hat{r}'$. (2) - hyp.6
– $\Gamma, r \Vdash \mu_f \sim_\sigma \mu'_f$ **(3)** - hyp.5 + (1) + (2)

We consider two cases: either $lev(\dot{\tau}) \sqsubseteq \sigma$ or $lev(\dot{\tau}) \not\sqsubseteq \sigma$. Suppose $lev(\dot{\tau}) \sqsubseteq \sigma$ (hyp.7):

– $\hat{r} = \hat{r}'$ (4) - hyp.4 + hyp.7 + (1) + (2)
– $\mu_f \restriction^{\Sigma_f, \sigma} = \mu \restriction^{\Sigma, \sigma} \cup \{(\hat{r}, \dot{\tau})\} \cup \{(\hat{r}, \_proto\_, null), (\hat{r}, \_proto\_)\}$ (5) - hyp.7 + (1)
– $\mu'_f \restriction^{\Sigma'_f, \sigma} = \mu' \restriction^{\Sigma', \sigma} \cup \{(\hat{r}, \dot{\tau})\} \cup \{(\hat{r}, \_proto\_, null), (\hat{r}, \_proto\_)\}$
(6) - hyp.7 + (2) + (4)
– $\mu_f, \Sigma_f \sim_\sigma \mu'_f, \Sigma'_f$ **(7)** - hyp.4 + (5) + (6)
– $lev(\dot{\tau}) \sqsubseteq \sigma \Rightarrow v_f = v'_f$ **(8)** - (1) + (2) + (4)

Suppose $lev(\dot{\tau}) \not\sqsubseteq \sigma$ (hyp.7):

– $\mu_f \restriction^{\Sigma_f, \sigma} = \mu \restriction^{\Sigma, \sigma}$ (9) - hyp.7 + (1)
– $\mu'_f \restriction^{\Sigma'_f, \sigma} = \mu' \restriction^{\Sigma', \sigma}$ (10) - hyp.7 + (2)
– $\mu_f, \Sigma_f \sim_\sigma \mu'_f, \Sigma'_f$ **(11)** - hyp.4 + (9) + (10)
– $lev(\dot{\tau}) \sqsubseteq \sigma \Rightarrow v_f = v'_f$ **(12)** - hyp.7

[Property Look-up] $e = e_0[e_1, P]$ for two expressions e_0 and e_1 (hyp.6). It follows:

– $r \vdash \langle \mu, \Sigma, e_0 \rangle \Downarrow \langle \mu_0, \Sigma_0, r_0 \rangle$ and $r \vdash \langle \mu_0, \Sigma_0, e_1 \rangle \Downarrow \langle \mu_f, \Sigma_f, m_1 \rangle$ for some memory μ_0, labeling Σ_0, reference r_0 and $\hat{r}$, and string m_1 such that: $\langle \mu_f, r_0, m_1 \rangle \mathcal{R}_{Proto}\ \hat{r}$, $\hat{r} \neq null \Rightarrow v_f = \mu_f(\hat{r})(m_1)$, $\hat{r} = null \Rightarrow v = undefined$. (1) - hyp.2 + hyp.6
– $r \vdash \langle \mu', \Sigma', e_0 \rangle \Downarrow \langle \mu'_0, \Sigma'_0, r'_0 \rangle$ and $r \vdash \langle \mu'_0, \Sigma'_0, e_1 \rangle \Downarrow \langle \mu'_f, \Sigma'_f, m'_1 \rangle$ for some memory μ'_0, labeling Σ'_0, reference r'_0 and $\hat{r}'$, and string m'_1 such that: $\langle \mu'_f, r'_0, m'_1 \rangle \mathcal{R}_{Proto}\ \hat{r}'$, $\hat{r}' \neq null \Rightarrow v'_f = \mu'_f(\hat{r}')(m'_1)$, $\hat{r} = null \Rightarrow v = undefined$. (2) - hyp.3 + hyp.6

– $\Gamma \vdash e_0 : \dot{\tau}_0, \sigma_0$, $\Gamma \vdash e_1 : \dot{\tau}_1, \sigma_1$, $\Rsh_{\uparrow}\,(\dot{\tau}_0, P) = (\sigma', \dot{\tau}_{lu})$, and $\dot{\tau} = \dot{\tau}_{lu}^{lev(\dot{\tau}_0) \sqcup lev(\dot{\tau}_1)}$
(3) - hyp.1 + hyp.6

– $\mu_0, \Sigma_0 \sim_\sigma \mu_0', \Sigma_0'$, $\Gamma, r \Vdash \mu_0 \sim_\sigma \mu_0'$, $lev(\dot{\tau}_0) \sqsubseteq \sigma \Rightarrow r_0 = r_0'$
(4) - **ih** + hyp.4 + hyp.5 + (1) + (2) + (3)

– $\mu_f, \Sigma_f \sim_\sigma \mu_f', \Sigma_f'$, $\Gamma, r \Vdash \mu_f \sim_\sigma \mu_f'$, $lev(\dot{\tau}_1) \sqsubseteq \sigma \Rightarrow m_1 = m_1'$
(5) - **ih** + (1) + (2) + (3) + (4)

It remains to prove that $lev(\dot{\tau}) \sqsubseteq \sigma \Rightarrow v_f = v_f'$. Assume that $lev(\dot{\tau}) \sqsubseteq \sigma$ (hyp.7). It follows that:

– $lev(\dot{\tau}_0) \sqcup lev(\dot{\tau}_1) \sqcup lev(\dot{\tau}_{lu}) \sqsubseteq \sigma$ (6) - hyp.7 + (3)

– $r_0 = r_0'$ and $m_1 = m_1'$ (7) - (4)-(6)

– $m_1 = m_1' \in P$ (8) - (1) + (2) + (7) + *Correct Annotation*

– $lev(\pi_{\texttt{type}}\,(\Rsh\,(\dot{\tau}_0, m_1))) \sqsubseteq lev(\pi_{\texttt{type}}\,(\Rsh_{\uparrow}\,(\dot{\tau}_0, P))) = lev(\dot{\tau}_{lu})$ (9) - (3) + (8)

– $lev(\pi_{\texttt{type}}\,(\Rsh\,(\dot{\tau}_0, m_1))) \sqcup lev(\dot{\tau}_0) \sqsubseteq \sigma$ (10) - (9)

– $\Sigma_f(r_0) = \Sigma_f'(r_0') \preceq \dot{\tau}_0$ (11) - (1) + (2) + (3) + *Well-labeled Memory*

– $lev(\Sigma_f(r_0)) = lev(\Sigma_f'(r_0')) \sqsubseteq lev(\dot{\tau}_0)$ (12) - (11)

– $\lfloor \Sigma_f(r_0) \rfloor = \lfloor \Sigma_f'(r_0') \rfloor = \lfloor \dot{\tau}_0 \rfloor$ (13) - (11)

– $\Rsh\,(\dot{\tau}_0, m_1) = \Rsh\,(\Sigma_f(r_0), m_1) = \Rsh\,(\Sigma_f'(r_0'), m_1')$ (14) - (13)

– $lev(\pi_{\texttt{type}}\,(\Rsh\,(\Sigma_f(r_0), m_1))) = lev(\pi_{\texttt{type}}\,(\Rsh\,(\Sigma_f'(r_0'), m_1'))) \sqsubseteq \sigma$ (15) - (10) + (14)

– $lev(\pi_{\texttt{type}}\,(\Rsh\,(\Sigma_f(r_0), m_1))) \sqcup lev(\Sigma_f(r_0)) \sqsubseteq \sigma$ (16) - (12) + (15)

– $\hat{r} = \hat{r}'$ and $\hat{r} \neq null \Rightarrow lev(\Sigma_f(\hat{r})) = lev(\Sigma_f'(\hat{r}')) \sqsubseteq \sigma$
(17) - (1) + (2) + (5) + (16) + Proto-Chain Indistinguishability (Lemma **??**)

We consider two cases: $\hat{r} \neq null$ or $\hat{r} = null$. Suppose $\hat{r} \neq null$ (hyp.8):

– $\hat{r}' \neq null$ (18) - hyp.8 + (17)

– $lev(\Sigma_f(\hat{r})) = lev(\Sigma_f'(\hat{r}')) \sqsubseteq \sigma$ (19) - hyp.8 + (17)

– $\Rsh\,(\Sigma_f(r_0), m_1) = \Rsh\,(\Sigma_f(\hat{r}), m_1)$ (20) - (1) + Well-Lab. Proto-Chains (Lemma **??**)

– $\Rsh\,(\Sigma_f'(r_0'), m_1) = \Rsh\,(\Sigma_f'(\hat{r}'), m_1)$ (21) - (2) + Well-Lab. Proto-Chains (Lemma **??**)

– $lev(\pi_{\texttt{type}}\,(\Rsh\,(\Sigma_f(\hat{r}), m_1))) = lev(\pi_{\texttt{type}}\,(\Rsh\,(\Sigma_f'(\hat{r}), m_1))) \sqsubseteq \sigma$
(22) - (15) + (20) + (21)

– $v_f = v_f'$ **(23)** - hyp.8 + (1) + (2) + (5) + (17) + (19) + (22)

Suppose $\hat{r} = null$ (hyp.8):

– $\hat{r}' = null$ (24) - hyp.8 + (15)

– $v_f = v_f' = undefined$ (25) - hyp.8 + (1) + (2) + (24)

[In Expression] $e = e_0\ \mathsf{in}^P\ e_1$ for two expressions e_0 and e_1 (hyp.6). It follows that:

– $r \vdash \langle \mu, \Sigma, e_0 \rangle \Downarrow \langle \mu_0, \Sigma_0, m_0 \rangle$ and $r \vdash \langle \mu_0, \Sigma_0, e_1 \rangle \Downarrow \langle \mu_f, \Sigma_f, r_1 \rangle$ for some memory μ_0, labeling Σ_0, a reference r_1, and a string m_0 such that: $\langle \mu_f, r_0, m_0 \rangle\ \mathcal{R}_{Proto}\ \hat{r}$, $\hat{r} \neq null \Rightarrow v = \texttt{ff}$, and $\hat{r} = null \Rightarrow v = \texttt{tt}$ (1) - hyp.2 + hyp.6

– $r \vdash \langle \mu, \Sigma, e_0 \rangle \Downarrow \langle \mu_0, \Sigma_0, m_0 \rangle$ and $r \vdash \langle \mu_0, \Sigma_0, e_1 \rangle \Downarrow \langle \mu_f, \Sigma_f, r_1 \rangle$ for some memory μ_0, labeling Σ_0, a reference r_1, and a string m_0 such that: $\langle \mu_f, r_0, m_0 \rangle\ \mathcal{R}_{Proto}\ \hat{r}$, $\hat{r} \neq null \Rightarrow v = \texttt{ff}$, and $\hat{r} = null \Rightarrow v = \texttt{tt}$ (2) - hyp.2 + hyp.6

– $\Gamma \vdash e_0 : \dot{\tau}_0, \sigma_0$ and $\Gamma \vdash e_1 : \dot{\tau}_1, \sigma_1$, where: $\dot{\tau} = \mathsf{PRIM}^{lev(\dot{\tau}_0) \sqcup lev(\dot{\tau}_1) \sqcup \pi_{\texttt{1ev}}(\Rsh_{\uparrow}\,(\dot{\tau}_1, P))}$ and $\hat{\sigma} = \sigma_0 \sqcap \sigma_1$ (3) - hyp.1 + hyp.6

– $\mu_0, \Sigma_0 \sim_\sigma \mu_0', \Sigma_0'$, $\Gamma, r \Vdash \mu_0 \sim_\sigma \mu_0'$, $lev(\dot{\tau}_0) \sqsubseteq \sigma \Rightarrow m_0 = m_0'$
(4) - **ih** + hyp.4 + hyp.5 + (1) + (2) + (3)

– $\mu_f, \Sigma_f \sim_\sigma \mu_f', \Sigma_f'$, $\Gamma, r \Vdash \mu_f \sim_\sigma \mu_f'$, $lev(\dot{\tau}_1) \sqsubseteq \sigma \Rightarrow r_1 = r_1'$
(5) - **ih** + (1) + (2) + (3) + (4)

It remains to prove that $lev(\dot{\tau}) \sqsubseteq \sigma \Rightarrow v = v'$. Assume that $lev(\dot{\tau}) \sqsubseteq \sigma$ (hyp.7). It follows that:

- $lev(\dot{\tau}_0) \sqcup lev(\dot{\tau}_1) \sqcup \pi_{\mathtt{lev}}\,(\Rsh_{\uparrow}\,(\dot{\tau}_1, P)) \sqsubseteq \sigma$ (6) - hyp.7 + (3)
- $m_0 = m_0'$ and $r_1 = r_1'$ (7) - (4)-(6)
- $m_0 = m_0' \in P$ (8) - (1) + (2) + (7) + *Correct Annotation*
- $\pi_{\mathtt{lev}}\,(\Rsh\,(\dot{\tau}_1, m_0)) \sqsubseteq \pi_{\mathtt{lev}}\,(\Rsh_{\uparrow}\,(\dot{\tau}_1, P)) \sqsubseteq \sigma$ (9) - (6) + (8)
- $\pi_{\mathtt{lev}}\,(\Rsh\,(\dot{\tau}_1, m_0)) \sqsubseteq \sigma$ (10) - (6) + (9)
- $\Sigma_f(r_1) \curlyvee \Sigma_f'(r_1') \preceq \dot{\tau}_1$ (11) - (1) - (3) + *Well-labeled Memory*
- $\Sigma_f(r_1) = \Sigma_f'(r_1') \preceq \dot{\tau}_1$ (12) - (5) + (6) + (11)
- $\lfloor \Sigma_f(r_1) \rfloor \equiv \lfloor \dot{\tau}_1 \rfloor$ and $lev(\Sigma_f(r_1)) \sqsubseteq lev(\dot{\tau}_1)$ (13) - (12)
- $\pi_{\mathtt{lev}}\,(\Rsh\,(\Sigma_f(r_1), m_0)) = \pi_{\mathtt{lev}}\,(\Rsh\,(\dot{\tau}_1, m_0)) \sqsubseteq \sigma$ (14) - (10) + (13)
- $lev(\Sigma_f(r_1)) \sqsubseteq \sigma$ (15) - (6) + (11)
- $\pi_{\mathtt{lev}}\,(\Rsh\,(\Sigma_f(r_1), m_0)) \sqcup lev(\Sigma_f(r_1)) \sqsubseteq \sigma$ (16) - (14) + (15)
- $\hat{r} = \hat{r}'$ (17) - (1) + (2) + (5) + (7) + (16)
- $v = v'$ (18) - (1) + (2) + (17) + Proto-Chain Indistinguishability (Lemma **??**)

[Property Assignment] $e = e_0[e_1] = e_2$ for three expressions e_0, e_1, and e_2 (hyp.6). We conclude that:

- $r \vdash \langle \mu, \Sigma, e_0 \rangle \Downarrow \langle \mu_0, \Sigma_0, r_0 \rangle$, $r \vdash \langle \mu_0, \Sigma_0, e_1 \rangle \Downarrow \langle \mu_1, \Sigma_1, m_1 \rangle$, $r \vdash \langle \mu_1, \Sigma_1, e_2 \rangle \Downarrow \langle \mu_2, \Sigma_f, v_f \rangle$ for three memories μ_0, μ_1, and μ_2, two labelings Σ_0 and Σ_1, a reference r_0, and a string m_1 such that: $\mu_f = \mu_2\,[r_0 \mapsto \mu_2(r_0)\,[m_1 \mapsto v_f]]$ (1) - hyp.2 + hyp.6
- $r \vdash \langle \mu', \Sigma', e_0 \rangle \Downarrow \langle \mu_0', \Sigma_0', r_0' \rangle$, $r \vdash \langle \mu_0', \Sigma_0', e_1 \rangle \Downarrow \langle \mu_1', \Sigma_1', m_1' \rangle$, $r \vdash \langle \mu_1', \Sigma_1', e_2 \rangle \Downarrow \langle \mu_2', \Sigma_f', v_f' \rangle$ for three memories μ_0', μ_1', and μ_2', two labelings Σ_0' and Σ_1', a reference r_0', and a string m_1' such that: $\mu_f' = \mu_2'\,[r_0' \mapsto \mu_2'(r_0')\,[m_1' \mapsto v_f']]$ (2) - hyp.3 + hyp.6
- $\Gamma \vdash e_0 : \dot{\tau}_0, \sigma_0$, $\Gamma \vdash e_1 : \dot{\tau}_1, \sigma_1$, and $\Gamma \vdash e_2 : \dot{\tau}_2, \sigma_2$ where: $\dot{\tau} = \dot{\tau}_2$, $\dot{\tau}_2 \preceq \pi_{\mathtt{type}}\,(\Rsh_{\downarrow}\,(\dot{\tau}_0, P))$, $lev(\dot{\tau}_0) \sqcup lev(\dot{\tau}_1) \sqsubseteq \pi_{\mathtt{lev}}\,(\Rsh_{\downarrow}\,(\dot{\tau}_0, P))$ (3) - hyp.1 + hyp.6
- $\mu_0, \Sigma_0 \sim_\sigma \mu_0', \Sigma_0'$, $\Gamma, r \Vdash \mu_0 \sim_\sigma \mu_0'$, $lev(\dot{\tau}_0) \sqsubseteq \sigma \Rightarrow r_0 = r_0'$ (4) - **ih** + hyp.4 + hyp.5 + (1) + (2) + (3)
- $\mu_1, \Sigma_1 \sim_\sigma \mu_1', \Sigma_1'$, $\Gamma, r \Vdash \mu_1 \sim_\sigma \mu_1'$, $lev(\dot{\tau}_1) \sqsubseteq \sigma \Rightarrow m_1 = m_1'$ (5) - **ih** + (1) + (2) + (3) + (4)
- $\mu_2, \Sigma_f \sim_\sigma \mu_2', \Sigma_f'$, $\Gamma, r \Vdash \mu_2 \sim_\sigma \mu_2'$, $lev(\dot{\tau}_2) \sqsubseteq \sigma \Rightarrow v_f = v_f'$ **(6)** - **ih** + (1) + (2) + (3) + (5)

We distinguish two different cases, either $lev(\dot{\tau}_0) \sqcup lev(\dot{\tau}_1) \sqsubseteq \sigma$ or $lev(\dot{\tau}_0) \sqcup lev(\dot{\tau}_1) \not\sqsubseteq \sigma$. Suppose $lev(\dot{\tau}_0) \sqcup lev(\dot{\tau}_1) \not\sqsubseteq \sigma$ (hyp.7), it follows that:

- $r_0 = r_0'$ and $m_1 = m_1'$ (7) - hyp.7 + (4) + (5)
- $\Sigma_f(r_0) \curlyvee \Sigma_f'(r_0) \preceq \dot{\tau}_0$ (8) - (1) - (3) + (7) + *Well-labeled Memory*
- $\Sigma_f(r_0) = \Sigma_f'(r_0) \preceq \dot{\tau}_0$ (9) - hyp.7 + (6) + (8)
- $\lfloor \Sigma_f(r_0) \rfloor \equiv \lfloor \Sigma_f'(r_0) \rfloor \equiv \lfloor \dot{\tau}_0 \rfloor$ (10) - (9)
- $lev(\Sigma_f(r_0)) = lev(\Sigma_f'(r_0)) \sqsubseteq lev(\dot{\tau}_0) \sqsubseteq \sigma$ (11) - hyp.7 + (9)
- $m_1 = m_1' \in P$ (12) - (1) + (2) + (7) + *Correct Annotation*
- $\pi_{\mathtt{type}}\,(\Rsh_{\downarrow}\,(\dot{\tau}_0, P)) \preceq \pi_{\mathtt{type}}\,(\Rsh\,(\dot{\tau}_0, m_1))$ (13) - (12)
- $\pi_{\mathtt{type}}\,(\Rsh\,(\dot{\tau}_0, m_1)) = \pi_{\mathtt{type}}\,(\Rsh\,(\Sigma_f(r_0), m_1))$ (14) - (10)
- $\dot{\tau}_2 \preceq \pi_{\mathtt{type}}\,(\Rsh\,(\Sigma_f(r_0), m_1))$ (15) - (3) + (13) + (14)
- $lev(\Sigma_f(r_0)) \sqcup lev(\pi_{\mathtt{type}}\,(\Rsh\,(\Sigma_f(r_0), m_1))) \sqsubseteq \sigma \Rightarrow lev(\dot{\tau}_2) \sqsubseteq \sigma$ (16) - (11) + (15)
- $lev(\Sigma_f(r_0)) \sqcup lev(\pi_{\mathtt{type}}\,(\Rsh\,(\Sigma_f(r_0), m_1))) \sqsubseteq \sigma \Rightarrow v_f = v_f'$ (17) - (6) + (16)
- $\mu_f, \Sigma_f \sim_\sigma \mu_f', \Sigma_f'$ **(18)** - (1) + (2) + (6) + (17) + Lemma **??**
- $\Gamma, r \Vdash \mu_f \sim_\sigma \mu_f'$ **(19)** - (1) + (2) + (6)

Suppose $lev(\dot{\tau}_0) \sqcup lev(\dot{\tau}_1) \not\sqsubseteq \sigma$ (hyp.7), it follows that:

- $\Sigma_f(r_0) \curlyvee \Sigma'_f(r'_0) \preceq \dot{\tau}_0$ (20) - (1) - (3) + *Well-labeled Memory*
- $\lfloor \Sigma_f(r_0) \rfloor \equiv \lfloor \Sigma'_f(r'_0) \rfloor \equiv \lfloor \dot{\tau}_0 \rfloor$ (21) - (20)
- $\{m_1, m'_1\} \subseteq P$ (22) - (1) + (2) + *Correct Annotation*
- $\pi_{\texttt{lev}}(\Rsh_{\downarrow}(\dot{\tau}_0, P)) \sqsubseteq \pi_{\texttt{lev}}(\Rsh(\dot{\tau}_0, m_1)) \sqcap \pi_{\texttt{lev}}(\Rsh(\dot{\tau}_0, m'_1))$ (23) - (22)
- $\pi_{\texttt{lev}}(\Rsh(\dot{\tau}_0, m_1)) \sqcap \pi_{\texttt{lev}}(\Rsh(\dot{\tau}_0, m'_1)) \not\sqsubseteq \sigma$ (24) - (3) + (23)
- $\mu_f, \Sigma_f \sim_\sigma \mu'_f, \Sigma'_f$ **(25)** - (1) + (2) + (6) + (24) + *Invisible Property Assignment*
- $\Gamma, r \Vdash \mu_f \sim_\sigma \mu'_f$ **(26)** - (1) + (2) + (6)

[Function Call] $e = e_0(e_1)$ for two expressions e_0 and e_1(hyp.6). We conclude that:

- $r \vdash \langle \mu, \Sigma, e_0 \rangle \Downarrow \langle \mu_0, \Sigma_0, r_0 \rangle$, $r \vdash \langle \mu_0, \Sigma_0, e_1 \rangle \Downarrow \langle \mu_1, \Sigma_1, v_1 \rangle$, $\hat{r} \vdash \langle \hat{\mu}, \Sigma_1, \hat{e} \rangle \Downarrow \langle \mu_f, \Sigma_f, v_f \rangle$ for three mems. μ_0, μ_1, and $\hat{\mu}$, two labs. Σ_0 and Σ_1, two refs. r_0 and $\hat{r}$, a value v_1, and an expr. $\hat{e}$ such that: $\langle \mu_1, r_0, v_1, \#glob \rangle \; \mathcal{R}_{NewScope} \; \langle \hat{\mu}, \hat{e}, \hat{r} \rangle$ (1) - hyp.2 + hyp.6
- $r \vdash \langle \mu', \Sigma', e_0 \rangle \Downarrow \langle \mu'_0, \Sigma'_0, r'_0 \rangle$, $r \vdash \langle \mu'_0, \Sigma'_0, e_1 \rangle \Downarrow \langle \mu'_1, \Sigma'_1, v'_1 \rangle$, $\hat{r}' \vdash \langle \hat{\mu}', \Sigma'_1, \hat{e}' \rangle \Downarrow \langle \mu'_f, \Sigma'_f, v'_f \rangle$ for three mems. μ'_0, μ'_1, and $\hat{\mu}'$, two labs. Σ'_0 and Σ'_1, two refs. r'_0 and $\hat{r}'$, a value v'_1, and an expr. $\hat{e}'$ such that: $\langle \mu'_1, r'_0, v'_1, \#glob \rangle \; \mathcal{R}_{NewScope} \; \langle \hat{\mu}', \hat{e}', \hat{r}' \rangle$ (2) - hyp.2 + hyp.6
- $\Gamma \vdash e_0 : \dot{\tau}_0, \sigma_0$ and $\Gamma \vdash e_1 : \dot{\tau}_1, \sigma_1$, where: $\dot{\tau}_0 = \langle \dot{\tau}'_0 . \dot{\tau}'_1 \xrightarrow{\hat{\sigma}} \dot{\tau}'_2 \rangle^{\hat{\sigma}'}$, $\dot{\tau}_{global} \preceq \dot{\tau}'_0$, $\dot{\tau}_1 \preceq \dot{\tau}'_1$, $lev(\dot{\tau}_0) \sqsubseteq \hat{\sigma}$, and $\dot{\tau} = (\dot{\tau}'_2)^{lev(\dot{\tau}_0)}$. (3) - hyp.1 + hyp.6
- $\mu_0, \Sigma_0 \sim_\sigma \mu'_0, \Sigma'_0$, $\Gamma, r \Vdash \mu_0 \sim_\sigma \mu'_0$, $lev(\dot{\tau}_0) \sqsubseteq \sigma \Rightarrow r_0 = r'_0$ (4) - **ih** + hyp.4 + hyp.5 + (1) + (2) + (3)
- $\mu_1, \Sigma_1 \sim_\sigma \mu'_1, \Sigma'_1$, $\Gamma, r \Vdash \mu_1 \sim_\sigma \mu'_1$, $lev(\dot{\tau}_1) \sqsubseteq \sigma \Rightarrow m_1 = m'_1$ (5) - **ih** + (1) + (2) + (3) + (4)

We consider two cases: $lev(\dot{\tau}_0) \sqsubseteq \sigma$ and $lev(\dot{\tau}_0) \not\sqsubseteq \sigma$. Suppose $lev(\dot{\tau}_0) \sqsubseteq \sigma$ (hyp.7). It follows that:

- $r_0 = r'_0$ (6) - hyp.7 + (4)
- $\Sigma_1(r_0) \curlyvee \Sigma'_1(r_0) \preceq \dot{\tau}_0$ (7) - (1) - (3) + (6) + *Well-labeled Memory*
- $\Sigma_1(r_0) = \Sigma'_1(r_0) \preceq \dot{\tau}_0$ (8) - hyp.7 + (5) + (7)
- $\lfloor \Sigma_1(r_0) \rfloor \equiv \lfloor \Sigma'_1(r_0) \rfloor \equiv \lfloor \dot{\tau}_0 \rfloor \equiv \langle \dot{\tau}'_0 . \dot{\tau}'_1 \xrightarrow{\hat{\sigma}} \dot{\tau}'_2 \rangle$ (9) - (8)
- $lev(\Sigma_1(r_0)) = lev(\Sigma'_1(r_0)) \sqsubseteq lev(\dot{\tau}_0) \sqsubseteq \sigma$ (10) - hyp.7 + (9)
- $\begin{cases} \mu_1(r_0)(@code) = \mu'_1(r_0)(@code) = \lambda^{(\hat{\Gamma}, \Sigma_1(r_0))} x. \left\{ \mathsf{var}^{\dot{\tau}_{y_1}, \cdots, \dot{\tau}_{y_n}} \; y_1, \cdots, y_n; \; \hat{e} \right\} \\ \mu_1(r_0)(@fscope) = \mu'_1(r_0)(@fscope) = \hat{r} = \hat{r}' \\ \hat{\Gamma}, \hat{r} \Vdash \mu_1 \sim_\sigma \mu'_1 \\ \hat{e} = \hat{e}' \end{cases}$

  for some typing environment $\hat{\Gamma}$ and variables $x, y_1, \cdots, y_n$ (11) - (5) + (9) + (10)
- $\bar{\Gamma} \vdash \hat{e} : \dot{\tau}'_2, \hat{\sigma}$, where $\bar{\Gamma} = \hat{\Gamma}[\mathsf{this} \mapsto \dot{\tau}'_0, x \mapsto \dot{\tau}'_1, y_1 \mapsto \dot{\tau}_{y_1}, \cdots, y_n \mapsto \dot{\tau}_{y_n}]$ (12) - (11) + *Well-labeled Memory*
- $lev(\dot{\tau}'_0) \sqsubseteq \sigma \Rightarrow \#glob = \#glob$ (13) - *tautology*
- $lev(\dot{\tau}'_1) \sqsubseteq \sigma \Rightarrow lev(\dot{\tau}_1) \sqsubseteq \sigma$ (14) - (3)
- $lev(\dot{\tau}'_1) \sqsubseteq \sigma \Rightarrow v_1 = v'_1$ (15) - (5) + (14)
- $\bar{\Gamma}, \hat{r} \Vdash \hat{\mu} \sim_\sigma \hat{\mu}'$ (16) - (1) + (2) + (5) + (10) + (13) + (15) + *Indist. Scope Alloc*
- $\hat{\mu}, \Sigma_1 \sim_\sigma \hat{\mu}', \Sigma'_1$ (17) - (1) + (2) + (5)
- $\mu_f, \Sigma_f \sim_\sigma \mu'_f, \Sigma'_f$, $\Gamma, r \Vdash \mu_f \sim_\sigma \mu'_f$, $lev(\dot{\tau}'_2) \sqsubseteq \sigma \Rightarrow v_f = v'_f$ **(18.1)** - **ih** + (1) + (2) + (12) + (17)
- $lev(\dot{\tau}) \sqsubseteq \sigma \Rightarrow lev(\dot{\tau}'_2) \sqsubseteq \sigma$ (18.2) - (3)
- $lev(\dot{\tau}) \sqsubseteq \sigma \Rightarrow v_f = v'_f$ **(18.3)** - (18.1) + (18.2)

Suppose $lev(\dot{\tau}_0) \not\sqsubseteq \sigma$ (hyp.7). It follows that:

- $\hat{\sigma} \not\sqsubseteq \sigma$ (19) - hyp.7 + (3)
- $\Sigma_1(r_0) \curlyvee \Sigma_1'(r_0') \preceq \dot{\tau}_0$ (20) - (1) - (3) + *Well-labeled Memory*
- $\lfloor \Sigma_1(r_0) \rfloor \equiv \lfloor \Sigma_1'(r_0') \rfloor \equiv \lfloor \dot{\tau}_0 \rfloor \equiv \langle \dot{\tau}_0'.\dot{\tau}_1' \xrightarrow{\hat{\sigma}} \dot{\tau}_2' \rangle$ (22) - (21)
- $\begin{cases} \mu_1(r_0)(@code) = \lambda^{(\hat{\Gamma}, \Sigma_1(r_0))} x. \left\{ \mathsf{var}^{\dot{\tau}_{y_1}, \cdots, \dot{\tau}_{y_n}} \ y_1, \cdots, y_n; \ \hat{e} \right\} \\ \mu_1(r_0)(@fscope) = \hat{r} \\ \bar{\Gamma} \vdash \hat{e} : \dot{\tau}_2', \hat{\sigma} \\ \bar{\Gamma} = \hat{\Gamma} \left[\mathsf{this} \mapsto \dot{\tau}_0', x \mapsto \dot{\tau}_1', y_1 \mapsto \dot{\tau}_{y_1}, \cdots, y_n \mapsto \dot{\tau}_{y_n} \right] \end{cases}$ (23) - (22) + *Well-labeled Memory*
- $\begin{cases} \mu_1'(r_0')(@code) = \lambda^{(\hat{\Gamma}', \Sigma_1'(r_0'))} x'. \left\{ \mathsf{var}^{\dot{\tau}'_{y_1'}, \cdots, \dot{\tau}'_{y_k'}} \ y_1', \cdots, y_k'; \ \hat{e}' \right\} \\ \mu_1'(r_0')(@fscope) = \hat{r}' \\ \bar{\Gamma}' \vdash \hat{e}' : \dot{\tau}_2', \hat{\sigma} \\ \bar{\Gamma}' = \hat{\Gamma}' \left[\mathsf{this} \mapsto \dot{\tau}_0', x' \mapsto \dot{\tau}_1', y_1' \mapsto \dot{\tau}'_{y_1'}, \cdots, y_n' \mapsto \dot{\tau}'_{y_n'} \right] \end{cases}$ (24) - (22) + *Well-labeled Memory*
- $\hat{\mu} \restriction^{\Sigma_1, \sigma} = \mu_1 \restriction^{\Sigma_1, \sigma}$ and $(\hat{\mu}, r) \restriction^{\Gamma, \sigma} = (\mu_1, r) \restriction^{\Gamma, \sigma}$ (25) - (1)
- $\mu_f \restriction^{\Sigma_f, \sigma} = \hat{\mu} \restriction^{\Sigma_1, \sigma}$ and $(\mu_f, \hat{r}) \restriction^{\bar{\Gamma}, \sigma} = (\hat{\mu}, \hat{r}) \restriction^{\bar{\Gamma}, \sigma}$ (26) - (1) + (23) + Confinement (Lemma **??**)
- $\hat{\mu}' \restriction^{\Sigma_1', \sigma} = \mu_1' \restriction^{\Sigma_1', \sigma}$ and $(\hat{\mu}', r) \restriction^{\Gamma, \sigma} = (\mu_1', r) \restriction^{\Gamma, \sigma}$ (27) - (2)
- $\mu_f' \restriction^{\Sigma_f', \sigma} = \hat{\mu}' \restriction^{\Sigma_1', \sigma}$ and $(\mu_f', \hat{r}') \restriction^{\bar{\Gamma}', \sigma} = (\hat{\mu}', \hat{r}') \restriction^{\bar{\Gamma}', \sigma}$ (28) - (2) + (24) + Confinement (Lemma **??**)
- $\mu_1 \restriction^{\Sigma_1, \sigma} = \mu_1' \restriction^{\Sigma_1', \sigma}$ and $(\mu_1, r) \restriction^{\Gamma, \sigma} = (\mu_1', r) \restriction^{\Gamma, \sigma}$ (29) - (5)
- $\mu_f \restriction^{\Sigma_f, \sigma} = \mu_f' \restriction^{\Sigma_f', \sigma} \Leftrightarrow \mu_f, \Sigma_f \sim_\sigma \mu_f', \Sigma_f'$ **(30)** - (25)-(29)
- $\Gamma, r \Vdash \mu_f \sim_\sigma \mu_f'$ **(31)** - (25)-(29)
- $lev(\dot{\tau}) \not\sqsubseteq \sigma$ (32) - hyp.7 + (3)
- $lev(\dot{\tau}) \sqsubseteq \sigma \Rightarrow v_f = v_f'$ **(33)** - (32)

[Method Call] $e = e_0[e_1, P](e_2)$ for two exprs. e_0 and e_1(hyp.6). We conclude that:

- $r \vdash \langle \mu, \Sigma, e_0 \rangle \Downarrow \langle \mu_0, \Sigma_0, r_0 \rangle$, $r \vdash \langle \mu_0, \Sigma_0, e_1 \rangle \Downarrow \langle \mu_1, \Sigma_1, m_1 \rangle$, $r \vdash \langle \mu_1, \Sigma_1, e_2 \rangle \Downarrow \langle \mu_2, \Sigma_2, v_2 \rangle$, $\hat{r} \vdash \langle \hat{\mu}, \Sigma_2, \hat{e} \rangle \Downarrow \langle \mu_f, \Sigma_f, v_f \rangle$ for four mems. μ_0, μ_1, μ_2, and $\hat{\mu}$, three labs. Σ_0, Σ_1, and Σ_2, two refs. r_0 and $\hat{r}$, a str. m_1, a val. v_2, and an expr. $\hat{e}$ s.t.: $\langle \mu_2, r_0, m_1 \rangle \ \mathcal{R}_{Proto} \ r_m$, $r_f = \mu_2(r_m)(m_1)$, and $\langle \mu_2, r_f, v_2, r_0 \rangle \ \mathcal{R}_{NewScope} \ \langle \hat{\mu}, \hat{e}, \hat{r} \rangle$ (1) - hyp.2 + hyp.6
- $r \vdash \langle \mu', \Sigma', e_0 \rangle \Downarrow \langle \mu_0', \Sigma_0', r_0' \rangle$, $r \vdash \langle \mu_0', \Sigma_0', e_1 \rangle \Downarrow \langle \mu_1', \Sigma_1', m_1' \rangle$, $r \vdash \langle \mu_1', \Sigma_1', e_2 \rangle \Downarrow \langle \mu_2', \Sigma_2', v_2 \rangle$, $\hat{r}' \vdash \langle \hat{\mu}', \Sigma_2', \hat{e}' \rangle \Downarrow \langle \mu_f', \Sigma_f', v_f' \rangle$ for four mems. μ_0', μ_1', μ_2', and $\hat{\mu}'$, three labs. Σ_0', Σ_1', and Σ_2', two refs. r_0' and $\hat{r}'$, a str. m_1', a val. v_2', and an expr. $\hat{e}'$ s.t.: $\langle \mu_2', r_0', m_1' \rangle \ \mathcal{R}_{Proto} \ r_m'$, $r_f' = \mu_2'(r_m')(m_1')$, and $\langle \mu_2', r_f', v_2', r_0' \rangle \ \mathcal{R}_{NewScope} \ \langle \hat{\mu}', \hat{e}', \hat{r}' \rangle$ (2) - hyp.2 + hyp.6
- $\Gamma \vdash e_i : \dot{\tau}_i, \sigma_i$: $i \in \{0, 1, 2\}$, $\Gamma_\uparrow (\dot{\tau}_0, P) = (\hat{\sigma}, \langle \dot{\tau}_0'.\dot{\tau}_1' \xrightarrow{\hat{\sigma}'} \dot{\tau}_2' \rangle^{\hat{\sigma}''})$, $\sigma' = \hat{\sigma}'' \sqcup lev(\dot{\tau}_0) \sqcup lev(\dot{\tau}_1)$, $\dot{\tau}_0 \preceq \dot{\tau}_0'$, $\dot{\tau}_2 \preceq \dot{\tau}_1'$, $\sigma' \sqsubseteq \hat{\sigma}'$, and $\dot{\tau} = (\dot{\tau}_2')^{\sigma'}$. (3) - hyp.1 + hyp.6
- $\mu_0, \Sigma_0 \sim_\sigma \mu_0', \Sigma_0'$, $\Gamma, r \Vdash \mu_0 \sim_\sigma \mu_0'$, $lev(\dot{\tau}_0) \sqsubseteq \sigma \Rightarrow r_0 = r_0'$ (4) - **ih** + hyp.4 + hyp.5 + (1) + (2) + (3)
- $\mu_1, \Sigma_1 \sim_\sigma \mu_1', \Sigma_1'$, $\Gamma, r \Vdash \mu_1 \sim_\sigma \mu_1'$, $lev(\dot{\tau}_1) \sqsubseteq \sigma \Rightarrow m_1 = m_1'$ (5) - **ih** + (1) + (2) + (3) + (4)
- $\mu_2, \Sigma_2 \sim_\sigma \mu_2', \Sigma_2'$, $\Gamma, r \Vdash \mu_2 \sim_\sigma \mu_2'$, $lev(\dot{\tau}_2) \sqsubseteq \sigma \Rightarrow v_2 = v_2'$ (6) - **ih** + (1) + (2) + (3) + (5)

We consider two cases: $\sigma' \sqsubseteq \sigma$ and $\sigma' \not\sqsubseteq \sigma$. Suppose $\sigma' \sqsubseteq \sigma$ (hyp.7). It follows that:

- $\hat{\sigma}'' \sqcup lev(\dot{\tau}_0) \sqcup lev(\dot{\tau}_1) \sqsubseteq \sigma$ (7) - hyp.7 + (3)
- $r_0 = r'_0$ and $m_1 = m'_1$ (8) - (4) + (5) + (7)
- $m_1 = m'_1 \in P$ (9) - (1) + (2) + (8) + *Correct Annotation*
- $\pi_{\texttt{type}}\left(\Cap\left(\dot{\tau}_0, m_1\right)\right) \preceq \pi_{\texttt{type}}\left(\Cap_{\uparrow}\left(\dot{\tau}_0, P\right)\right) = \langle \dot{\tau}'_0 . \dot{\tau}'_1 \xrightarrow{\hat{\sigma}'} \dot{\tau}'_2 \rangle^{\hat{\sigma}''}$ (10) - (3) + (9)
- $\Sigma_2(r_0) \curlyvee \Sigma'_2(r'_0) \preceq \dot{\tau}_0$ (11) - (1) + (2) + (3) + *Well-labeled Memory*
- $\Sigma_2(r_0) = \Sigma'_2(r_0) \preceq \dot{\tau}_0$ (12) - hyp.7 + (6)-(8) + (11)
- $lev(\Sigma_2(r_0)) = lev(\Sigma'_2(r_0)) \sqsubseteq lev(\dot{\tau}_0)$ (13) - (12)
- $\lfloor \Sigma_2(r_0) \rfloor = \lfloor \Sigma'_2(r_0) \rfloor = \lfloor \dot{\tau}_0 \rfloor$ (14) - (12)
- $\Cap\left(\dot{\tau}_0, m_1\right) = \Cap\left(\Sigma_2(r_0), m_1\right) = \Cap\left(\Sigma'_2(r_0), m_1\right)$ (15) - (14)
- $\pi_{\texttt{type}}\left(\Cap\left(\Sigma_2(r_0), m_1\right)\right) = \pi_{\texttt{type}}\left(\Cap\left(\Sigma'_f(r_0), m_1\right)\right) = \pi_{\texttt{type}}\left(\Cap\left(\dot{\tau}_0, m_1\right)\right)$ (16) - (15)
- $\pi_{\texttt{lev}}\left(\Cap\left(\Sigma_2(r_0), m_1\right)\right) \sqsubseteq lev(\pi_{\texttt{type}}\left(\Cap\left(\Sigma_2(r_0), m_1\right)\right))$ (17) - *Syntax of Types*
- $lev(\pi_{\texttt{type}}\left(\Cap\left(\Sigma_2(r_0), m_1\right)\right)) \sqsubseteq lev(\pi_{\texttt{type}}\left(\Cap\left(\dot{\tau}_0, m_1\right)\right)) \sqsubseteq \hat{\sigma}'' \sqsubseteq \sigma$ (18) - (7) + (10) + (15)
- $\pi_{\texttt{lev}}\left(\Cap\left(\Sigma_2(r_0), m_1\right)\right) \sqsubseteq \sigma$ (19) - (17) + (18)
- $lev(\Sigma_2(r_0)) \sqsubseteq \sigma$ (20) - (7) + (12)
- $\pi_{\texttt{lev}}\left(\Cap\left(\Sigma_2(r_0), m_1\right)\right) \sqcup lev(\Sigma_2(r_0)) \sqsubseteq \sigma$ (21) - (19) + (20)
- $r_m = r'_m$ and $r_m \neq null \Rightarrow lev(\Sigma_2(r_m)) = lev(\Sigma'_2(r'_m)) \sqsubseteq \sigma$ (22) - (1) + (2) + (6) + (21) + Proto-Chain Indistinguishability (Lemma **??**)
- $lev(\Sigma_2(r_m)) = lev(\Sigma'_2(r'_m)) \sqsubseteq \sigma$ (23) - (1) + (2) + (22)
- $\Sigma_2(r_m) = \Sigma'_2(r_m)$ (24) - (6) + (22) + (23)
- $\Cap\left(\Sigma_2(r_0), m_1\right) = \Cap\left(\Sigma_2(r_m), m_1\right)$ (25) - (1) + Well-Lab. Proto-Chains (Lemma **??**)
- $\Cap\left(\Sigma'_2(r_0), m_1\right) = \Cap\left(\Sigma'_2(r_m), m_1\right)$ (26) - (2) + Well-Lab. Proto-Chains (Lemma **??**)
- $lev(\pi_{\texttt{type}}\left(\Cap\left(\Sigma_2(r_m), m_1\right)\right)) = lev(\pi_{\texttt{type}}\left(\Cap\left(\Sigma'_2(r_m), m_1\right)\right)) \sqsubseteq \sigma$ (27) - (18) + (25) + (26)
- $r_f = r'_f$ (28) - (1) + (2) + (6) + (8) + (22) + (23) + (27)
- $\Sigma_2(r_f) \curlyvee \Sigma'_2(r_f) \preceq \pi_{\texttt{type}}\left(\Cap\left(\dot{\tau}_0, P\right)\right)$ (29) - (1) - (3) + (6) + *Well-labeled Memory*
- $\Sigma_2(r_f) = \Sigma'_2(r_f) \preceq \pi_{\texttt{type}}\left(\Cap\left(\dot{\tau}_0, P\right)\right)$ (30) - (6) + (7) + (29)
- $\lfloor \Sigma_2(r_f) \rfloor \equiv \lfloor \Sigma'_2(r_f) \rfloor \equiv \lfloor \pi_{\texttt{type}}\left(\Cap\left(\dot{\tau}_0, P\right)\right) \rfloor \equiv \langle \dot{\tau}'_0 . \dot{\tau}'_1 \xrightarrow{\hat{\sigma}'} \dot{\tau}'_2 \rangle$ (31) - (30)
- $lev(\Sigma_2(r_f)) = lev(\Sigma'_2(r_f)) \sqsubseteq lev(\pi_{\texttt{type}}\left(\Cap\left(\dot{\tau}_0, P\right)\right)) \sqsubseteq \sigma$ (32) - (8) + (30)
- $\begin{cases} \mu_2(r_f)(@code) = \mu'_2(r_2)(@code) = \lambda^{(\hat{\Gamma}, \Sigma_2(r_f))} x.\, \left\{ \mathsf{var}^{\dot{\tau}_{y_1}, \cdots, \dot{\tau}_{y_n}} \; y_1, \cdots, y_n;\; \hat{e} \right\} \\ \mu_2(r_f)(@fscope) = \mu'_2(r_f)(@fscope) = \hat{r} = \hat{r}' \\ \hat{\Gamma}, \hat{r} \Vdash \mu_2 \sim_\sigma \mu'_2 \\ \hat{e} = \hat{e}' \end{cases}$ for some typing environment $\hat{\Gamma}$ and variables $x, y_1, \cdots, y_n$ (33) - (6)
- $\bar{\Gamma} \vdash \hat{e} : \dot{\tau}'_2, \hat{\sigma}$, where $\bar{\Gamma} = \hat{\Gamma}\left[\mathsf{this} \mapsto \dot{\tau}'_0, x \mapsto \dot{\tau}'_1, y_1 \mapsto \dot{\tau}_{y_1}, \cdots, y_n \mapsto \dot{\tau}_{y_n}\right]$ (34) - (33) + *Well-labeled Memory*
- $lev(\dot{\tau}'_0) \sqsubseteq \sigma \Rightarrow r_0 = r_0$ (35) - *tautology*
- $lev(\dot{\tau}'_1) \sqsubseteq \sigma \Rightarrow lev(\dot{\tau}_2) \sqsubseteq \sigma$ (36) - (3)
- $lev(\dot{\tau}'_1) \sqsubseteq \sigma \Rightarrow v_2 = v'_2$ (37) - (6) + (36)
- $\bar{\Gamma}, \hat{r} \Vdash \hat{\mu} \sim_\sigma \hat{\mu}'$ (38) - (1) + (2) + (6) + (32) + (35) + (37) + *Indist. Scope Alloc*
- $\hat{\mu}, \Sigma_2 \sim_\sigma \hat{\mu}', \Sigma'_2$ (39) - (1) + (2) + (6)
- $\mu_f, \Sigma_f \sim_\sigma \mu'_f, \Sigma'_f,\ \Gamma, r \Vdash \mu_f \sim_\sigma \mu'_f,\ lev(\dot{\tau}'_2) \sqsubseteq \sigma \Rightarrow v_f = v'_f$ **(40.1)** - **ih** + (1) + (2) + (34) + (39)
- $lev(\dot{\tau}) \sqsubseteq \sigma \Rightarrow lev(\dot{\tau}'_2) \sqsubseteq \sigma$ (40.2) - (3)
- $lev(\dot{\tau}) \sqsubseteq \sigma \Rightarrow v_f = v'_f$ **(40.3)** - (40.1) + (40.2)

Suppose $\sigma' \sqsubseteq \sigma$ (hyp.7). It follows that:

- $\hat{\sigma}' \not\sqsubseteq \sigma$ (41) - hyp.7 + (3)
- $\Sigma_2(r_f) \curlyvee \Sigma_2'(r_f') \preceq \pi_{\texttt{type}}\left(\Gamma^{\star}\left(\dot{\tau}_0, P\right)\right)$ (42) - (1) - (3) + (6) + *Well-labeled Memory*
- $\lfloor \Sigma_2(r_f) \rfloor \equiv \lfloor \Sigma_2'(r_f') \rfloor \equiv \lfloor \pi_{\texttt{type}}\left(\Gamma^{\star}\left(\dot{\tau}_0, P\right)\right) \rfloor \equiv \langle \dot{\tau}_0'.\dot{\tau}_1' \xrightarrow{\hat{\sigma}'} \dot{\tau}_2' \rangle$ (43) - (42)
- $\begin{cases} \mu_2(r_f)(@code) = \lambda^{(\hat{\Gamma}, \Sigma_2(r_f))} x.\left\{\mathsf{var}^{\dot{\tau}_{y_1}, \cdots, \dot{\tau}_{y_n}}\ y_1, \cdots, y_n;\ \hat{e}\right\} \\ \mu_2(r_f)(@fscope) = \hat{r} \\ \bar{\Gamma} \vdash \hat{e} : \dot{\tau}_2', \hat{\sigma}' \\ \bar{\Gamma} = \hat{\Gamma}\left[\mathsf{this} \mapsto \dot{\tau}_0', x \mapsto \dot{\tau}_1', y_1 \mapsto \dot{\tau}_{y_1}, \cdots, y_n \mapsto \dot{\tau}_{y_n}\right] \end{cases}$
  for some typing environment $\hat{\Gamma}$ and variables $x, y_1, \cdots, y_n$
  (44) - (1) + (3) + (43) + *Well-labeled Memory*
- $\begin{cases} \mu_2'(r_f')(@code) = \lambda^{(\hat{\Gamma}', \Sigma_2'(r_f'))} x'.\left\{\mathsf{var}^{\dot{\tau}'_{y_1'}, \cdots, \dot{\tau}'_{y_k'}}\ y_1', \cdots, y_k';\ \hat{e}'\right\} \\ \mu_2'(r_f')(@fscope) = \hat{r}' \\ \bar{\Gamma}' \vdash \hat{e}' : \dot{\tau}_2', \hat{\sigma}' \\ \bar{\Gamma}' = \hat{\Gamma}'\left[\mathsf{this} \mapsto \dot{\tau}_0', x' \mapsto \dot{\tau}_1', y_1' \mapsto \dot{\tau}'_{y_1'}, \cdots, y_n' \mapsto \dot{\tau}'_{y_n'}\right] \end{cases}$
  (45) - (2) + (3) + (43) + *Well-labeled Memory*
- $\hat{\mu} \restriction^{\Sigma_2, \sigma} = \mu_2 \restriction^{\Sigma_2, \sigma}$ and $(\hat{\mu}, r) \restriction^{\Gamma, \sigma} = (\mu_2, r) \restriction^{\Gamma, \sigma}$ (46) - (1)
- $\mu_f \restriction^{\Sigma_f, \sigma} = \hat{\mu} \restriction^{\Sigma_2, \sigma}$ and $(\mu_f, \hat{r}) \restriction^{\bar{\Gamma}, \sigma} = (\hat{\mu}, \hat{r}) \restriction^{\bar{\Gamma}, \sigma}$
  (47) - (1) + (44) + Confinement (Lemma **??**)
- $\hat{\mu}' \restriction^{\Sigma_2', \sigma} = \mu_2' \restriction^{\Sigma_2', \sigma}$ and $(\hat{\mu}', r) \restriction^{\Gamma, \sigma} = (\mu_2', r) \restriction^{\Gamma, \sigma}$ (48) - (2)
- $\mu_f' \restriction^{\Sigma_f', \sigma} = \hat{\mu}' \restriction^{\Sigma_2', \sigma}$ and $(\mu_f', \hat{r}') \restriction^{\bar{\Gamma}', \sigma} = (\hat{\mu}', \hat{r}') \restriction^{\bar{\Gamma}', \sigma}$
  (49) - (2) + (45) + Confinement (Lemma **??**)
- $\mu_2 \restriction^{\Sigma_2, \sigma} = \mu_2' \restriction^{\Sigma_2', \sigma}$ and $(\mu_2, r) \restriction^{\Gamma, \sigma} = (\mu_2', r) \restriction^{\Gamma, \sigma}$ (50) - (6)
- $\mu_f \restriction^{\Sigma_f, \sigma} = \mu_f' \restriction^{\Sigma_f', \sigma} \Leftrightarrow \mu_f, \Sigma_f \sim_\sigma \mu_f', \Sigma_f'$ **(51)** - (46)-(50)
- $\Gamma, r \Vdash \mu_f \sim_\sigma \mu_f'$ **(52)** - (46)-(50)
- $lev(\dot{\tau}) \not\sqsubseteq \sigma$ (53) - (3) + (41)
- $lev(\dot{\tau}) \sqsubseteq \sigma \Rightarrow v_f = v_f'$ **(54)** - (53)

[Property Deletion] $e = \mathsf{delete}\ e_0.p$ for some expression e_0 and property p (hyp.6). It follows:

- $r \vdash \langle \mu, \Sigma, e_0 \rangle \Downarrow \langle \mu_0, \Sigma_f, r_0 \rangle$ for some memory μ_0, labeling Σ_f, and reference r_0 such that: $\mu_f = \mu_0\left[r_0 \mapsto \mu_0(r_0)|_{dom(\mu_0(r_0) - p)}\right]$ and $v_f = \texttt{tt}$. (1) - hyp.2 + hyp.6
- $r \vdash \langle \mu', \Sigma', e_0 \rangle \Downarrow \langle \mu_0', \Sigma_f', r_0' \rangle$ for some memory μ_0', labeling Σ_f', and reference r_0' such that: $\mu_f' = \mu_0'\left[r_0' \mapsto \mu_0'(r_0')|_{dom(\mu_0'(r_0') - p)}\right]$ and $v_f' = \texttt{tt}$. (2) - hyp.3 + hyp.6
- $\Gamma \vdash e_0 : \dot{\tau}_0, \sigma_0, \Gamma^{\star}\left(\dot{\tau}_0, p\right) = (\sigma_0', \dot{\tau}_0'), lev(\dot{\tau}_0) \sqsubseteq \sigma_0'$, and $\dot{\tau} = \textsc{prim}^{\perp}$.
  (3) - hyp.1 + hyp.6
- $\mu_0, \Sigma_0 \sim_\sigma \mu_0', \Sigma_0', \Gamma, r \Vdash \mu_0 \sim_\sigma \mu_0', lev(\dot{\tau}_0) \sqsubseteq \sigma \Rightarrow r_0 = r_0'$
  (4) - **ih** + hyp.4 + hyp.5 + (1) + (2) + (3)
- $\Gamma, r \Vdash \mu_f \sim_\sigma \mu_f'$ **(5)** - (1) + (2) + (4)
- $v_f = v_f' = \texttt{tt}$ (6) - (1) + (2)
- $lev(\dot{\tau}) \sqsubseteq \sigma \Rightarrow v_f = v_f'$ **(7)** - (6)

We consider two cases: $lev(\dot{\tau}_0) \sqsubseteq \sigma$ and $lev(\dot{\tau}_0) \not\sqsubseteq \sigma$. Suppose $lev(\dot{\tau}_0) \sqsubseteq \sigma$ (hyp.7). It follows that:

- $r_0 = r_0'$ (8) - hyp.7 + (4)
- $\mu_f, \Sigma_f \sim_\sigma \mu_f', \Sigma_f'$ **(9)** - (1) + (2) + (4) + (8)

Suppose $lev(\dot{\tau}_0) \not\sqsubseteq \sigma$ (hyp.7). It follows that:

- $\pi_{\texttt{1ev}}\left(\Gamma^{\star}\left(\dot{\tau}_0, p\right)\right) \not\sqsubseteq \sigma$ (10) - hyp.7 + (4)

- $\Sigma_f(r_0) \curlyvee \Sigma'_f(r'_0) \preceq \dot{\tau}_0$ (11) - (1)-(3) + *Well-Lab. Mem.*
- $\lfloor \Sigma_f(r_0) \rfloor \equiv \lfloor \Sigma'_f(r'_0) \rfloor \equiv \lfloor \dot{\tau}_0 \rfloor$ (12) - (11)
- $\pi_{\texttt{lev}}\left(\Cap\left(\dot{\tau}_0, p\right)\right) = \pi_{\texttt{lev}}\left(\Cap\left(\Sigma_f(r_0), p\right)\right) = \pi_{\texttt{lev}}\left(\Cap\left(\Sigma'_f(r'_0), p\right)\right)$ (13) - (11)
- $\pi_{\texttt{lev}}\left(\Cap\left(\Sigma_f(r_0), p\right)\right) = \pi_{\texttt{lev}}\left(\Cap\left(\Sigma'_f(r'_0), p\right)\right) \not\sqsubseteq \sigma$ (14) - (10) + (13)
- $\mu_f, \Sigma_f \sim_\sigma \mu'_f, \Sigma'_f$ **(15)** - (1) + (2) + (4) + (14)

[Sequence] $e = e_0, e_1$ for two exprs. e_0 and e_1 (hyp.6). We conclude that:

- $r \vdash \langle \mu, \Sigma, e_0 \rangle \Downarrow \langle \mu_0, \Sigma_0, v_0 \rangle$ and $r \vdash \langle \mu_0, \Sigma_0, e_1 \rangle \Downarrow \langle \mu_f, \Sigma_f, v_f \rangle$ for some memory μ_0, labeling Σ_0, and value v_0. (1) - hyp.2 + hyp.6
- $r \vdash \langle \mu', \Sigma', e_0 \rangle \Downarrow \langle \mu'_0, \Sigma'_0, v'_0 \rangle$ and $r \vdash \langle \mu'_0, \Sigma'_0, e'_1 \rangle \Downarrow \langle \mu'_f, \Sigma'_f, v'_f \rangle$ for some memory μ'_0, labeling Σ'_0, and value v'_0. (2) - hyp.3 + hyp.6
- $\Gamma \vdash e_0 : \dot{\tau}_0, \sigma_0$ and $\Gamma \vdash e_1 : \dot{\tau}_1, \sigma_1$, where: $\dot{\tau} = \dot{\tau}_1$. (3) - hyp.1 + hyp.6
- $\mu_0, \Sigma_0 \sim_\sigma \mu'_0, \Sigma'_0, \Gamma, r \Vdash \mu_0 \sim_\sigma \mu'_0, lev(\dot{\tau}_0) \sqsubseteq \sigma \Rightarrow v_0 = v'_0$ (4) - **ih** + hyp.4 + hyp.5 + (1) + (2) + (3)
- $\mu_f, \Sigma_f \sim_\sigma \mu'_f, \Sigma'_f, \Gamma, r \Vdash \mu_f \sim_\sigma \mu'_f, lev(\dot{\tau}) \sqsubseteq \sigma \Rightarrow v_f = v'_f$ **(5)** - **ih** + (1) + (2) + (3) + (4)

[Conditional Expression] $e = e_0\ ?\ (e_1) : (e_2)$ for three exprs. e_0, e_1, and e_2 (hyp.6). We conclude that:

- $r \vdash \langle \mu, \Sigma, e_0 \rangle \Downarrow \langle \mu_0, \Sigma_0, v_0 \rangle$ and $r \vdash \langle \mu_0, \Sigma_0, e_i \rangle \Downarrow \langle \mu_f, \Sigma_f, v_f \rangle$ for some memory μ_0, labeling Σ_0, and value v_0 such that: $v_0 \notin V_F \Rightarrow i = 1$ and $v_0 \in V_F \Rightarrow i = 2$. (1) - hyp.2 + hyp.6
- $r \vdash \langle \mu', \Sigma', e_0 \rangle \Downarrow \langle \mu'_0, \Sigma'_0, v'_0 \rangle$ and $r \vdash \langle \mu'_0, \Sigma'_0, e_j \rangle \Downarrow \langle \mu'_f, \Sigma'_f, v'_f \rangle$ for some memory μ'_0, labeling Σ'_0, and value v'_0 such that: $v'_0 \notin V_F \Rightarrow j = 1$ and $v'_0 \in V_F \Rightarrow j = 2$. (2) - hyp.3 + hyp.6
- $\Gamma \vdash e_i : \dot{\tau}_i, \sigma_i$ for $i \in \{0, 1, 2\}$, $lev(\dot{\tau}_0) \sqsubseteq \sigma_1 \sqcap \sigma_2$, and $\dot{\tau} = (\dot{\tau}_1 \curlyvee \dot{\tau}_2)^{lev(\dot{\tau}_0)}$. (3) - hyp.1 + hyp.6
- $\mu_0, \Sigma_0 \sim_\sigma \mu'_0, \Sigma'_0, \Gamma, r \Vdash \mu_0 \sim_\sigma \mu'_0, lev(\dot{\tau}_0) \sqsubseteq \sigma \Rightarrow v_0 = v'_0$ (4) - **ih** + hyp.4 + hyp.5 + (1) + (2) + (3)

We consider two cases: $lev(\dot{\tau}_0) \sqsubseteq \sigma$ and $lev(\dot{\tau}_0) \not\sqsubseteq \sigma$. Suppose $lev(\dot{\tau}_0) \sqsubseteq \sigma$ (hyp.7). It follows that:

- $v_0 = v'_0$ (5) - hyp.7 + (4)
- $i = j$ (6) - (1) + (2) + (5)
- $\mu_f, \Sigma_f \sim_\sigma \mu'_f, \Sigma'_f, \Gamma, r \Vdash \mu_f \sim_\sigma \mu'_f, lev(\dot{\tau}_i) \sqsubseteq \sigma \Rightarrow v_f = v'_f$ **(7)** - **ih** + (1) + (2) + (3) + (4) + (6)
- $lev(\dot{\tau}) \sqsubseteq \sigma \Rightarrow lev(\dot{\tau}_i) \sqsubseteq \sigma$ (8) - (3)
- $lev(\dot{\tau}) \sqsubseteq \sigma \Rightarrow v_f = v'_f$ **(9)** - (7) + (8)

Suppose $lev(\dot{\tau}_0) \not\sqsubseteq \sigma$ (hyp.7)

- $\sigma_1 \sqcap \sigma_2 \not\sqsubseteq \sigma$ (10) - hyp.7 + (3)
- $\mu_f \restriction^{\Sigma_f, \sigma} = \mu_0 \restriction^{\Sigma_0, \sigma}$ and $(\mu_f, r) \restriction^{\Gamma, \sigma} = (\mu_0, r) \restriction^{\Gamma, \sigma}$ (11) - (1) + (10) + Confinement (Lemma **??**)
- $\mu'_f \restriction^{\Sigma'_f, \sigma} = \mu'_0 \restriction^{\Sigma'_0, \sigma}$ and $(\mu'_f, r) \restriction^{\Gamma, \sigma} = (\mu'_0, r) \restriction^{\Gamma, \sigma}$ (12) - (2) + (10) + Confinement (Lemma **??**)
- $\mu_0 \restriction^{\Sigma_0, \sigma} = \mu'_0 \restriction^{\Sigma'_0, \sigma}$ and $(\mu_0, r) \restriction^{\Gamma, \sigma} = (\mu'_0, r) \restriction^{\Gamma, \sigma}$ (13) - (4)
- $\mu_f \restriction^{\Sigma_f, \sigma} = \mu'_f \restriction^{\Sigma'_f, \sigma} \Leftrightarrow \mu_f, \Sigma_f \sim_\sigma \mu'_f, \Sigma'_f$ **(14)** - (11)-(13)

– $\Gamma, r \Vdash \mu_f \sim_\sigma \mu'_f$ **(15)** - (11)-(13)
– $lev(\dot{\tau}) \not\sqsubseteq \sigma$ (16) - hyp.7
– $lev(\dot{\tau}) \sqsubseteq \sigma \Rightarrow v_f = v'_f$ **(17)** - (16)

[Function Literal] $e = \mathsf{function}^{\Gamma,\dot{\tau},i}(x)\{\mathsf{var}^{\dot{\tau}_1,\cdots,\dot{\tau}_n}\ y_1, \cdots, y_n;\ \hat{e}\}$ (hyp.6). Let $f = \lambda^{(\Gamma,\dot{\tau})}x.\ \{\mathsf{var}^{\dot{\tau}_1,\cdots,\dot{\tau}_n}\ y_1, \cdots, y_n;\ \hat{e}\}$, we conclude that:

– $\mu_f = \mu\,[\hat{r} \mapsto [@fscope \mapsto r, @code \mapsto f]]$ and $\Sigma_f = \Sigma\,[\hat{r} \mapsto \dot{\tau}]$,
where: $\hat{r} = fresh(\mu, \Sigma, lev(\dot{\tau}))$. (1) - hyp.1 + hyp.2 + hyp.6
– $\mu'_f = \mu'\left[\hat{r'} \mapsto [@fscope \mapsto r, @code \mapsto f]\right]$ and $\Sigma'_f = \Sigma'\left[\hat{r'} \mapsto \dot{\tau}\right]$,
where: $\hat{r'} = fresh(\mu', \Sigma', lev(\dot{\tau}))$. (2) - hyp.1 + hyp.3 + hyp.6

We consider two cases: either $lev(\dot{\tau}) \sqsubseteq \sigma$ or $lev(\dot{\tau}) \not\sqsubseteq \sigma$. Suppose $lev(\dot{\tau}) \sqsubseteq \sigma$ (hyp.7):

– $\hat{r} = \hat{r}'$ (3) - hyp.4 + hyp.7 + (1) + (2)
– $\mu_f \restriction^{\Sigma_f,\sigma} = \mu \restriction^{\Sigma,\sigma} \cup\{(\hat{r}, f, (\mu, r) \restriction^{\Gamma,\sigma})\}$ (4) - hyp.7 + (1)
– $\mu'_f \restriction^{\Sigma'_f,\sigma} = \mu' \restriction^{\Sigma',\sigma} \cup\{(\hat{r}, f, (\mu', r) \restriction^{\Gamma,\sigma})\}$ (5) - hyp.7 + (1)
– $\mu \restriction^{\Sigma,\sigma} = \mu' \restriction^{\Sigma',\sigma}$ (6) - hyp.4
– $(\mu, r) \restriction^{\Gamma,\sigma} = (\mu', r) \restriction^{\Gamma,\sigma}$ (7) - hyp.5
– $\mu_f, \Sigma_f \sim_\sigma \mu'_f, \Sigma'_f$ **(8)** - (4)-(7)
– $\Gamma, r \Vdash \mu_f \sim_\sigma \mu'_f$ **(9)** - (1) + (2)
– $lev(\dot{\tau}) \sqsubseteq \sigma \Rightarrow v_f = v'_f$ **(10)** - (1) + (2) + (3)

Suppose $lev(\dot{\tau}) \sqsubseteq \sigma$ (hyp.7):

– $\mu_f \restriction^{\Sigma_f,\sigma} = \mu \restriction^{\Sigma,\sigma}$ (11) - hyp.7 + (1)
– $\mu'_f \restriction^{\Sigma'_f,\sigma} = \mu' \restriction^{\Sigma',\sigma}$ (12) - hyp.7 + (2)
– $\mu_f, \Sigma_f \sim_\sigma \mu'_f, \Sigma'_f$ **(13)** - hyp.4 + (11) + (12)
– $\Gamma, r \Vdash \mu_f \sim_\sigma \mu'_f$ **(14)** - (1) + (2)
– $lev(\dot{\tau}) \sqsubseteq \sigma \Rightarrow v_f = v'_f$ **(15)** - hyp.7

□

D Proofs for the Hybrid Type System

We say that two memories μ_0 and μ_1 are *equal up to TS variables*, written $\mu_0 \sim \mu_1$, if they coincide everywhere except in TS variables. Since TS variables are not labeled, they are never part of the low-projection of a memory (or scope-chain). Hence, if $\mu_0 \sim_\sigma \mu_1$, $\mu_0 \sim \mu'_0$, and $\mu_1 \sim \mu'_1$ for a security level σ, we conclude that: $\mu'_0 \sim_\sigma \mu'_1$.

Lemma 8 (Noninterference). *Given two memories μ and μ' well-labeled by Σ and Σ' resp., an expression e, a reference r, a typing environment Γ, and a reference r such that: $\Gamma \vdash e \rightsquigarrow e', e'' : T, L$ (hyp.1), $r \vdash \langle \mu, \Sigma, e' \rangle \Downarrow \langle \mu_f, \Sigma_f, v_f \rangle$ (hyp.2), $r \vdash \langle \mu', \Sigma', e' \rangle \Downarrow \langle \mu'_f, \Sigma'_f, v'_f \rangle$ (hyp.3), $\mu, \Sigma \sim_\sigma \mu', \Sigma'$ (hyp.4), and $\Gamma, r \Vdash \mu \sim_\sigma \mu'$(hyp.5); it holds that: (i) $\mu_f, \Sigma_f \sim_\sigma \mu'_f, \Sigma'_f$, (ii) $\Gamma, r \Vdash \mu_f \sim_\sigma \mu'_f$, and (iii) for all $(\dot{\tau}, \omega) \in T$, if $lev(\dot{\tau}) \sqsubseteq \sigma$ then: $\mu_f, r \vDash \omega \Leftrightarrow \mu'_f, r \vDash \omega$ and $\mu, r \vDash \omega \Rightarrow v_f = v'_f$.*

Proof. We proceed by induction on the derivation of *hyp.*2. For simplicity, we structure our analysis of the cases according to the last rule used in the typing of e.

[Val] $e = v$ for some value v (hyp.6). We conclude that:

- $e' = v$ (1) - hyp.1 + hyp.6
- $v_f = v'_f = v$ (2) - hyp.2 + hyp.3 + hyp.6
- $\mu_f = \mu$, $\mu'_f = \mu'$, $\Sigma_f = \Sigma$, $\Sigma'_f = \Sigma'$ (3) - hyp.2 + hyp.3 + hyp.6 + (1)
- $\mu_f, \Sigma_f \sim_\sigma \mu'_f, \Sigma'_f$ **(4)** - hyp.4 + (3)
- $\Gamma, r \Vdash \mu_f \sim_\sigma \mu'_f$ **(5)** - hyp.5 + (3)
- $T = \{(\mathsf{PRIM}^{\perp}, \mathtt{tt})\}$ (6) - hyp.1 + hyp.6
- $\mu_f, r \vDash \mathtt{tt}$ and $\mu'_f, r \vDash \mathtt{tt}$ (7) - *tautology*
- $\mu_f, r \vDash \mathtt{tt} \Rightarrow v_f = v'_f$ **(8)** - (2)
- $\mu_f, r \vDash \mathtt{tt} \Leftrightarrow \mu'_f, r \vDash \mathtt{tt}$ **(9)** - (8)

[This] $e = \mathtt{this}$ (hyp.6). We conclude that:

- $e' = \mathtt{this}$ (1) - hyp.1 + hyp.6
- $v_f = \mu(r)(@this)$ and $v'_f = \mu'(r)(@this)$ (2) - hyp.2 + hyp.3 + (1)
- $lev(\Gamma(\mathsf{this})) \sqsubseteq \sigma \Rightarrow v_f = v'_f$ (3) - hyp.5 + (2)
- $\mu_f = \mu$, $\mu'_f = \mu'$, $\Sigma_f = \Sigma$, and $\Sigma'_f = \Sigma'$. (4) - hyp.2 + hyp.3 + (1)
- $\mu_f, \Sigma_f \sim_\sigma \mu'_f, \Sigma'_f$ **(5)** - hyp.4 + (4)
- $\Gamma, r \Vdash \mu_f \sim_\sigma \mu'_f$ **(6)** - hyp.5 + (4)
- $T = \{(\Gamma(\mathsf{this}), \mathtt{tt})\}$ (7) - hyp.1 + hyp.6

In order to prove the third claim of the lemma, suppose that $lev(\Gamma(\mathsf{this})) \sqsubseteq \sigma$ (hyp.7). It follows that:

- $\mu_f, r \vDash \mathtt{tt} \Leftrightarrow \mu'_f, r \vDash \mathtt{tt}$ **(8)** - *tautology*
- $v_f = v'_f$ (9) - hyp.7 + (3)
- $\mu_f, r \vDash \mathtt{tt} \Rightarrow v_f = v'_f$ **(10)** - *tautology*

[Variable] $e = x^i$, for some variable x and index i (hyp.6). We conclude that:

- $e' = \hat{x}_i = x$ (1) - hyp.2 + hyp.6
- $\mu = \mu_f$, $\Sigma = \Sigma_f$, and $v_f = \mu(r_x)(x)$, where: $\langle \mu, r, x \rangle \ \mathcal{R}_{Scope}\ r_x$ for some reference r_x. (2) - hyp.2 + (1)
- $\mu' = \mu'_f$, $\Sigma' = \Sigma'_f$, $v'_f = \mu'(r'_x)(x)$, where: $\langle \mu', r, x \rangle \ \mathcal{R}_{Scope}\ r'_x$ for some reference r'_x. (3) - hyp.3 + (1)
- $lev(\Gamma(x)) \sqsubseteq \sigma \Rightarrow v_f = v'_f$ (4) - hyp.5 + (2) + (3)
- $\mu_f, \Sigma_f \sim_\sigma \mu'_f, \Sigma'_f$ **(5)** - hyp.4 + (2) + (3)
- $\Gamma, r \Vdash \mu_f \sim_\sigma \mu'_f$ **(6)** - hyp.5 + (2) + (3)
- $T = \{(\Gamma(x), \mathtt{tt})\}$ (7) - hyp.1 + hyp.6

Suppose that $lev(\Gamma(x)) \sqsubseteq \sigma$ (hyp.7). It follows that:

- $\mu_f, r \vDash \mathtt{tt} \Leftrightarrow \mu'_f, r \vDash \mathtt{tt}$ **(8)** - *tautology*
- $v_f = v'_f$ (9) - hyp.7 + (4)
- $\mu_f, r \vDash \mathtt{tt} \Rightarrow v_f = v'_f$ **(10)** - (9)

[Binary Operation] $e = e_0\ \mathsf{op}^j\ e_1$ for two exprs. e_0 and e_1 (hyp.6). We conclude that:

- $\Gamma \vdash e_i \rightsquigarrow e'_i / e''_i : T_i, L_i$, where: $i \in \{0, 1\}$, $e' = e'_0, e'_1, \hat{x}_j = e''_0\ \mathsf{op}\ e''_1$, and $T = T_0 \oplus_\gamma T_1$. (1) - hyp.1 + hyp.6

- $r \vdash \langle \mu, \Sigma, e_0' \rangle \Downarrow \langle \mu_0, \Sigma_0, v_0 \rangle$ and $r \vdash \langle \mu_0, \Sigma_0, e_1' \rangle \Downarrow \langle \mu_1, \Sigma_f, v_1 \rangle$ for some memories μ_0 and μ_1, labeling Σ_0, and two values v_0 and v_1 such that: $\mu_f \sim \mu_1$ and $v_f = v_0$ op v_1 (2) - hyp.2 + (1) + *Transparency*
- $r \vdash \langle \mu', \Sigma', e_0' \rangle \Downarrow \langle \mu_0', \Sigma_0', v_0' \rangle$ and $r \vdash \langle \mu_0', \Sigma_0', e_1' \rangle \Downarrow \langle \mu_1', \Sigma_f', v_1' \rangle$ for some memories μ_0' and μ_1', labeling Σ_0', and two values v_0' and v_1' such that: $\mu_f' \sim \mu_1'$ and $v_f' = v_0'$ op v_1' (3) - hyp.3 + (1) + *Transparency*
- $\mu_0, \Sigma_0 \sim_\sigma \mu_0', \Sigma_0'$, $\Gamma, r \Vdash \mu_0 \sim_\sigma \mu_0'$, and $\forall_{(\dot\tau_0, \omega_0) \in T_0} lev(\dot\tau_0) \sqsubseteq \sigma \Rightarrow (\mu_0, r \vDash \omega_0 \Leftrightarrow \mu_0', r \vDash \omega_0) \wedge (\mu_0, r \vDash \omega_0 \Rightarrow v_0 = v_0')$. (4) - **ih** + hyp.4 + hyp.5 + (1) + (2) + (3)
- $\mu_1, \Sigma_1 \sim_\sigma \mu_1', \Sigma_1'$, $\Gamma, r \Vdash \mu_1 \sim_\sigma \mu_1'$, and $\forall_{(\dot\tau_1, \omega_1) \in T_1} lev(\dot\tau_1) \sqsubseteq \sigma \Rightarrow (\mu_1, r \vDash \omega_1 \Leftrightarrow \mu_1', r \vDash \omega_1) \wedge (\mu_1, r \vDash \omega_1 \Rightarrow v_1 = v_1')$. (5) - **ih** + (1) + (2) + (3) + (4)
- $\mu_f, \Sigma_f \sim_\sigma \mu_f', \Sigma_f'$ and $\Gamma, r \Vdash \mu_f \sim_\sigma \mu_f'$
  (6) - (1) + (2) + (3) + (5) + *Low-Equality Preservation for Internal Updates*

Suppose that $(\dot\tau, \omega) \in T$ (hyp.7), $lev(\dot\tau) \sqsubseteq \sigma$ (hyp.8), and $\mu_f, r \vDash \omega$ (hyp.9). It follows that there are $(\dot\tau_0, \omega_0) \in T_0$ and $(\dot\tau_1, \omega_1) \in T_1$ such that:

- $\forall_{\hat\mu, \hat r}\ \hat\mu, \hat r \vDash \omega \Leftrightarrow (\hat\mu, \hat r \vDash \omega_0 \wedge \omega_1) \wedge (\dot\tau = \dot\tau_0 \curlyvee \dot\tau_1)$ (7) - hyp.7 + (1)
- $\mu_f, r \vDash \omega \Leftrightarrow (\mu_f, r \vDash \omega_0 \wedge \omega_1) \wedge (\dot\tau = \dot\tau_0 \curlyvee \dot\tau_1)$ (8) - (7)
- $\mu_f, r \vDash \omega_0$, $\mu_f, r \vDash \omega_1$, and $\dot\tau = \dot\tau_0 \curlyvee \dot\tau_1$. (9) - hyp.9 + (8)
- $\mu_f, r \vDash \omega_0 \Leftrightarrow \mu_0, r \vDash \omega_0$ and $\mu_f, r \vDash \omega_1 \Leftrightarrow \mu_1, r \vDash \omega_1$
  (10) - (1) + (2) + *Invariance of Dynamic Assertions*
- $\mu_f', r \vDash \omega_0 \Leftrightarrow \mu_0', r \vDash \omega_0$ and $\mu_f', r \vDash \omega_1 \Leftrightarrow \mu_1', r \vDash \omega_1$
  (11) - (1) + (3) + *Invariance of Dynamic Assertions*
- $\mu_0, r \vDash \omega_0$ and $\mu_1, r \vDash \omega_1$ (12) - (9) + (10)
- $lev(\dot\tau_0) \sqsubseteq \sigma$ and $lev(\dot\tau_1) \sqsubseteq \sigma$ (13) - hyp.8 + (9)
- $\mu_0', r \vDash \omega_0$ and $v_0 = v_0'$ (14) - (4) + (12) + (13)
- $\mu_1', r \vDash \omega_1$ and $v_1 = v_1'$ (15) - (5) + (12) + (13)
- $\mu_f', r \vDash \omega_0$ and $\mu_f', r \vDash \omega_1$ (16) - (11) + (14) + (15)
- $\mu_f', r \vDash \omega_0 \wedge \omega_1$ **(17)** - (16)
- $v_f = v_f'$ **(18)** - (2) + (3) + (14) + (15)

[Object Literal] $e = \{\}^{\dot\tau, i}$ for an index i and a type $\dot\tau$ (hyp.6). We conclude that:

- $T = \{(\tau, \mathtt{tt})\}$ and $e' = \hat x_i = \{\}^\tau$ (1) - hyp.1 + hyp.6
- $\hat r = fresh(\mu, \Sigma, lev(\dot\tau))$, $\hat\mu = \mu\,[\hat r \mapsto [\_proto\_ \mapsto null]]$, $\mu_f \sim \hat\mu$, $\Sigma_f = \Sigma\,[\hat r \mapsto \dot\tau]$, $v_f = \hat r$. (1) - hyp.2 + hyp.6
- $\hat r' = fresh(\mu', \Sigma', lev(\dot\tau)')$, $\hat\mu' = \mu'\,[\hat r' \mapsto [\_proto\_ \mapsto null]]$, $\mu_f' \sim \hat\mu'$, $\Sigma_f' = \Sigma'\,[\hat r' \mapsto \dot\tau]$, $v_f' = \hat r'$. (1) - hyp.3 + hyp.6
- $\Gamma, r \Vdash \mu_f \sim_\sigma \mu_f'$ **(4)** - hyp.5 + (2) + (3)

Suppose that $(\dot\tau', \omega) \in T$ (hyp.7), it follows that $\dot\tau' = \dot\tau$ and $\omega = \mathtt{tt}$. We consider two cases: either $lev(\dot\tau) \sqsubseteq \sigma$ or $lev(\dot\tau) \not\sqsubseteq \sigma$. Suppose $lev(\dot\tau) \sqsubseteq \sigma$ (hyp.8):

- $\hat r = \hat r'$ (5) - hyp.4 + hyp.8 + (2) + (3)
- $\mu_f \restriction^{\Sigma_f, \sigma} = \mu \restriction^{\Sigma, \sigma} \cup \{(\hat r, \dot\tau)\} \cup \{(\hat r, \_proto\_, null), (\hat r, \_proto\_)\}$ (6) - hyp.8 + (2)
- $\mu_f' \restriction^{\Sigma_f', \sigma} = \mu' \restriction^{\Sigma', \sigma} \cup \{(\hat r, \dot\tau)\} \cup \{(\hat r, \_proto\_, null), (\hat r, \_proto\_)\}$
  (7) - hyp.8 + (3) + (5)
- $\mu_f, \Sigma_f \sim_\sigma \mu_f', \Sigma_f'$ **(8)** - hyp.4 + (6) + (7)
- $\mu_f, r \vDash \mathtt{tt} \Leftrightarrow \mu_f', r \vDash \mathtt{tt}$ **(9)** - *tautology*
- $v_f = v_f'$ (10) - (2) + (3) + (5)
- $\mu_f, r \vDash \mathtt{tt} \Rightarrow v_f = v_f'$ **(11)** - (10)

Suppose $lev(\dot\tau) \not\sqsubseteq \sigma$ (hyp.8):

- $\mu_f \restriction^{\Sigma_f,\sigma} = \mu \restriction^{\Sigma,\sigma}$ (12) - hyp.8 + (2)
- $\mu'_f \restriction^{\Sigma'_f,\sigma} = \mu' \restriction^{\Sigma',\sigma}$ (13) - hyp.8 + (3)
- $\mu_f, \Sigma_f \sim_\sigma \mu'_f, \Sigma'_f$ **(13)** - hyp.4 + (12) + (13)

[Variable Assignment] $e = x = e_0$ for some variable e and expression e_0 (hyp.6). We conclude that:

- $\Gamma \vdash e_0 \rightsquigarrow e'_0/e''_0 : T, L_0, \mathit{When}^{?}_{\preceq}(T, \{(\Gamma(x), \mathtt{tt})\}) = \omega$, and $e' = e'_0, \mathit{wrap}(\omega, x = e''_0)$. (1) - hyp.1 + hyp.6
- $r \vdash \langle \mu, \Sigma, e'_0\rangle \Downarrow \langle \mu_0, \Sigma_f, v_f\rangle$, $\langle \mu_0, r, x\rangle \mathcal{R}_{Scope}\ r_x$, $\mu_f = \mu_0\,[r_x \mapsto \mu_0(r_x)\,[x \mapsto v_f]]$, and $\mu_0, r \vDash \omega$, for some memory μ_0 and reference r_x. (2) - hyp.2 + hyp.6
- $r \vdash \langle \mu', \Sigma', e'_0\rangle \Downarrow \langle \mu'_0, \Sigma'_f, v'_f\rangle, \langle \mu'_0, r, x\rangle \mathcal{R}_{Scope}\ r'_x, \mu'_0 = \mu'_0\,[r'_x \mapsto \mu'_0(r'_x)\,[x \mapsto v'_f]]$, and $\mu'_f, r \vDash \omega$, for some memory μ'_0 and reference r'_x. (3) - hyp.3 + hyp.6
- $\mu_0, \Sigma_0 \sim_\sigma \mu'_0, \Sigma'_0$, $\Gamma, r \Vdash \mu_0 \sim_\sigma \mu'_0$, and $\forall_{(\dot\tau_0,\omega_0)\in T} \mathit{lev}(\dot\tau_0) \sqsubseteq \sigma \Rightarrow (\mu_0, r \vDash \omega_0 \Leftrightarrow \mu'_0, r \vDash \omega_0) \wedge (\mu_0, r \vDash \omega_0 \Rightarrow v_f = v'_f)$. **(5)** - **ih** + hyp.4 + hyp.5 + (1) + (2) + (3)
- $\mu_f, \Sigma_f \sim_\sigma \mu'_f, \Sigma'_f$ **(6)** - (2) + (3) + (5)
- $\forall_{\hat\mu,\hat r}\ \hat\mu, \hat r \vDash \omega \Leftrightarrow \exists_{(\dot\tau_0,\omega_0)\in T}\ \dot\tau_0 \preceq \Gamma(x) \wedge \hat\mu, \hat r \vDash \omega_0$ (7) - (1)
- $\mu_0, r \vDash \omega \Leftrightarrow \exists_{(\dot\tau_0,\omega_0)\in T}\ \dot\tau_0 \preceq \Gamma(x) \wedge \mu_0, r \vDash \omega_0$ (8) - (7)
- $\exists_{(\dot\tau_0,\omega_0)\in T}\ \dot\tau_0 \preceq \Gamma(x) \wedge \mu_0, r \vDash \omega_0$ (9) - (2) + (8)
- $\mathit{lev}(\Gamma(x)) \sqsubseteq \sigma \Rightarrow \exists_{(\dot\tau_0,\omega_0)\in T}\ \dot\tau_0 \preceq \sigma \wedge \mu_0, r \vDash \omega_0$ (10) - (9)
- $\mathit{lev}(\Gamma(x)) \sqsubseteq \sigma \Rightarrow v_f = v'_f$ (11) - (5) + (10)
- $\Gamma, r \Vdash \mu_f \sim_\sigma \mu'_f$ **(8)** - (2) + (3) + (5) + (11) + Lemma **??**

[Property Look-up] $e = e_0[e_1, P]^j$ for two expressions e_0 and e_1 (hyp.6). It follows:

- $\Gamma \vdash e_i \rightsquigarrow e'_i/e''_i : T_i, L_i,\ T = \left(\pi_{\mathtt{type}}\left(\restriction^{?}(T_0, P, e''_1)\right)\right)^{\mathit{lev}(T_0)\oplus_\sqcup \mathit{lev}(T_1)}$, and $e' = e'_0, e'_1, \hat x_j = e''_0[e''_1]$ (1) - hyp.1 + hyp.6
- $r \vdash \langle \mu, \Sigma, e'_0\rangle \Downarrow \langle \mu_0, \Sigma_0, r_0\rangle$ and $r \vdash \langle \mu_0, \Sigma_0, e'_1\rangle \Downarrow \langle \mu_1, \Sigma_f, m_1\rangle$ for two mems. μ_0 and μ_1, labeling Σ_0, refs. r_0 and $\hat r$, and string m_1 such that: $\langle \mu_1, r_0, m_1\rangle \mathcal{R}_{Proto}\ \hat r$, $\hat r \neq \mathit{null} \Rightarrow v_f = \mu_f(\hat r)(m_1)$, $\hat r = \mathit{null} \Rightarrow v = \mathit{undefined}$, and $\mu_f \sim \mu_1$. (2) - hyp.2 + hyp.6
- $r \vdash \langle \mu', \Sigma', e'_0\rangle \Downarrow \langle \mu'_0, \Sigma'_0, r'_0\rangle$ and $r \vdash \langle \mu'_0, \Sigma'_0, e'_1\rangle \Downarrow \langle \mu'_1, \Sigma'_f, m'_1\rangle$ for two memories μ'_0 and μ'_1, labeling Σ'_0, references r'_0 and $\hat r'$, and string m'_1 such that: $\langle \mu'_f, r'_0, m'_1\rangle \mathcal{R}_{Proto}\ \hat r'$, $\hat r' \neq \mathit{null} \Rightarrow v'_f = \mu'_f(\hat r')(m'_1)$, $\hat r = \mathit{null} \Rightarrow v_f = \mathit{undefined}$, and $\mu'_f \sim \mu'_1$. (3) - hyp.3 + hyp.6
- $\mu_0, \Sigma_0 \sim_\sigma \mu'_0, \Sigma'_0$, $\Gamma, r \Vdash \mu_0 \sim_\sigma \mu'_0$, $\forall_{(\dot\tau_0,\omega_0)\in T_0} \mathit{lev}(\dot\tau_0) \sqsubseteq \sigma \Rightarrow (\mu_0, r \vDash \omega_0 \Leftrightarrow \mu'_0, r \vDash \omega_0) \wedge (\mu_0, r \vDash \omega_0 \Rightarrow r_0 = r'_0)$ (4) - **ih** + hyp.4 + hyp.5 + (1) + (2) + (3)
- $\mu_1, \Sigma_1 \sim_\sigma \mu'_1, \Sigma'_1$, $\Gamma, r \Vdash \mu_1 \sim_\sigma \mu'_1$, $\forall_{(\dot\tau_1,\omega_1)\in T_1} \mathit{lev}(\dot\tau_1) \sqsubseteq \sigma \Rightarrow (\mu_1, r \vDash \omega_1 \Leftrightarrow \mu'_1, r \vDash \omega_1) \wedge (\mu_1, r \vDash \omega_1 \Rightarrow m_1 = m'_1)$ **(5)** - **ih** + (1) + (2) + (3) + (4)
- $\mu_f, \Sigma_f \sim_\sigma \mu'_f, \Sigma'_f$ and $\Gamma, r \Vdash \mu_f \sim_\sigma \mu'_f$
  (6) - (1) + (2) + (3) + (5) + *Low-Equality Preservation for Internal Updates*

It remains to prove that $\forall_{(\dot\tau,\omega)\in T} \mathit{lev}(\dot\tau) \sqsubseteq \sigma \Rightarrow (\mu_f, r \vDash \omega \Leftrightarrow \mu'_f, r \vDash \omega) \wedge (\mu_f, r \vDash \omega \Rightarrow v_f = v'_f)$. Suppose that $(\dot\tau, \omega) \in T$ (hyp.7.1), $\mathit{lev}(\dot\tau) \sqsubseteq \sigma$ (hyp.7.2), and $\mu_f, r \vDash \omega$ (hyp.7.3). It follows that there is $(\dot\tau_0, \omega_0), (\dot\tau'_0, \omega'_0) \in T_0$, $(\dot\tau_1, \omega_1) \in T_1$ and $p \in \mathcal{S}tr$ such that:

- $\dot\tau = \left(\pi_{\mathtt{type}}\left(\restriction^{\cdot}(\dot\tau'_0, p)\right)\right)^{\mathit{lev}(\dot\tau_0)\sqcup \mathit{lev}(\dot\tau_1)} \wedge \omega = \omega_0 \wedge \omega_1 \wedge \omega'_0 \wedge \omega_p$, where $p \in \mathit{dom}(\dot\tau'_0) \Rightarrow \omega_p = e''_1 \in \{p\}$ and $p \notin \mathit{dom}(\dot\tau'_0) \Rightarrow \omega_p = \neg(e''_0 \in \mathit{dom}(\dot\tau'_0) \cap P)$ (7) - hyp.7.1
- $\mu_f, r \vDash \omega_0$, $\mu_f, r \vDash \omega_1$, $\mu_f, r \vDash \omega'_0$, and $\mu_f, r \vDash \omega_p$ (8) - hyp.7.3 + (7)
- $\dot\tau'_0 = \dot\tau_0$ and $\omega'_0 = \omega_0$ (9) - (1) + (2) + (7) + *Incompatible Assertions*
- $\dot\tau = \left(\pi_{\mathtt{type}}\left(\restriction^{\cdot}(\dot\tau_0, p)\right)\right)^{\mathit{lev}(\dot\tau_0)\sqcup \mathit{lev}(\dot\tau_1)} \wedge \omega = \omega_0 \wedge \omega_1 \wedge \omega_p$, where $p \in \mathit{dom}(\dot\tau_0) \Rightarrow \omega_p = \hat x_j \in \{p\}$ and $p \notin \mathit{dom}(\dot\tau_0) \Rightarrow \omega_p = \neg(\hat x_j \in \mathit{dom}(\dot\tau_0) \cap P)$ (10) - (7) + (9)

- $lev(\dot{\tau}_0) \sqcup lev(\dot{\tau}_1) \sqcup lev(\pi_{\texttt{type}}\,(\Rsh\,(\dot{\tau}_0, p))) \sqsubseteq \sigma$ (11) - hyp.7.2 + (3)
- $\mu_f, r \vDash \omega_0 \Leftrightarrow \mu_0, r \vDash \omega_0$, $\mu_f, r \vDash \omega_1 \Leftrightarrow \mu_1, r \vDash \omega_1$, and $\mu_f, r \vDash \omega_p \Leftrightarrow \mu_1, r \vDash \omega_p$ (12) - (1) + (2) + *Invariance of Dynamic Assertions*
- $\mu'_f, r \vDash \omega_0 \Leftrightarrow \mu'_0, r \vDash \omega_0$, $\mu'_f, r \vDash \omega_1 \Leftrightarrow \mu'_1, r \vDash \omega_1$, and $\mu'_f, r \vDash \omega_p \Leftrightarrow \mu'_1, r \vDash \omega_p$ (13) - (1) + (3) + *Invariance of Dynamic Assertions*
- $\mu'_0, r \vDash \omega_0$ and $r_0 = r'_0$ (14) - (4) + (8) + (11) + (12)
- $\mu'_1, r \vDash \omega_1$ and $m_1 = m'_1$ (15) - (5) + (8) + (11) + (12)
- $\mu'_f, r \vDash \omega_0$ and $\mu'_f, r \vDash \omega_1$ (16) - (13)-(15)
- $\mu_1, r \vDash \omega_p \Rightarrow (p \in dom(\dot{\tau}_0) \wedge m_1 = p) \vee (p \notin dom(\dot{\tau}_0) \wedge m_1 \notin dom(\dot{\tau}_0))$ (17) - (2) + (7) + *Invariance of Bookkeeping Expressions*
- $(p \in dom(\dot{\tau}_0) \wedge m_1 = p) \vee (p \notin dom(\dot{\tau}_0) \wedge m_1 \notin dom(\dot{\tau}_0))$ (18) - (8) + (12) + (17)
- $\Rsh\,(\dot{\tau}_0, m_1) = \Rsh\,(\dot{\tau}_0, p)$ (19) - (7) + (18)
- $\mu_0, r \vDash \omega_0 \Rightarrow \Sigma_0(r_0) \preceq \dot{\tau}_0$ (20) - hyp.7.1 + (1) + (2) + *Well-labeled Memory*
- $\mu'_0, r \vDash \omega_0 \Rightarrow \Sigma'_0(r'_0) \preceq \dot{\tau}_0$ (21) - hyp.7.1 + (1) + (3) + *Well-labeled Memory*
- $\mu_0, r \vDash \omega_0 \Rightarrow \Sigma_0(r_0) \curlyvee \Sigma'_0(r'_0) \preceq \dot{\tau}_0$ (22) - (4) + (8) + (20) + (21)
- $\mu_f, r \vDash \omega_0 \Rightarrow \Sigma_f(r_0) \curlyvee \Sigma'_f(r'_0) \preceq \dot{\tau}_0$ (23) - (22) + *Invariance of Dynamic Assertions*
- $\Sigma_f(r_0) = \Sigma'_f(r_0) \preceq \dot{\tau}_0$ (24) - (4) + (8) + (11) + (14) + (23)
- $\lfloor \Sigma_f(r_0) \rfloor = \lfloor \Sigma'_f(r'_0) \rfloor = \lfloor \dot{\tau}_0 \rfloor$ (25) - (24)
- $\Rsh\,(\dot{\tau}_0, p) = \Rsh\,(\dot{\tau}_0, m_1) = \Rsh\,(\Sigma_f(r_0), m_1) = \Rsh\,(\Sigma'_f(r'_0), m'_1)$ (26) - (19) + (24)
- $lev(\pi_{\texttt{type}}\,(\Rsh\,(\Sigma_f(r_0), m_1))) = lev(\pi_{\texttt{type}}\,(\Rsh\,(\Sigma'_f(r'_0), m'_1))) \sqsubseteq \sigma$ (27) - (11) + (26)
- $lev(\pi_{\texttt{type}}\,(\Rsh\,(\Sigma_f(r_0), m_1))) \sqcup lev(\Sigma_f(r_0)) \sqsubseteq \sigma$ (28) - (11) + (25)
- $\hat{r} = \hat{r}'$ and $\hat{r} \neq null \Rightarrow lev(\Sigma_f(\hat{r})) = lev(\Sigma'_f(\hat{r}')) \sqsubseteq \sigma$ (29) - (2) + (3) + (6) + (28) + Proto-Chain Indistinguishability (Lemma **??**)

We consider two cases: $\hat{r} \neq null$ or $\hat{r} = null$. Suppose $\hat{r} \neq null$ (hyp.8):

- $\hat{r}' \neq null$ and $lev(\Sigma_f(\hat{r})) = lev(\Sigma'_f(\hat{r}')) \sqsubseteq \sigma$ (30) - hyp.8 + (29)
- $\Rsh\,(\Sigma_f(r_0), m_1) = \Rsh\,(\Sigma_f(\hat{r}), m_1)$ (31) - (1) + Well-Lab. Proto-Chains (Lemma **??**)
- $\Rsh\,(\Sigma'_f(r'_0), m_1) = \Rsh\,(\Sigma'_f(\hat{r}'), m_1)$ (32) - (2) + Well-Lab. Proto-Chains (Lemma **??**)
- $lev(\pi_{\texttt{type}}\,(\Rsh\,(\Sigma_f(\hat{r}), m_1))) = lev(\pi_{\texttt{type}}\,(\Rsh\,(\Sigma'_f(\hat{r}), m_1))) \sqsubseteq \sigma$ (33) - (27) + (31) + (32)
- $v_f = v'_f$ **(34)** - hyp.8 + (2) + (3) + (6) + (29) + (30) + (33)

Suppose $\hat{r} = null$ (hyp.8):

- $\hat{r}' = null$ (35) - hyp.8 + (29)
- $v_f = v'_f = undefined$ **(36)** - hyp.8 + (2) + (3) + (35)

[In Expression] $e = e_0\ \mathsf{in}_j^P\ e_1$ for two expressions e_0 and e_1, a set of properties P, and an index j (hyp.6). It follows that:

- $\Gamma \vdash e_i \rightsquigarrow e'_i/e''_i : T_i, L_i$, $T = \{(\mathsf{PRIM}^{\perp}, \texttt{tt})\}^{\pi_{\texttt{lev}}\left(\Rsh^?(T_0, P, e''_0)\right) \oplus_{\sqcup} lev(T_0) \oplus_{\sqcup} lev(T_1)}$, and $e' = e'_0, e'_1, \hat{x}_j = e''_0\ \mathsf{in}\ e''_1$ for $i \in \{0, 1\}$ (1) - hyp.1 + hyp.6
- $r \vdash \langle \mu, \Sigma, e'_0 \rangle \Downarrow \langle \mu_0, \Sigma_0, m_0 \rangle$ and $r \vdash \langle \mu_0, \Sigma_0, e'_1 \rangle \Downarrow \langle \mu_1, \Sigma_f, r_1 \rangle$ for two mems. μ_0 and μ_1, labeling Σ_0, refs. r_1 and $\hat{r}$, and string m_0 such that: $\langle \mu_1, r_0, m_1 \rangle\ \mathcal{R}_{Proto}\ \hat{r}$, $\hat{r} \neq null \Rightarrow v_f = \texttt{tt}$, $\hat{r} = null \Rightarrow v_f = \texttt{ff}$, and $\mu_f \sim \mu_1$. (2) - hyp.2 + hyp.6
- $r \vdash \langle \mu', \Sigma', e'_0 \rangle \Downarrow \langle \mu'_0, \Sigma'_0, m'_0 \rangle$ and $r \vdash \langle \mu'_0, \Sigma'_0, e'_1 \rangle \Downarrow \langle \mu'_1, \Sigma'_f, r'_1 \rangle$ for two mems. μ'_0 and μ'_1, labeling Σ'_0, refs. r'_1 and $\hat{r}'$, and string m'_0 such that: $\langle \mu'_1, r'_0, m'_1 \rangle\ \mathcal{R}_{Proto}\ \hat{r}'$, $\hat{r}' \neq null \Rightarrow v'_f = \texttt{tt}$, $\hat{r}' = null \Rightarrow v'_f = \texttt{ff}$, and $\mu'_f \sim \mu'_1$. (3) - hyp.3 + hyp.6

- $\mu_0, \Sigma_0 \sim_\sigma \mu'_0, \Sigma'_0$, $\Gamma, r \Vdash \mu_0 \sim_\sigma \mu'_0$, $\forall_{(\dot\tau_0,\omega_0)\in T_0} lev(\dot\tau_0) \sqsubseteq \sigma \Rightarrow (\mu_0, r \vDash \omega_0 \Leftrightarrow \mu'_0, r \vDash \omega_0) \wedge (\mu_0, r \vDash \omega_0 \Rightarrow m_0 = m'_0)$ (4) - **ih** + hyp.4 + hyp.5 + (1) + (2) + (3)
- $\mu_1, \Sigma_1 \sim_\sigma \mu'_1, \Sigma'_1$, $\Gamma, r \Vdash \mu_1 \sim_\sigma \mu'_1$, $\forall_{(\dot\tau_1,\omega_1)\in T_1} lev(\dot\tau_1) \sqsubseteq \sigma \Rightarrow (\mu_1, r \vDash \omega_1 \Leftrightarrow \mu'_1, r \vDash \omega_1) \wedge (\mu_1, r \vDash \omega_1 \Rightarrow r_1 = r'_1)$ (5) - **ih** + (1) + (2) + (3) + (4)
- $\mu_f, \Sigma_f \sim_\sigma \mu'_f, \Sigma'_f$ and $\Gamma, r \Vdash \mu_f \sim_\sigma \mu'_f$
  (6) - (1) + (2) + (3) + (5) + *Low-Equality Preservation for Internal Updates*

It remains to prove that $\forall_{(\dot\tau,\omega)\in T} lev(\dot\tau) \sqsubseteq \sigma \Rightarrow (\mu_f, r \vDash \omega \Leftrightarrow \mu'_f, r \vDash \omega) \wedge (\mu_f, r \vDash \omega \Rightarrow v_f = v'_f)$. Suppose that $(\dot\tau, \omega) \in T$ (hyp.7.1), $lev(\dot\tau) \sqsubseteq \sigma$ (hyp.7.2), and $\mu_f, r \vDash \omega$ (hyp.7.3). It follows that there is $(\dot\tau_0, \omega_0) \in T_0$, $(\dot\tau_1, \omega_1), (\dot\tau'_1, \omega'_1) \in T_1$, and $p \in \mathcal{S}tr$:

- $\dot\tau = \mathsf{PRIM}^{\pi_{\mathtt{lev}}}\big(\Rsh\,(\dot\tau'_1, p)\big)_{\sqcup lev(\dot\tau_0)\sqcup lev(\dot\tau_1)}$ and $\omega = \omega_0 \wedge \omega_1 \wedge \omega'_1 \wedge \omega_p$, where $p \in dom(\dot\tau'_1) \Rightarrow \omega_p = (e''_0 \in \{p\})$ and $p \notin dom(\dot\tau'_1) \Rightarrow \omega_p = \neg(e''_0 \in dom(\dot\tau'_1) \cap P)$ (7) - hyp.7.1 + (1)
- $\mu_f, r \vDash \omega_0$, $\mu_f, r \vDash \omega_1$, $\mu_f, r \vDash \omega'_1$, and $\mu_f, r \vDash \omega_p$ (8) - hyp.7.3 + (7)
- $\dot\tau'_1 = \dot\tau_1$ and $\omega'_1 = \omega_1$ (9) - (1) + (2) + (8) + *Incompatible Assertions*
- $\dot\tau = \mathsf{PRIM}^{\pi_{\mathtt{lev}}}\big(\Rsh\,(\dot\tau_1, p)\big)_{\sqcup lev(\dot\tau_0)\sqcup lev(\dot\tau_1)}$ and $\omega = \omega_0 \wedge \omega_1 \wedge \omega_p$, where $p \in dom(\dot\tau_1) \Rightarrow \omega_p = (e''_0 \in \{p\})$ and $p \notin dom(\dot\tau_1) \Rightarrow \omega_p = \neg(e''_0 \in dom(\dot\tau_1) \cap P)$ (10) - (7) + (9)
- $lev(\dot\tau_0) \sqcup lev(\dot\tau_1) \sqcup \pi_{\mathtt{lev}}\,(\Rsh\,(\dot\tau_0, p)) \sqsubseteq \sigma$ (11) - hyp.7.2 + (7)
- $\mu_f, r \vDash \omega_0 \Leftrightarrow \mu_0, r \vDash \omega_0$, $\mu_f, r \vDash \omega_1 \Leftrightarrow \mu_1, r \vDash \omega_1$, and $\mu_f, r \vDash \omega_p \Leftrightarrow \mu_1, r \vDash \omega_p$
  (12) - (1) + (2) + *Invariance of Dynamic Assertions*
- $\mu'_f, r \vDash \omega_0 \Leftrightarrow \mu'_0, r \vDash \omega_0$, $\mu'_f, r \vDash \omega_1 \Leftrightarrow \mu'_1, r \vDash \omega_1$, and $\mu'_f, r \vDash \omega_p \Leftrightarrow \mu'_1, r \vDash \omega_p$
  (13) - (1) + (3) + *Invariance of Dynamic Assertions*
- $\mu'_0, r \vDash \omega_0$ and $m_0 = m'_0$ (14) - (4) + (8) + (11) + (12)
- $\mu'_1, r \vDash \omega_1$ and $r_1 = r'_1$ (15) - (5) + (8) + (11) + (12)
- $\mu'_f, r \vDash \omega_0$ and $\mu'_f, r \vDash \omega_1$ (16) - (13)-(15)
- $\mu_1, r \vDash \omega_p \Leftrightarrow (p \in dom(\dot\tau_1) \wedge m_0 = p) \vee (p \notin dom(\dot\tau_1) \wedge m_0 \notin dom(\dot\tau_1))$
  (17) - (2) + (7) + *Invariance of Bookkeeping Expressions*
- $\mu'_1, r \vDash \omega_p \Leftrightarrow (p \in dom(\dot\tau_1) \wedge m'_0 = p) \vee (p \notin dom(\dot\tau_1) \wedge m'_0 \notin dom(\dot\tau_1))$
  (18) - (3) + (7) + *Invariance of Bookkeeping Expressions*
- $(p \in dom(\dot\tau_1) \wedge m_0 = p) \vee (p \notin dom(\dot\tau_1) \wedge m_0 \notin dom(\dot\tau_1))$ (19) - (8) + (12) + (17)
- $(p \in dom(\dot\tau_1) \wedge m'_0 = p) \vee (p \notin dom(\dot\tau_1) \wedge m'_0 \notin dom(\dot\tau_1))$ (20) - (14) + (19)
- $\mu'_1, r \vDash \omega_p$ (21) - (18) + (20)
- $\mu'_f, r \vDash \omega_p$ (22) - (13) + (21)
- $\mu'_f, r \vDash \omega$ **(23)** - (10) + (16) + (22)
- $\Rsh\,(\dot\tau_1, m_0) = \Rsh\,(\dot\tau_1, m'_0) = \Rsh\,(\dot\tau_1, p)$ (24) - (7) + (14) + (19)
- $\mu_1, r \vDash \omega_1 \Rightarrow \Sigma_1(r_1) \preceq \dot\tau_1$ (25) - hyp.7.1 + (1) + (2) + *Well-labeled Memory*
- $\mu'_1, r \vDash \omega_1 \Rightarrow \Sigma'_1(r'_1) \preceq \dot\tau_1$ (26) - hyp.7.1 + (1) + (3) + *Well-labeled Memory*
- $\mu_1, r \vDash \omega_1 \Rightarrow \Sigma_1(r_1) \curlyvee \Sigma'_1(r'_1) \preceq \dot\tau_1$ (27) - (5) + (11) + (25) + (26)
- $\mu_f, r \vDash \omega_1 \Rightarrow \Sigma_f(r_1) \curlyvee \Sigma'_f(r'_1) \preceq \dot\tau_1$
  (28) - (2) + (27) + *Invariance of Dynamic Assertions*
- $\Sigma_f(r_1) = \Sigma'_f(r_1) \preceq \dot\tau_1$ (29) - (6) + (9) + (11) + (15) + (28)
- $\lfloor \Sigma_f(r_1) \rfloor = \lfloor \Sigma'_f(r'_1) \rfloor = \lfloor \dot\tau_1 \rfloor$ (30) - (29)
- $\Rsh\,(\dot\tau_1, p) = \Rsh\,(\dot\tau_1, m_0) = \Rsh\,(\Sigma_f(r_1), m_0)$ (31) - (19) + (30)
- $\pi_{\mathtt{lev}}\,(\Rsh\,(\Sigma_f(r_1), m_0)) = \pi_{\mathtt{lev}}\,(\Rsh\,(\dot\tau_1, p)) \sqsubseteq \sigma$ (32) - (11) + (31)
- $\pi_{\mathtt{lev}}\,(\Rsh\,(\Sigma_f(r_1), m_0)) \sqcup lev(\Sigma_f(r_1)) \sqsubseteq \sigma$ (33) - (11) + (29) + (32)
- $\hat r = \hat r'$ and $\hat r \neq \mathit{null} \Rightarrow lev(\Sigma_f(\hat r)) = lev(\Sigma'_f(\hat r')) \sqsubseteq \sigma$
  (34) - (2) + (3) + (6) + (33) + Proto-Chain Indistinguishability (Lemma **??**)
- $v_f = v'_f$ **(35)** - (2) + (3) + (34)

[Property Deletion] $e = \mathsf{delete}^i\ e_0.p$ for some expression e_0, property p, and index i (hyp.6). It follows:

- $\Gamma \vdash e_0 \rightsquigarrow e_0'/e_0'' : T_0, L_0$, $T = \{(\mathsf{PRIM}^{\perp}, \mathtt{tt})\}$, $e' = e_0', wrap(\omega, \hat{x}_i = \mathsf{delete}\ e_0''.p)$, where $\omega = When^{?}_{\sqsubseteq}(lev(T_0), \pi_{\mathtt{lev}}\left(\Rsh^{?}\ (T_0, \{p\}, e_0'')\right))$ (1) - hyp.1 + hyp.6
- $r \vdash \langle \mu, \Sigma, e_0' \rangle \Downarrow \langle \mu_0, \Sigma_f, r_0 \rangle$ for some mems. μ_0 and $\hat{\mu}$, labeling Σ_f, and reference r_0 such that: $\mu_0, r \vDash \omega$, $\hat{\mu} = \mu_0\left[r_0 \mapsto \mu_0(r_0)|_{dom(\mu_0(r_0)-p)}\right]$, $v_f = \mathtt{tt}$, and $\mu_f \sim \hat{\mu}$ (2) - hyp.2 + hyp.6
- $r \vdash \langle \mu', \Sigma', e_0' \rangle \Downarrow \langle \mu_0', \Sigma_f', r_0' \rangle$ for some mems. μ_0' and $\hat{\mu}'$, labeling Σ_f', and reference r_0' such that: $\mu_0', r \vDash \omega$, $\hat{\mu} = \mu_0'\left[r_0' \mapsto \mu_0'(r_0')|_{dom(\mu_0'(r_0')-p)}\right]$, $v_f' = \mathtt{tt}$, and $\mu_f' \sim \hat{\mu}'$ (3) - hyp.3 + hyp.6
- $\mu_0, \Sigma_0 \sim_\sigma \mu_0', \Sigma_0'$, $\Gamma, r \Vdash \mu_0 \sim_\sigma \mu_0'$, $\forall_{(\dot{\tau}_0, \omega_0) \in T_0} lev(\dot{\tau}_0) \sqsubseteq \sigma \Rightarrow (\mu_0, r \vDash \omega_0 \Leftrightarrow \mu_0', r \vDash \omega_0) \wedge (\mu_0, r \vDash \omega_0 \Rightarrow r_0 = r_0')$ (4) - **ih** + hyp.4 + hyp.5 + (1) + (2) + (3)
- $\Gamma, r \Vdash \mu_f \sim_\sigma \mu_f'$ **(5)** - (2)-(4)
- $v_f = v_f' = \mathtt{tt}$ (6) - (1) + (2)
- $\mu_f, r \vDash \mathtt{tt} \Rightarrow v_f = v_f'$ (7) - (6)
- $\mu_f, r \vDash \mathtt{tt} \Leftrightarrow \mu_f', r \vDash \mathtt{tt}$ (8) - *tautology*
- $\forall_{(\dot{\tau}, \omega) \in T} lev(\dot{\tau}) \sqsubseteq \sigma \Rightarrow (\mu_f, r \vDash \omega \Leftrightarrow \mu_f', r \vDash \omega) \wedge (\mu_f, r \vDash \omega \Rightarrow v_f = v_f')$ **(9)** - (1) + (7) + (8)
- $\forall_{\hat{\mu}, \hat{r}}\ \hat{\mu}, \hat{r} \vDash \omega \Leftrightarrow \exists_{(\dot{\tau}_0, \omega_0) \in T_0}\ lev(\dot{\tau}_0) \preceq \pi_{\mathtt{lev}}\left(\Rsh\ (\dot{\tau}_0, p)\right) \wedge \hat{\mu}, \hat{r} \vDash \omega_0$ (10) - (1)
- $\mu_0, r \vDash \omega \Leftrightarrow \exists_{(\dot{\tau}_0, \omega_0) \in T_0}\ lev(\dot{\tau}_0) \preceq \pi_{\mathtt{lev}}\left(\Rsh\ (\dot{\tau}_0, p)\right) \wedge \mu_0, r \vDash \omega_0$ (11) - (10)
- $lev(\dot{\tau}_0) \preceq \pi_{\mathtt{lev}}\left(\Rsh\ (\dot{\tau}_0, p)\right) \wedge \mu_0, r \vDash \omega_0$ for some $(\dot{\tau}_0, \omega_0) \in T_0$ (12) - (2) + (11)

We consider two cases: $lev(\dot{\tau}_0) \sqsubseteq \sigma$ and $lev(\dot{\tau}_0) \not\sqsubseteq \sigma$. Suppose $lev(\dot{\tau}_0) \sqsubseteq \sigma$ (hyp.7). It follows that:

- $r_0 = r_0'$ (13) - hyp.7 + (4) + (12)
- $\mu_f, \Sigma_f \sim_\sigma \mu_f', \Sigma_f'$ **(14)** - (2) + (3) + (4) + (13)

Suppose $lev(\dot{\tau}_0) \not\sqsubseteq \sigma$ (hyp.7). It follows that:

- $\pi_{\mathtt{lev}}\left(\Rsh\ (\dot{\tau}_0, p)\right) \not\sqsubseteq \sigma$ (15) - hyp.7 + (12)
- $\Sigma_f(r_0) \curlyvee \Sigma_f'(r_0') \preceq \dot{\tau}_0$ (16) - (1)-(4) + (12) + *Well-Lab. Mem.*
- $\lfloor \Sigma_f(r_0) \rfloor \equiv \lfloor \Sigma_f'(r_0') \rfloor \equiv \lfloor \dot{\tau}_0 \rfloor$ (17) - (16)
- $\pi_{\mathtt{lev}}\left(\Rsh\ (\dot{\tau}_0, p)\right) = \pi_{\mathtt{lev}}\left(\Rsh\ (\Sigma_f(r_0), p)\right) = \pi_{\mathtt{lev}}\left(\Rsh\ (\Sigma_f'(r_0'), p)\right)$ (18) - (17)
- $\pi_{\mathtt{lev}}\left(\Rsh\ (\Sigma_f(r_0), p)\right) = \pi_{\mathtt{lev}}\left(\Rsh\ (\Sigma_f'(r_0'), p)\right) \not\sqsubseteq \sigma$ (19) - (15) + (18)
- $\mu_f, \Sigma_f \sim_\sigma \mu_f', \Sigma_f'$ **(20)** - (2) + (3) + (4) + (19)

[Conditional Expression] $e = e_0\ ?^j\ (e_1) : (e_2)$ for three exprs. e_0, e_1, and e_2 (hyp.6). We conclude that:

- $\Gamma \vdash e_i \rightsquigarrow e_i'/e_i'' : T_i, L_i$ for $i \in \{0, 1, 2\}$, $\omega = When^{?}_{\sqsubseteq}(lev(T_0), L_1 \oplus_{\sqcap} L_2)$, $T = T_1^{\omega_{\mathtt{tt}}} \cup T_2^{\omega_{\mathtt{ff}}}$, $e' = e_0', wrap(\omega, e_0''\ ?\ (e_1', \hat{x}_j = e_1'') : (e_2', \hat{x}_j = e_2''))$, $\omega_{\mathtt{tt}} = \neg(e_0'' \in V_F)$, and $\omega_{\mathtt{ff}} = (e_0'' \in V_F)$ (1) - hyp.1 + hyp.6
- $r \vdash \langle \mu, \Sigma, e_0' \rangle \Downarrow \langle \mu_0, \Sigma_0, v_0 \rangle$ and $r \vdash \langle \mu_0, \Sigma_0, e_k' \rangle \Downarrow \langle \hat{\mu}, \Sigma_f, v_f \rangle$ for two mems. μ_0 and $\hat{\mu}$, labeling Σ_0, values v_0 and v_f such that: $\mu_f \sim \hat{\mu}$, $v_0 \notin V_F \Rightarrow k = 1$, $v_0 \in V_F \Rightarrow k = 2$, and $\mu_0, r \vDash \omega$ (2) - hyp.2 + hyp.6
- $r \vdash \langle \mu', \Sigma', e_0' \rangle \Downarrow \langle \mu_0', \Sigma_0', v_0' \rangle$ and $r \vdash \langle \mu_0', \Sigma_0', e_l' \rangle \Downarrow \langle \hat{\mu}', \Sigma_f', v_f' \rangle$ for two mems. μ_0' and $\hat{\mu}'$, labeling Σ_0', values v_0' and v_f' such that: $\mu_f' \sim \hat{\mu}'$, $v_0' \notin V_F \Rightarrow l = 1$, $v_0' \in V_F \Rightarrow l = 2$, and $\mu_0', r \vDash \omega$ (3) - hyp.2 + hyp.6
- $\mu_0, \Sigma_0 \sim_\sigma \mu_0', \Sigma_0'$, $\Gamma, r \Vdash \mu_0 \sim_\sigma \mu_0'$, $\forall_{(\dot{\tau}_0, \omega_0) \in T_0} lev(\dot{\tau}_0) \sqsubseteq \sigma \Rightarrow (\mu_0, r \vDash \omega_0 \Leftrightarrow \mu_0', r \vDash \omega_0) \wedge (\mu_0, r \vDash \omega_0 \Rightarrow v_0 = v_0')$ (4) - **ih** + hyp.4 + hyp.5 + (1) + (2) + (3)
- $\forall_{\hat{\mu}, \hat{r}}\ \hat{\mu}, \hat{r} \vDash \omega \Leftrightarrow \exists_{(\dot{\tau}_0, \omega_0) \in T_0, (\sigma_1, \omega_1) \in L_1, (\sigma_2, \omega_2) \in L_2}\ lev(\dot{\tau}_0) \preceq \sigma_1 \sqcap \sigma_2 \wedge \hat{\mu}, \hat{r} \vDash \omega_0 \wedge \omega_1 \wedge \omega_2$ (5) - (1)

– $lev(\dot{\tau}_0) \preceq \sigma_1 \sqcap \sigma_2 \wedge \mu_0, r \vDash \omega_0 \wedge \omega_1 \wedge \omega_2$, for some $(\dot{\tau}_0, \omega_0) \in T_0$, $(\sigma_1, \omega_1) \in L_1$, and $(\sigma_2, \omega_2) \in L_2$ (6) - (2) + (5)

We consider two cases: $lev(\dot{\tau}_0) \sqsubseteq \sigma$ and $lev(\dot{\tau}_0) \not\sqsubseteq \sigma$. Suppose $lev(\dot{\tau}_0) \sqsubseteq \sigma$ (hyp.7). It follows that:

– $v_0 = v_0'$ (7) - hyp.7 + (4) + (6)
– $k = l$ (8) - (2) + (3) + (7)
– $\hat{\mu}, \Sigma_f \sim_\sigma \hat{\mu}', \Sigma_f'$, $\Gamma, r \Vdash \hat{\mu} \sim_\sigma \hat{\mu}'$, $\forall_{(\dot{\tau}_k, \omega_k) \in T_k} lev(\dot{\tau}_k) \sqsubseteq \sigma \Rightarrow (\hat{\mu}, r \vDash \omega_k \Leftrightarrow \hat{\mu}', r \vDash \omega) \wedge (\hat{\mu}, r \vDash \omega \Rightarrow v_f = v_f')$ (9) - **ih** + (1) + (2) + (3) + (4) + (8)
– For all $(\dot{\tau}, \omega) \in T_1^{\omega_{\mathtt{tt}}} \cup T_2^{\omega_{\mathtt{ff}}}$, there is $(\dot{\tau}_l, \omega_l) \in T_l$ with $l \in \{1, 2\}$ such that: $(lev(\dot{\tau}) \sqsubseteq \sigma \wedge \hat{\mu}, \hat{r} \vDash \omega) \Rightarrow \big(lev(\dot{\tau}_l) \sqsubseteq \sigma \wedge \hat{\mu}, \hat{r} \vDash \omega_l \wedge (l = 1 \Rightarrow \hat{\mu}, \hat{r} \vDash \omega_{\mathtt{tt}}) \wedge (l = 2 \Rightarrow \hat{\mu}, \hat{r} \vDash \omega_{\mathtt{ff}})\big)$ (10) - *definition*
– For all $(\dot{\tau}, \omega) \in T_l^{\omega_l}$ with $l \in \{1, 2\} \backslash \{k\}$, where $\omega_l = \omega_{\mathtt{tt}}$ if $l = 1$ and $\omega_l = \omega_{\mathtt{ff}}$ if $l = 2$: $\hat{\mu}, \hat{r} \nvDash \omega_l$. (11) - (2)
– $\forall_{(\dot{\tau}, \omega) \in T} lev(\dot{\tau}) \sqsubseteq \sigma \Rightarrow (\mu_f, r \vDash \omega \Leftrightarrow \mu_f', r \vDash \omega) \wedge (\mu_f, r \vDash \omega \Rightarrow v_f = v_f')$ (12) - (11) + (9)

[Binary Operation] $e = e_0 \ \mathsf{op}^j \ e_1$ for two exprs. e_0 and e_1 (hyp.6). We conclude that:

– $\Gamma \vdash e_i \rightsquigarrow e_i'/e_i'' : T_i, L_i$, where: $i \in \{0, 1\}$, $e' = e_0', e_1', \hat{x}_j = e_0''$ op e_1'', and $T = T_0 \oplus_\Upsilon T_1$. (1) - hyp.1 + hyp.6
– $r \vdash \langle \mu, \Sigma, e_0' \rangle \Downarrow \langle \mu_0, \Sigma_0, v_0 \rangle$ and $r \vdash \langle \mu_0, \Sigma_0, e_1' \rangle \Downarrow \langle \mu_1, \Sigma_f, v_1 \rangle$ for some memories μ_0 and μ_1, labeling Σ_0, and two values v_0 and v_1 such that: $\mu_f \sim \mu_1$ and $v_f = v_0$ op v_1 (2) - hyp.2 + (1) + *Transparency*
– $r \vdash \langle \mu', \Sigma', e_0' \rangle \Downarrow \langle \mu_0', \Sigma_0', v_0' \rangle$ and $r \vdash \langle \mu_0', \Sigma_0', e_1' \rangle \Downarrow \langle \mu_1', \Sigma_f', v_1' \rangle$ for some memories μ_0' and μ_1', labeling Σ_0', and two values v_0' and v_1' such that: $\mu_f' \sim \mu_1'$ and $v_f' = v_0'$ op v_1' (3) - hyp.3 + (1) + *Transparency*
– $\mu_0, \Sigma_0 \sim_\sigma \mu_0', \Sigma_0'$, $\Gamma, r \Vdash \mu_0 \sim_\sigma \mu_0'$, and $\forall_{(\dot{\tau}_0, \omega_0) \in T_0} lev(\dot{\tau}_0) \sqsubseteq \sigma \Rightarrow (\mu_0, r \vDash \omega_0 \Leftrightarrow \mu_0', r \vDash \omega_0) \wedge (\mu_0, r \vDash \omega_0 \Rightarrow v_0 = v_0')$. (4) - **ih** + hyp.4 + hyp.5 + (1) + (2) + (3)
– $\mu_1, \Sigma_1 \sim_\sigma \mu_1', \Sigma_1'$, $\Gamma, r \Vdash \mu_1 \sim_\sigma \mu_1'$, and $\forall_{(\dot{\tau}_1, \omega_1) \in T_1} lev(\dot{\tau}_1) \sqsubseteq \sigma \Rightarrow (\mu_1, r \vDash \omega_1 \Leftrightarrow \mu_1', r \vDash \omega_1) \wedge (\mu_1, r \vDash \omega_1 \Rightarrow v_1 = v_1')$. (5) - **ih** + (1) + (2) + (3) + (4)
– $\mu_f, \Sigma_f \sim_\sigma \mu_f', \Sigma_f'$ and $\Gamma, r \Vdash \mu_f \sim_\sigma \mu_f'$
(6) - (1) + (2) + (3) + (5) + *Low-Equality Preservation for Internal Updates*

Suppose that $(\dot{\tau}, \omega) \in T$ (hyp.7), $lev(\dot{\tau}) \sqsubseteq \sigma$ (hyp.8), and $\mu_f, r \vDash \omega$ (hyp.9). It follows that there are $(\dot{\tau}_0, \omega_0) \in T_0$ and $(\dot{\tau}_1, \omega_1) \in T_1$ such that:

– $\forall_{\hat{\mu}, \hat{r}} \hat{\mu}, \hat{r} \vDash \omega \Leftrightarrow (\hat{\mu}, \hat{r} \vDash \omega_0 \wedge \omega_1) \wedge (\dot{\tau} = \dot{\tau}_0 \curlyvee \dot{\tau}_1)$ (7) - hyp.7 + (1)
– $\mu_f, r \vDash \omega \Leftrightarrow (\mu_f, r \vDash \omega_0 \wedge \omega_1) \wedge (\dot{\tau} = \dot{\tau}_0 \curlyvee \dot{\tau}_1)$ (8) - (7)
– $\mu_f, r \vDash \omega_0$, $\mu_f, r \vDash \omega_1$, and $\dot{\tau} = \dot{\tau}_0 \curlyvee \dot{\tau}_1$. (9) - hyp.9 + (8)
– $\mu_f, r \vDash \omega_0 \Leftrightarrow \mu_0, r \vDash \omega_0$ and $\mu_f, r \vDash \omega_1 \Leftrightarrow \mu_1, r \vDash \omega_1$
(10) - (1) + (2) + *Invariance of Dynamic Assertions*
– $\mu_f', r \vDash \omega_0 \Leftrightarrow \mu_0', r \vDash \omega_0$ and $\mu_f', r \vDash \omega_1 \Leftrightarrow \mu_1', r \vDash \omega_1$
(11) - (1) + (3) + *Invariance of Dynamic Assertions*
– $\mu_0, r \vDash \omega_0$ and $\mu_1, r \vDash \omega_1$ (12) - (9) + (10)
– $lev(\dot{\tau}_0) \sqsubseteq \sigma$ and $lev(\dot{\tau}_1) \sqsubseteq \sigma$ (13) - hyp.8 + (9)
– $\mu_0', r \vDash \omega_0$ and $v_0 = v_0'$ (14) - (4) + (12) + (13)
– $\mu_1', r \vDash \omega_1$ and $v_1 = v_1'$ (15) - (5) + (12) + (13)
– $\mu_f', r \vDash \omega_0$ and $\mu_f', r \vDash \omega_1$ (16) - (11) + (14) + (15)
– $\mu_f', r \vDash \omega_0 \wedge \omega_1$ **(17)** - (16)
– $v_f = v_f'$ **(18)** - (2) + (3) + (14) + (15)