JSAI Proof Sketches

1 Introduction

We sketch proofs of soundness for a subset of the abstract semantics that deal with objects. We begin by defining the concrete and abstract object lattices and the concretization function that maps abstract objects to concrete objects. We then give soundness theorems for a select subset of the abstract semantics.

2 Object Concrete Lattice

The concrete object semantic domain and associated helper functions are defined in the notJS semantics document. The concrete object lattice is the powerset of objects where the partial order is subset inclusion, join is set union, and meet is set intersection:

$$\mathcal{L} = (\mathcal{P}(\mathit{Object}), \subseteq, \cup, \cap)$$

3 Object Abstract Lattice

The abstract object semantic domain and associated helper functions are defined in the notJS semantics document. We use the following notation:

- Let $\sqsubseteq_x$, $\sqcup_x$, γ_x be the partial orders, joins, and concretization functions on abstract domains $x \in \{\mathit{String}^\sharp, \mathit{BValue}^\sharp, \mathit{Env}^\sharp, \mathit{Store}^\sharp, \mathcal{P}(\mathit{Closure}^\sharp)\}$. We will omit the subscript when the domain is obvious from the surrounding context.
- Let $\texttt{ext}(\hat{o}) = \pi_1(\hat{o})$ (the object's external map), $\texttt{int}(\hat{o}) = \pi_2(\hat{o})$ (the object's internal map), and $\texttt{def}(\hat{o}) = \pi_3(\hat{o})$ (the object's set of definitely-present properties).
- We consider maps to be sets of tuples, and therefore sometimes use set notation on maps.

The abstract object join semi-lattice is:

$$\mathcal{L}^\sharp = (\mathit{Object}^\sharp \cup \{\top\}, \sqsubseteq, \sqcup)$$

where:

- $\forall \hat{o} \in \mathit{Object}^\sharp \,.\, \hat{o} \sqsubseteq \top$. Objects are only comparable to other objects of the same class, thus we need an artificial top element to make the abstract domain a semi-lattice.
- $\hat{o}_1 \sqsubseteq \hat{o}_2$ iff
  - $\forall \widehat{str}_1 \in \texttt{dom}(\texttt{ext}(\hat{o}_1)) \,.\, \exists \widehat{str}_2 \in \texttt{dom}(\texttt{ext}(\hat{o}_2))$ such that $\widehat{str}_1 \sqsubseteq \widehat{str}_2$ and $\texttt{ext}(\hat{o}_1)(\widehat{str}_1) \sqsubseteq \texttt{ext}(\hat{o}_2)(\widehat{str}_2)$.
  - $\texttt{dom}(\texttt{int}(\hat{o}_1)) = \texttt{dom}(\texttt{int}(\hat{o}_2))$ and $\forall \mathit{str} \in \texttt{dom}(\texttt{int}(\hat{o}_1))$:
    * $\texttt{int}(\hat{o}_1)(\mathit{str}) = \texttt{int}(\hat{o}_2)(\mathit{str})$ if $\mathit{str} = \texttt{"class"}$,
    * $\texttt{int}(\hat{o}_1)(\mathit{str}) \sqsubseteq \texttt{int}(\hat{o}_2)(\mathit{str})$ otherwise.
  - $\texttt{def}(\hat{o}_2) \subseteq \texttt{def}(\hat{o}_1)$.

  This preorder is not antisymmetric and hence cannot be a semi-lattice, however if we define $\hat{o}_1$ and $\hat{o}_2$ to be equivalent iff $\hat{o}_1 \sqsubseteq \hat{o}_2$ and $\hat{o}_2 \sqsubseteq \hat{o}_1$, then the result is a semi-lattice. Note that all object's internal maps are guaranteed to have a `"class"` field and that all objects of the same class are guaranteed to have the same set of internal map fields.

- Join is defined as:

$$\hat{o}_1 \sqcup \hat{o}_2 = \begin{cases} \top & \text{if } \texttt{int}(\hat{o}_1)(\texttt{"class"}) \neq \texttt{int}(\hat{o}_2)(\texttt{"class"}) \\ (\mathit{extern}, \mathit{intern}, \mathit{present}) & \text{otherwise} \end{cases}$$

where:

$$\mathit{extern} = \mathit{same} \cup \mathit{distinct}$$

$$\mathit{same} = \left\{ (\widehat{str}, \widehat{bv}) \;\middle|\; (\widehat{str}, \widehat{bv}_1) \in \texttt{ext}(\hat{o}_1),\ (\widehat{str}, \widehat{bv}_2) \in \texttt{ext}(\hat{o}_2),\ \widehat{bv} = \widehat{bv}_1 \sqcup \widehat{bv}_2 \right\}$$

$$\mathit{distinct} = \left\{ (\widehat{str}, \widehat{bv}) \;\middle|\; (\widehat{str}, \widehat{bv}) \in \texttt{ext}(\hat{o}_1) \textbf{ xor } (\widehat{str}, \widehat{bv}) \in \texttt{ext}(\hat{o}_2) \right\}$$

$$\mathit{intern} = \{ (\mathit{str}, x) \mid \mathit{str} \in \texttt{dom}(\texttt{int}(\hat{o}_1)),\ x = \texttt{int}(\hat{o}_1)(\mathit{str}) \oplus \texttt{int}(\hat{o}_2)(\mathit{str}) \}$$

$$x \oplus y = \begin{cases} x & \text{if } x = y \\ x \sqcup y & \text{otherwise} \end{cases}$$

$$\mathit{present} = \texttt{def}(\hat{o}_1) \cap \texttt{def}(\hat{o}_2)$$

Note that in the object's internal maps the same field name is guaranteed to map to the same domain of values (either Class, BValue, or $\mathcal{P}(\mathit{Closure}^\sharp)$).

We could extend the semi-lattice to a full lattice by adding a $\perp$ element and a meet operator, but that isn't necessary for our purposes.

4 Object Concretization

We define the concretization function that maps abstract objects to sets of concrete objects. By abuse of notation we use the subscript i to range over all elements of a set; for example, the definition of def below says: "build a set of maps, each one of which maps each element of the definitely-present set to an element of the concretization of the value for the corresponding element in the external map".

$$\gamma \in \mathcal{L}^\sharp \to \mathcal{L}$$

$$\gamma(\hat{o}) = \{ (a \cup b, c) \mid a \in \mathit{def},\ b \in \mathcal{P}(\mathit{notdef}),\ c \in \mathit{intern} \}$$

where:

$$\mathit{def} = \{\ [\texttt{def}(\hat{o})_i \mapsto bv_i] \mid bv_i \in \gamma(\texttt{ext}(\hat{o})(\alpha(\texttt{def}(\hat{o})_i)))\ \}$$

$$\mathit{notdef} = \bigcup_{\widehat{str} \in X} \left\{\ (\mathit{str}, bv) \mid \mathit{str} \in \gamma(\widehat{str}),\ bv \in \gamma(\texttt{ext}(\hat{o})(\widehat{str}))\ \right\}$$

$$X = \texttt{dom}(\texttt{ext}(\hat{o})) \setminus \alpha(\texttt{def}(\hat{o}))$$

$$\mathit{intern} = \{\ [\texttt{dom}(\texttt{int}(\hat{o}))_i \mapsto x_i] \mid x_i \in \delta(\texttt{int}(\hat{o})(\texttt{dom}(\texttt{int}(\hat{o}))_i))\ \}$$

$$\delta(x) = \begin{cases} \{x\} & \text{if } x \in \mathit{Class} \\ \gamma(x) & \text{otherwise} \end{cases}$$

5 Proof Sketches

We provide soundness proof sketches for object access, update, and delete. Object access is an expression; we prove that the abstract base value returned by the abstract expression evaluation over-approximates the concrete base value returned by the concrete expression evaluation. Object update and delete are statements; we prove that the set of states returned by each statement in the abstract semantic rules over-approximates the state returned by each statement in the concrete semantic rules. For details on the definitions of these algorithms, see the specification of the **notJS** abstract semantics.

5.1 Object Access

Theorem 1. *For $\rho \in \gamma(\hat{\rho})$, $\sigma \in \gamma(\hat{\sigma})$, and e_1, e_2 such that $\eta(e_1, \rho, \sigma) = a$, $\eta(e_2, \rho, \sigma) = \mathit{str}$, $\eta^\sharp(e_1, \hat{\rho}, \hat{\sigma}) = \widehat{bv}_1$, $\eta^\sharp(e_2, \hat{\rho}, \hat{\sigma}) = \widehat{bv}_2$, $a \in \gamma(\widehat{bv}_1)$, $\mathit{str} \in \gamma(\widehat{bv}_2)$: $\eta(e_1.e_2, \rho, \sigma) \in \gamma(\eta^\sharp(e_1.e_2, \hat{\rho}, \hat{\sigma}))$.*

Proof. By induction on the length of the concrete prototype chain. η will call $\texttt{lookup}(a, str, \sigma)$, while $\eta^\sharp$ will call $\texttt{lookup}^\sharp(\pi_{\hat{a}}(\widehat{bv}_1), \pi_{\widehat{str}}(\widehat{bv}_2), \hat{\sigma})$. $\texttt{lookup}^\sharp$ in turn will call $\texttt{look}^\sharp$ for each $\hat{a} \in \pi_{\hat{a}}(\widehat{bv}_1)$. Let $\hat{a}$ be the one such that $a \in \gamma(\hat{a})$ (which by the premises must exist), $o = \sigma(a)$, $\hat{o} = \hat{\sigma}(\hat{a})$. By definition of γ on objects and stores, $o \in \gamma(\hat{o})$. Then we need to show that $\texttt{lookup}(a, str, \sigma) \subseteq \gamma(\texttt{look}^\sharp(\hat{o}, \widehat{str}, \hat{\sigma}))$, where $\widehat{str} = \pi_{\widehat{str}}(\widehat{bv}_2)$.

- **Base case 1:** $str \in \texttt{dom}(o)$**.** Then $\texttt{lookup}(a, str, \sigma) = o(str)$ by definition of $\texttt{lookup}$, and we must show that $\gamma(\texttt{look}^\sharp(\hat{o}, \widehat{str}, \hat{\sigma}))$ contains $o(str)$. By definition of γ for objects, there must be some $\widehat{str}' \in \texttt{dom}(\hat{o})$, $\widehat{str} \sqsubseteq \widehat{str}' \vee \widehat{str}' \sqsubseteq \widehat{str}$ and $o(str) \in \gamma(\hat{o}(\widehat{str}'))$. The value $\hat{o}(\widehat{str}')$ will be computed by the *local* component of $\texttt{look}^\sharp(\hat{o}, \widehat{str}, \hat{\sigma})$.

- **Base case 2:** $str \notin \texttt{dom}(o)$, $\pi_2(o)(\texttt{"proto"}) = \textsf{null}$**.** Then $\texttt{lookup}(a, str, \sigma) = \textsf{undef}$ by definition of $\texttt{lookup}$, and we must show that $\gamma(\texttt{look}^\sharp(\hat{o}, \widehat{str}, \hat{\sigma}))$ contains $\textsf{undef}$. By definition of γ for objects, $\widehat{str} \notin \pi_3(\hat{o})$ and $\pi_{\widehat{null}}(\texttt{getProto}^\sharp(\hat{o})) \neq \emptyset$. The value $\texttt{inject}^\sharp(\textsf{undef})$ will be computed by the *fin* component of $\texttt{look}^\sharp(\hat{o}, \widehat{str}, \hat{\sigma})$.

- **Inductive case:** $str \notin \texttt{dom}(o)$, $\pi_2(o)(\texttt{"proto"}) = a'$**.** Then $\texttt{lookup}(a, str, \sigma) = \texttt{lookup}(a', str, \sigma)$ by definition of $\texttt{lookup}$. By definition of γ for objects, $\widehat{str} \notin \pi_3(\hat{o})$ and $\exists \hat{a}' \in \pi_2(\hat{o})(\texttt{"proto"}) \,.\, a' \in \gamma(\hat{a}')$. Then $\texttt{look}^\sharp(\hat{\sigma}(\hat{a}'), \widehat{str}, \hat{\sigma})$ will be computed by the *chain* component of $\texttt{look}^\sharp(\hat{o}, \widehat{str}, \hat{\sigma})$. Since by construction there cannot exist cycles in the prototype chain, this recursive call will eventually (potentially via a series of inductive cases) trigger one of the two base cases, and thus $\texttt{lookup}(a, str, \sigma) \subseteq \gamma(\texttt{look}^\sharp(\hat{o}, \widehat{str}, \hat{\sigma}))$.

□

5.2 Object Update

Theorem 2. *For* $\rho \in \gamma(\hat{\rho})$, $\sigma \in \gamma(\hat{\sigma})$, $\kappa \in \gamma(\hat{\kappa})$*:* $\mathcal{F}(e_1.e_2 := e_3, \rho, \sigma, \kappa) \in \gamma(\mathcal{F}^\sharp(e_1.e_2 := e_3, \hat{\rho}, \hat{\sigma}, \hat{\kappa}))$.

Proof. By cases. The rule for $\mathcal{F}(e_1.e_2 := e_3, \rho, \sigma, \kappa)$ will call $\texttt{updateObj}(bv_1, bv_2, bv_3\sigma)$ (where $bv_1 = [\![e_1]\!]$, $bv_2 = [\![e_2]\!]$, $bv_3 = [\![e_3]\!]$) to compute a new value v and store σ', producing the state $(v, \rho, \sigma', \kappa)$. The rules for $\mathcal{F}^\sharp(e_1.e_2 := e_3, \hat{\rho}, \hat{\sigma}, \hat{\kappa})$ will call $\texttt{updateObj}^\sharp(\widehat{bv}_1, \widehat{bv}_2, \widehat{bv}_3, \hat{\sigma})$ (where $\widehat{bv}_1 = [\![e_1]\!]$, $\widehat{bv}_2 = [\![e_2]\!]$, $\widehat{bv}_3 = [\![e_3]\!]$ and $bv_1 \in \gamma(\widehat{bv}_1)$, $bv_2 \in \gamma(\widehat{bv}_2)$, $bv_3 \in \gamma(\widehat{bv}_3)$) to compute one or both of a new (base value, store) pair $(\widehat{bv}, \hat{\sigma}')$ and a exceptional value $\widehat{ev}$, producing one or both of the new states $(\widehat{bv}, \hat{\rho}, \hat{\sigma}', \hat{\kappa})$ and $(\widehat{ev}, \hat{\rho}, \hat{\sigma}, \hat{\kappa})$.

- **Case 1:** $v = \textsf{exc}\ \texttt{"TypeError"}$**.** Then by definition of $\texttt{updateObj}$, $bv_1 \in \{\textsf{null}, \textsf{undef}\}$ and $\sigma' = \sigma$. By definition of γ on $BValue^\sharp$, $\pi_{\widehat{null}}(\widehat{bv}_1) = \{\textsf{null}\}$ or $\pi_{\widehat{undef}}(\widehat{bv}_1) = \{\textsf{undef}\}$. Thus via the *exc* case $\texttt{updateObj}^\sharp$ will return $\widehat{ev}$ such that $ev \in \gamma(\widehat{ev})$.

- **Case 2:** $v = \textsf{exc}\ \texttt{"RangeError"}$**.** Then by definition of $\texttt{updateObj}$, $a = bv_1$, $str = bv_2$, $\texttt{getClass}(\sigma(a)) = \textsf{array}$, $str = \texttt{"length"}$, $\neg u32?(bv_3)$. By definition of γ on $BValue^\sharp$, $maybeArray?$ and $\texttt{"length"} \in \gamma(\widehat{str})$ and $notU32(\widehat{bv})$. Thus via the *exc* case $\texttt{updateObj}^\sharp$ will return $\widehat{ev}$ such that $ev \in \gamma(\widehat{ev})$.

- **Case 3:** $v = bv_3$, $a = bv_1$, $str = bv_2$, $\texttt{getClass}(\sigma(a)) = \textsf{array}$, $str = \texttt{"length"}$, $u32?(bv_3)$**.** Then by definition of $\texttt{updateObj}$, $v = bv_3$ and σ' equals σ except it maps a to its old object updated to map $\texttt{"length"}$ to bv_3 and to remove all numeric properties greater than the new value of $\texttt{"length"}$. By definition of γ on $BValue^\sharp$, $\exists \hat{a} \in \pi_{\hat{a}}(\widehat{bv}_1)$ such that $a \in \gamma(\hat{a})$ and $newobj$ includes $\texttt{insert}^\sharp(\hat{\sigma}(\hat{a}), \widehat{str}, \widehat{bv}_3)$. By definition of γ on $BValue^\sharp$, $array?$ must be true and $\texttt{"length"} \in \gamma(\widehat{str})$. If $\{\texttt{"length"}\} = \gamma(\widehat{str})$ then $\texttt{insert}^\sharp$ weakly deletes all numeric properties in the object and leaves $\texttt{"length"}$ as a generic u32; otherwise it strongly removes all numeric properties and updates $\widehat{str}$ to map to $\widehat{bv}_3$; this new object will be joined with the old object by $\texttt{updateObj}^\sharp$. Thus $\texttt{updateObj}^\sharp$ will return a pair $(\widehat{bv}_3, \hat{\sigma}')$ such that $\hat{\sigma}'$ equals $\hat{\sigma}$ except it maps $\hat{a}$ to its old object with all numeric properties weakly removed and $\texttt{"length"}$ as a generic u32 value, and possibly other abstract addresses to weakly updated objects.

- **Case 4:** $v = bv_3$, $a = bv_1$, $str = bv_2$, $\texttt{getClass}(\sigma(a)) = \textsf{array}$, $str \neq \texttt{"length"}$, $u32?(\textsf{tonum}\ str)$**.** Then by definition of $\texttt{updateObj}$, $v = bv_3$ and σ' equals σ except it maps a to its old object updated to map str to bv_3 and with $\texttt{"length"}$ adjusted appropriately. By definition of γ on $BValue^\sharp$, $\exists \hat{a} \in \pi_{\hat{a}}(\widehat{bv}_1)$ such that $a \in \gamma(\hat{a})$ and $newobj$ includes $\texttt{insert}^\sharp(\hat{\sigma}(\hat{a}), \widehat{str}, \widehat{bv}_3)$. By definition of γ on $BValue^\sharp$, $array?$ must be true and $\{\texttt{"length"}\} \neq \gamma(\widehat{str})$. Then $\texttt{insert}^\sharp$ maps $\widehat{str}$ to $\widehat{bv}_3$, deletes all numeric properties, and $\texttt{"length"}$ is guaranteed to remain a generic u32; this new object will be joined with the old object by $\texttt{updateObj}^\sharp$. Thus $\texttt{updateObj}^\sharp$ will return a pair $(\widehat{bv}_3, \hat{\sigma}')$ such that $\hat{\sigma}'$ equals $\hat{\sigma}$ except it maps $\hat{a}$ to its old object with $\widehat{str}$ mapped to $\widehat{bv}_3$, all numeric properties weakly removed, $\texttt{"length"}$ as a generic u32 value, and possibly other abstract addresses to weakly updated objects.

- **Case 5:** $v = bv_3,\ a = bv_1,\ str = bv_2,\ \texttt{getClass}(\sigma(a)) \neq \textsf{\textbf{array}} \vee (str \neq \texttt{"length"},\ \neg u32?(\textsf{\textbf{tonum}}\ str))$**.** Then by definition of $\texttt{updateObj}$, $v = bv_3$ and σ' equals σ except it maps a to its old object updated to map str to bv_3. By definition of γ on $BValue^\sharp$, $\exists \hat{a} \in \pi_{\hat{a}}(\widehat{bv_1})$ such that $a \in \gamma(\hat{a})$ and $newobj$ includes $\texttt{insert}^\sharp(\hat{\sigma}(\hat{a}), \widehat{str}, \widehat{bv_3})$. By definition of γ on $BValue^\sharp$, $array?$ must be false or $\{\texttt{"length"}\} \neq \gamma(\widehat{str})$. Then $\texttt{insert}^\sharp$ maps $\widehat{str}$ to $\widehat{bv_3}$; any other changes will be weak. Thus $\texttt{updateObj}^\sharp$ will return a pair $(\widehat{bv_3}, \hat{\sigma}')$ such that $\hat{\sigma}'$ equals $\hat{\sigma}$ except it maps $\hat{a}$ to its old object with $\widehat{str}$ mapped to $\widehat{bv_3}$, and possibly other abstract addresses to weakly updated objects.

□

5.3 Object Delete

Theorem 3. *For $\rho \in \gamma(\hat{\rho})$, $\sigma \in \gamma(\hat{\sigma})$, $\kappa \in \gamma(\hat{\kappa})$: $\mathcal{F}(\textsf{\textbf{del}}\ e_1.e_2, \rho, \sigma, \kappa) \in \gamma(\mathcal{F}^\sharp(\textsf{\textbf{del}}\ e_1.e_2, \hat{\rho}, \hat{\sigma}, \hat{\kappa}))$.*

Proof. By cases. The rule for $\mathcal{F}(\textsf{\textbf{del}}\ e_1.e_2, \rho, \sigma, \kappa)$ will call $\texttt{delete}(bv_1, bv_2, x, \rho, \sigma)$ (where $bv_1 = [\![e_1]\!]$, $bv_2 = [\![e_2]\!]$) to compute a new value v and store σ', producing the state $(v, \rho, \sigma', \kappa)$. The rules for $\mathcal{F}^\sharp(\textsf{\textbf{del}}\ e_1.e_2, \hat{\rho}, \hat{\sigma}, \hat{\kappa})$ will call $\texttt{delete}^\sharp(\widehat{bv_1}, \widehat{bv_2}, x, \hat{\rho}, \hat{\sigma})$ (where $\widehat{bv_1} = [\![e_1]\!]$, $\widehat{bv_2} = [\![e_2]\!]$ and $bv_1 \in \gamma(\widehat{bv_1})$, $bv_2 \in \gamma(\widehat{bv_2})$) to compute one or both of a new store $\hat{\sigma}'$ and a exceptional value $\widehat{ev}$, producing one or both of the new states $(\texttt{inject}^\sharp(\textsf{\textbf{undef}}), \hat{\rho}, \hat{\sigma}', \hat{\kappa})$ and $(\widehat{ev}, \hat{\rho}, \hat{\sigma}, \hat{\kappa})$.

- **Case 1:** $v = ev$**.** Then by definition of $\texttt{delete}$, $bv_1 \in \{\textsf{\textbf{null}}, \textsf{\textbf{undef}}\}$ and $\sigma' = \sigma$. By definition of γ on $BValue^\sharp$, $\pi_{\widehat{null}}(\widehat{bv_1}) = \{\textsf{\textbf{null}}\}$ or $\pi_{\widehat{undef}}(\widehat{bv_1}) = \{\textsf{\textbf{undef}}\}$. Thus via the exc case $\texttt{delete}^\sharp$ will return $\widehat{ev}$ such that $ev \in \gamma(\widehat{ev})$.

- **Case 2:** $v = bv,\ bv_1 = a,\ bv_2 = str,\ str \in \texttt{dom}(\sigma(a))$**.** Then by definition of $\texttt{delete}$, $bv = \textsf{\textbf{undef}}$ and σ' equals σ except it maps a to its associated object with property str removed and maps $\rho(x)$ to $\textsf{\textbf{true}}$. By definition of γ on $BValue^\sharp$, $\exists \hat{a} \in \pi_{\hat{a}}(\widehat{bv_1})\,.\,a \in \gamma(\hat{a})$ and $str \in \gamma(\widehat{bv_2})$ and $\exists \widehat{str}' \in \texttt{dom}(\hat{\sigma}(\hat{a}))\,.\,str \in \gamma(\widehat{str}')$. Thus $defAbsent$ cannot be true, and so $\texttt{delete}^\sharp$ will return a store $\hat{\sigma}'$ that equals $\hat{\sigma}$ except it maps $\hat{a}$ to its associated object with property $\widehat{str}'$ either strongly or weakly removed and maps $\hat{\rho}(x)$ to a value that includes $\textsf{\textbf{true}}$.

- **Case 3:** $v = bv,\ bv_1 \neq a \vee bv_2 \neq str \vee str \notin \texttt{dom}(\sigma(a))$**.** Then by definition of $\texttt{delete}$, $bv = \textsf{\textbf{undef}}$ and σ' equals σ except it maps $\rho(x)$ to $\textsf{\textbf{false}}$. By definition of γ on $BValue^\sharp$, $defPresent$ cannot be true and $|newobj| > 0$. Thus $\texttt{delete}^\sharp$ will return a store $\hat{\sigma}'$ that equals $\hat{\sigma}$ except it maps all $\hat{a} \in \pi_{\hat{a}}(\widehat{bv_1})$ to either their old value or their old value with the $\widehat{str}$ property weakly removed (which therefore over-approximates the old value by definition of $\sqsubseteq$ on objects) and maps $\hat{\rho}(x)$ to a value that includes $\textsf{\textbf{false}}$.

□