IMPERIAL

Project Notes

Background & Progress Report

Author: Opale Sjöstedt

Supervisor: Philippa Gardner

August 24, 2024

Submitted in partial fulfilment of the requirements for the MSc Degree in Computing (Software Engineering)

Contents

Chapter 1

Proofs of soundness

1.1 Exclusive

1.1.1 Resource Algebra

$$\text{Ex}(X) \stackrel{\text{def}}{=} \text{ex}(x : X)$$
$$|\text{ex}(x)| \stackrel{\text{def}}{=} \perp$$
$$\text{ex}(x_1) \cdot \text{ex}(x_2) \text{ is always undefined}$$

We define the actions of the state model as $\mathcal{A} = \{\texttt{load}, \texttt{store}\}$, and the predicates $\Delta = \{\texttt{ex}\}$. We define predicate satisfiability and symbolic interpretation as:

$$\textsc{ExPredSat}\quad \frac{\sigma = \text{ex}(x)}{\sigma \models_\Delta \langle \texttt{ex} \rangle([]; [x])} \qquad \textsc{ExSymInterpretation}\quad \frac{[\![\hat{x}]\!]_{\theta,s} = x}{\theta, s, \text{ex}(x) \models \text{ex}(\hat{x})}$$

1.1.2 Compositional Concrete Rules

$$\textsc{CExLoadOk}\quad \texttt{load}(\text{ex}(x), []) = (\texttt{Ok}, \text{ex}(x), [x]) \qquad \textsc{CExLoadMiss}\quad \texttt{load}(\perp, []) \rightsquigarrow (\texttt{Miss}, \perp, [])$$

$$\textsc{CExStoreOk}\quad \texttt{store}(\text{ex}(x), [x']) \rightsquigarrow (\texttt{Ok}, \text{ex}(x'), []) \qquad \textsc{CExStoreMiss}\quad \texttt{store}(\perp, [x']) \rightsquigarrow (\texttt{Miss}, \perp, [])$$

1.1.3 Compositional Symbolic Rules

$$\textsc{ExLoadOk}\quad \texttt{load}(\text{ex}(\hat{x}), []) \rightsquigarrow (\texttt{Ok}, \text{ex}(\hat{x}), [\hat{x}], []) \qquad \textsc{ExLoadMiss}\quad \texttt{load}(\perp, []) \rightsquigarrow (\texttt{Miss}, \perp, [], [])$$

$$\textsc{ExStoreOk}\quad \texttt{store}(\text{ex}(\hat{x}), [\hat{x}']) \rightsquigarrow (\texttt{Ok}, \text{ex}(\hat{x}'), [], []) \qquad \textsc{ExStoreMiss}\quad \texttt{store}(\perp, [\hat{x}']) \rightsquigarrow (\texttt{Miss}, \perp, [], [])$$

$$\textsc{ExConsOk}\quad \texttt{consume}(\text{ex}(\hat{x}), \texttt{ex}, []) \rightsquigarrow (\texttt{Ok}, \perp, [\hat{x}], []) \qquad \textsc{ExConsMiss}\quad \texttt{consume}(\perp, \texttt{ex}, []) \rightsquigarrow (\texttt{Miss}, \perp, [], [])$$

ExProd
$\texttt{produce}(\bot, \texttt{ex}, [], [\hat{x}]) \rightsquigarrow (\mathrm{ex}(\hat{x}), [])$

ExFix
$\texttt{fix}\ [] = [\{\exists \hat{x}.\, \langle \texttt{ex} \rangle(;\hat{x})\}]$

1.1.4 Soundness Proofs

Proof.

Proposition: OX Soudness
Assume

(H1) $\theta, s, \sigma \models \hat{\sigma} \wedge \alpha(\sigma, \vec{v}_i) = (o, \sigma', \vec{v}_o) \wedge [\![\vec{\hat{v}}_i]\!]_{\theta,s} = \vec{v}_i$

(H2) $\forall o', \hat{\sigma}', \vec{\hat{v}}'_o, \pi'.\, \alpha(\hat{\sigma}, \vec{\hat{v}}_i) \rightsquigarrow (o', \hat{\sigma}', \vec{\hat{v}}'_o, \pi') \Rightarrow o' \in \{\texttt{Ok}, \texttt{Err}\}$

To prove

(G1) $\exists \hat{\sigma}', \vec{\hat{v}}_o, \pi, \theta'.\, \hat{\alpha}(\hat{\sigma}, \vec{\hat{v}}_i) \rightsquigarrow (o, \hat{\sigma}', \vec{\hat{v}}_o, \pi) \wedge \theta', s, \sigma' \models \hat{\sigma}' \wedge \mathsf{SAT}_{\theta',s}(\pi) \wedge [\![\vec{\hat{v}}_o]\!]_{\theta',s} = \vec{v}_o$

The proof is analogous for both actions, so we only consider the case where $\alpha = \texttt{load}$. Given (H1) and the definition of $\models$, there are two further cases: $\sigma = \mathrm{ex}(x)$ and $\hat{\sigma} = \mathrm{ex}(\hat{x})$, or $\sigma = \bot$ and $\hat{\sigma} = \bot$. Again, both cases are analogous, so we only consider $\sigma = \mathrm{ex}(x)$, $\hat{\sigma} = \mathrm{ex}(\hat{x})$.

(H3) From our hypothesis, $\sigma = \mathrm{ex}(x)$ and $\hat{\sigma} = \mathrm{ex}(\hat{x})$

(H4) From the definition of concrete actions CExLoadOk, we get $\vec{v}_i = []$, $o = \texttt{Ok}$, $\sigma' = \sigma$, $\vec{v}_o = [x]$.

(H5) From the definition of $\models$ we also have $[\![\hat{x}]\!]_{\theta,s} = x$

(H6) We can pick $\vec{\hat{v}}_o$, π and θ' such that $\vec{\hat{v}}_o = [\hat{x}]$, $\pi = []$ and $\theta' = \theta$.

(H7) From (H6), we get $[\![\vec{\hat{v}}_o]\!]_{\theta,s} = \vec{v}_o$, as well as $\mathsf{SAT}_{\theta,s}(\pi)$

(H8) From ExLoadOk and (H6), we have $\texttt{load}(\hat{\sigma}, \vec{\hat{v}}_i) \rightsquigarrow (\texttt{Ok}, \hat{\sigma}', \vec{\hat{v}}_o, \pi)$ with $\hat{\sigma}' = \hat{\sigma}$.

(H9) Finally, from (H4) and (H8), we have $\sigma' = \sigma$ and $\hat{\sigma}' = \hat{\sigma}$, thus from (H1) it follows that $\theta, s, \sigma' \models \hat{\sigma}'$.

Combining (H7), (H8) and (H9), we get our goal (G1).

Proposition: UX Soudness
Assume

(H1) $\hat{\alpha}(\hat{\sigma}, \vec{\hat{v}}_i) \rightsquigarrow (o, \hat{\sigma}', \vec{\hat{v}}_o, \pi) \wedge \mathsf{SAT}_{\theta,s}(\pi) \wedge \theta, s, \sigma' \models \hat{\sigma}' \wedge$
$[\![\vec{\hat{v}}_o]\!]_{\hat{s},\pi} \rightsquigarrow (\vec{v}_o, \pi') \wedge [\![\vec{\hat{v}}_i]\!]_{\hat{s},\pi'} \rightsquigarrow (\vec{v}_i, \pi'')$

To prove

(G1) $\exists \sigma.\, \theta, s, \sigma \models \hat{\sigma} \wedge \alpha(\sigma, \vec{v}_i) = (o, \sigma', \vec{v}_o)$

We again only consider the case where $\alpha = \texttt{load}$ and $\sigma' = \mathrm{ex}(x)$, $\hat{\sigma}' = \mathrm{ex}(\hat{x})$ – the other three cases are analogous.

(H2) From ExLoadOk, we get $\hat{\sigma} = \hat{\sigma}' = \mathrm{ex}(\hat{x})$, $\vec{\hat{v}}_i = []$, $o = \texttt{Ok}$, $\vec{\hat{v}}_o = [\hat{x}]$, $\pi = []$, with $s = [x \mapsto \hat{x}]$

(H3) From (H2) and (H1) we have $\vec{v}_i = []$ and $\vec{v}_o = [x]$

(H4) We can pick $\sigma = \sigma' = \text{ex}(x)$, which from (H1) and (H2) give us $\theta, s, \sigma \models \hat{\sigma}$

(H5) From CExLoadOk, (H3) and (H4), we have $\texttt{load}(\sigma, \vec{v}_i) = (\texttt{Ok}, \sigma', \vec{v}_o)$.

Combinding (H4) and (H5) gives our goal (G1).

Proposition: Frame subtraction is satisfied
Assume

(H1) $\sigma \mathbin{\#} \sigma_f \wedge \alpha(\sigma \cdot \sigma_f, \vec{v}_i) = (o, \sigma', \vec{v}_o)$

To prove

(G1) $$\begin{aligned}&\exists \sigma'', o', \vec{v}'_o.\, \alpha(\sigma, \vec{v}_i) = (o', \sigma'', \vec{v}'_o) \wedge \\ &\quad (o' \neq \texttt{Miss} \implies o' = o \wedge \vec{v}'_o = \vec{v}_o \wedge \sigma' = \sigma'' \cdot \sigma_f)\end{aligned}$$

(H2) $\texttt{load}$ and $\texttt{store}$ are always defined for respectively 0 and 1 arguments, so from (H1) we know $\exists \sigma'', o', \vec{v}'_o.\, \alpha(\sigma, \vec{v}_i) = (o', \sigma'', \vec{v}'_o)$.

(H3) Assume $o' \neq \texttt{Miss}$

(H4) If $\sigma = \bot$, the rules say $o' = \texttt{Miss}$, contradicting (H3). Thus $\sigma = \text{ex}(x)$.

(H5) From (H1), we know $\sigma \cdot \sigma_f$ is defined, so it must be that $\sigma_f = \bot$ as $\text{ex}(x) \cdot \text{ex}(y)$ is undefined.

From (H5) and composition rules we know $\sigma \cdot \sigma_f = \text{ex}(x) \cdot \bot = \text{ex}(x) = \sigma$. This gives our goal (G1).

Proposition: Frame addition is satisfied
Assume

(H1) $\alpha(\sigma, \vec{v}_i) = (o, \sigma', \vec{v}_o)$

(H2) $\sigma_f \mathbin{\#} \sigma'$

(H3) $o \neq \texttt{Miss}$

To prove

(G1) $\sigma \mathbin{\#} \sigma_f \wedge \alpha(\sigma \cdot \sigma_f, \vec{v}_i) = (o, \sigma' \cdot \sigma_f, \vec{v}_o)$

(H4) From (H3) and the action rules, we know $\sigma' = \text{ex}(x)$.

(H5) From composition, (H4) and (H2), we get $\sigma_f = \bot$.

From (H5) we get $\sigma' \cdot \sigma_f = \sigma' \cdot \bot = \sigma'$, and from (H1) our goal (G1) follows.

Proposition: Consume soundness
Assume

(H1) $\texttt{consume}(\hat{\sigma}, \delta, \vec{\hat{v}}_i) \rightsquigarrow (\texttt{Ok}, \hat{\sigma}_f, \vec{\hat{v}}_o, \pi)$

(H2) $\theta, s, \sigma \models \hat{\sigma} \wedge \mathsf{SAT}_{\theta,s}(\pi)$

To prove

(G1) $\exists \sigma_\delta, \sigma_f.\, \sigma_\delta \mathbin{\#} \sigma_f \wedge \sigma = \sigma_\delta \cdot \sigma_f \wedge \theta, s, \sigma_\delta \models_\Delta \langle \delta \rangle (\vec{\hat{v}}_i; \vec{\hat{v}}_o) \wedge \theta, s, \sigma_f \models \hat{\sigma}_f$

(H3) There is only one consume rules yielding Ok, giving us $\delta = \mathsf{ex}$, $\hat{\sigma} = \mathrm{ex}(\hat{x})$, $\hat{\sigma}_f = \bot$, $\vec{\hat{v}}_i = []$, $\vec{\hat{v}}_o = [\hat{x}]$, $\pi = []$

(H4) From the definition of $\models$, we must have $\sigma = \mathrm{ex}(x)$ and $\sigma_f = \bot$ such that $\theta, s, \sigma \models \hat{\sigma}$ and $\theta, s, \sigma_f \models \hat{\sigma}_f$.

(H5) From (H4), we have $[\![\hat{x}]\!]_{\theta,s} = x$, such that $[\![\vec{\hat{v}}_i]\!]_{\theta,s} = []$ and $[\![\vec{\hat{v}}_o]\!]_{\theta,s} = [x]$.

(H6) From the definition of composition, we must have $\sigma_\delta = \mathrm{ex}(x)$ such that $\sigma = \sigma_\delta \cdot \sigma_f$, which also implies $\sigma_\delta \mathbin{\#} \sigma_f$

(H7) From (H6), (H5) and the definition of $\models_\Delta$, we have $\theta, s, \sigma_\delta \models_\Delta \langle\delta\rangle(\vec{\hat{v}}_i; \vec{\hat{v}}_o)$.

Combining (H4), (H6) and (H7) gives our goal (G1).

Proposition: Consume completeness
Assume

(H1) $\mathsf{consume}(\hat{\sigma}, \delta, \vec{\hat{v}}_i) \rightsquigarrow (\mathsf{Ok}, \hat{\sigma}_f, \vec{\hat{v}}_o, \pi)$

(H2) $\theta, s, \sigma_f \models \hat{\sigma}_f \wedge \theta, s, \sigma_\delta \models_\Delta \langle\delta\rangle(\vec{\hat{v}}_i; \vec{\hat{v}}_o) \wedge \sigma_f \mathbin{\#} \sigma_\delta$

To prove

(G1) $\exists \sigma.\, \sigma = \sigma_\delta \cdot \sigma_f \wedge \theta, s, \sigma \models \hat{\sigma} \wedge \mathsf{SAT}_{\theta,s}(\pi)$

(H3) There is only one consume rules yielding Ok, giving us $\delta = \mathsf{ex}$, $\hat{\sigma} = \mathrm{ex}(\hat{x})$, $\hat{\sigma}_f = \bot$, $\vec{\hat{v}}_i = []$, $\vec{\hat{v}}_o = [\hat{x}]$, $\pi = []$

(H4) From (H3) and the definition of $\models$, if $\theta, s, \sigma_f \models \hat{\sigma}_f$ we have $\sigma_f = \bot$.

(H5) From (H3) and the definition of $\models_\Delta$, if $\theta, s, \sigma_\delta \models_\Delta \langle\delta\rangle(\vec{\hat{v}}_i; \vec{\hat{v}}_o)$, we have $\sigma_\delta = \mathrm{ex}(x)$, with $\vec{v}_i = []$ and $\vec{v}_o = [x]$ such that $[\![\vec{\hat{v}}_i]\!]_{\theta,s} = \vec{v}_i$ and $[\![\vec{\hat{v}}_o]\!]_{\theta,s} = \vec{v}_o$.

(H6) From (H4), (H5) and the rule for composition, we have $\sigma_f \mathbin{\#} \sigma_\delta$.

(H7) We can pick $\sigma = \sigma_\delta = \mathrm{ex}(x)$.

(H8) From the definition of composition, (H4) and (H5), we have $\sigma = \sigma_\delta \cdot \sigma_f$.

(H9) Given (H3) and (H7), we have $\theta, s, \sigma \models \hat{\sigma}$.

(H10) From (H3), $\pi = []$ thus $\mathsf{SAT}_{\theta,s}(\pi)$. Together with (H8) and (H9), this gives our goal (G1).

Proposition: Consume: OX sound
Assume

(H1) $\forall o, \hat{\sigma}_f, \vec{\hat{v}}_o, \pi.\, \mathsf{consume}(\mathsf{OX}, \hat{\sigma}, \delta, \vec{\hat{v}}_i) \rightsquigarrow (o, \hat{\sigma}_f, \vec{\hat{v}}_o, \pi) \Rightarrow o_c = \mathsf{Ok}$

To prove

(G1) $\exists \hat{\sigma}'_f, \vec{\hat{v}}'_o, \pi'.\, \mathsf{consume}(\mathsf{OX}, \hat{\sigma}, \delta, \vec{v}_i, \pi) \rightsquigarrow (\mathsf{Ok}, \hat{\sigma}'_f, \vec{\hat{v}}'_o, \pi')$

From the consume rules, we never vanish, so if all consumptions are Ok we must have $\hat{\sigma} = \mathrm{ex}(\hat{x})$, $\delta = \mathsf{ex}$, $\vec{\hat{v}}_i = []$, $\hat{\sigma}'_f = \bot$, $\vec{\hat{v}}'_o = [\hat{x}]$, $\pi' = []$. Our goal (G1) follows.

Proposition: Produce soundness
Assume

(H1) $\mathsf{produce}(\hat{\sigma}_f, \delta, \vec{\hat{v}}_i, \vec{\hat{v}}_o) \rightsquigarrow (\hat{\sigma}, \pi)$

(H2) $\mathsf{SAT}_{\theta,s}(\pi) \wedge \theta, s, \sigma \models \hat{\sigma}$

To prove

(G1) $\exists \sigma_\delta, \sigma_f.\, \sigma_\delta \# \sigma_f \wedge \sigma = \sigma_\delta \cdot \sigma_f \wedge \theta, s, \sigma_\delta \models_\Delta \langle\delta\rangle(\vec{\hat{v}}_i; \vec{\hat{v}}_o) \wedge \theta, s, \sigma_f \models \hat{\sigma}_f$

(H3) From the $\mathsf{produce}$ rule, we have $\hat{\sigma}_f = \bot$, $\delta = \mathsf{ex}$, $\vec{\hat{v}}_i = []$, $\vec{\hat{v}}_o = [\hat{x}]$, $\hat{\sigma} = \mathrm{ex}(\hat{x})$ and $\pi = []$.

(H4) From (H2) and the definition of $\models$ we have $\sigma = \mathrm{ex}(x)$ and $[\![\hat{x}]\!]_{\theta,s} = x$

(H5) Given (H4), we have $\sigma_\delta = \mathrm{ex}(x)$ and $\sigma_f = \bot$, such that $\sigma_\delta \# \sigma_f$ and $\sigma = \sigma_\delta \cdot \sigma_f$.

(H6) From (H4), we have $[\![\vec{\hat{v}}_i]\!]_{\theta,s} = []$ and $[\![\vec{\hat{v}}_o]\!]_{\theta,s} = [x]$, thus from (H5) and the definition of $\models_\Delta$ we have $\theta, s, \sigma_\delta \models_\Delta \langle\delta\rangle(\vec{\hat{v}}_i; \vec{\hat{v}}_o)$.

(H7) From (H3) and (H5) we have $\sigma_f = \bot$ and $\hat{\sigma}_f = \bot$, thus from the definition of $\models$, we have $\theta, s, \sigma_f \models \hat{\sigma}_f$. This with (H6) gives our goal (G1).

Proposition: Produce completeness
Assume

(H1) $\theta, s, \sigma_f \models \hat{\sigma}_f \wedge \theta, s, \sigma_\delta \models_\Delta \langle\delta\rangle(\vec{\hat{v}}_i; \vec{\hat{v}}_o) \wedge \sigma_f \# \sigma_\delta$

To prove

(G1) $\exists \hat{\sigma}.\, \mathsf{produce}(\hat{\sigma}_f, \delta, \vec{\hat{v}}_i, \vec{\hat{v}}_o) \rightsquigarrow (\hat{\sigma}, \pi) \wedge \mathsf{SAT}_{\theta,s}(\pi) \wedge \theta, s, (\sigma_f \cdot \sigma_\delta) \models \hat{\sigma}$

(H2) From (H9), $\sigma_\delta \models_\Delta \langle\delta\rangle(\vec{\hat{v}}_i; \vec{\hat{v}}_o)$, thus from the definition of $\models_\Delta$ we have $\sigma_\delta = \mathrm{ex}(x)$, $\delta = \mathsf{ex}$, $\vec{\hat{v}}_i = []$, $\vec{\hat{v}}_o = [\hat{x}]$ with $[\![\hat{x}]\!]_{\theta,s} = x$.

(H3) From (H9), $\sigma_f \# \sigma_\delta$, thus from (H2) and the rules of composition, we have $\sigma_f = \bot$.

(H4) From the rules of $\models$ and (H3), we have $\hat{\sigma}_f = \bot$.

(H5) We can pick $\hat{\sigma} = \mathrm{ex}(\hat{x})$.

(H6) From (H4), (H2), (H5) and the rules of $\mathsf{produce}$, we have $\mathsf{produce}(\sigma_f, \delta, \vec{\hat{v}}_i, \vec{\hat{v}}_o) \rightsquigarrow (\hat{\sigma}, \pi)$, with $\pi = []$.

(H7) From (H6), $\mathsf{SAT}_{\theta,s}(\pi)$.

(H8) From (H3) and (H2), $\sigma_f = \bot$ and $\sigma_\delta = \mathrm{ex}(x)$, thus from the rules of composition, $\sigma_f \cdot \sigma_\delta = \mathrm{ex}(x)$.

(H9) From (H2), $[\![\hat{x}]\!]_{\theta,s} = x$, thus from (H5) and (H8) we have $\theta, s, (\sigma_f \cdot \sigma_\delta) \models \hat{\sigma}$, which combined with (H6) and (H7) gives our goal (G1).

$\square$

1.2 Partial Map

1.2.1 Resource Algebra

$$
\begin{aligned}
\textsc{PMap}(I, \mathbb{S}) &\stackrel{\text{def}}{=} I \xrightarrow{\textit{fin}} \mathbb{S}.\Sigma \times \mathcal{P}(I)^? \\
(h, d) \cdot (h', d') &\stackrel{\text{def}}{=} (h'', d'') \\
\text{where } h'' &\stackrel{\text{def}}{=} \lambda i. \begin{cases} h(i) \cdot h'(i) & \text{if } i \in \operatorname{dom}(h) \cap \operatorname{dom}(h') \\ h(i) & \text{if } i \in \operatorname{dom}(h) \setminus \operatorname{dom}(h') \\ h'(i) & \text{if } i \in \operatorname{dom}(h') \setminus \operatorname{dom}(h) \\ \text{undefined} & \text{otherwise} \end{cases} \\
\text{and } d'' &\stackrel{\text{def}}{=} \begin{cases} d & \text{if } d' = \bot \\ d' & \text{if } d = \bot \\ \text{undefined} & \text{otherwise} \end{cases} \\
\text{and } d'' &= \bot \vee \operatorname{dom}(h'') \subseteq d'' \\
|(h, d)| &\stackrel{\text{def}}{=} \begin{cases} \bot & \text{if } \operatorname{dom}(h') = \emptyset \\ (h', \bot) & \text{otherwise} \end{cases} \\
\text{where } h' &\stackrel{\text{def}}{=} \lambda i. \begin{cases} |h(i)| & \text{if } i \in \operatorname{dom}(h) \wedge |h(i)| \neq \bot \\ \text{undefined} & \text{otherwise} \end{cases}
\end{aligned}
$$

For a wrapped state model $\mathbb{S} = \{\Sigma_{\mathbb{S}}, \mathcal{A}_{\mathbb{S}}, \Delta_{\mathbb{S}}\}$, we define the actions of $\textsc{PMap}(I, \mathbb{S})$ as $\mathcal{A} = \mathcal{A}_{\mathbb{S}} \uplus \{\texttt{alloc}\}$, and the predicates $\Delta = \Delta_{\mathbb{S}} \uplus \{\texttt{domainset}\}$.

We define predicate satisfiability as:

$$
\textsc{PMapPredSat}\ \frac{\sigma_i \models_\Delta \langle \delta \rangle(\vec{v}_i; \vec{v}_o)}{([i \mapsto \sigma_i], \bot) \models_\Delta \langle \delta \rangle(i :: \vec{v}_i; \vec{v}_o)}
\qquad
\textsc{PMapPredSatBot}\ \frac{\bot \models_\Delta \langle \delta \rangle(\vec{v}_i; \vec{v}_o)}{\bot \models_\Delta \langle \delta \rangle(i :: \vec{v}_i; \vec{v}_o)}
\qquad
\textsc{PMapPredDomainSet}\ (\emptyset, d) \models_\Delta \langle \texttt{domainset} \rangle(; d)
$$

We define symbolic interpretation as:

$$
\textsc{PMapSymInterpretation}\ \frac{\forall \hat{i} \in \operatorname{dom}(\hat{h}).\ [\![\hat{i}]\!]_{\theta,s} = i \wedge i \in \operatorname{dom}(h) \wedge \theta, s, h(i) \models \hat{h}(\hat{i}) \qquad [\![\operatorname{dom}(\hat{h})]\!]_{\theta,s} = \operatorname{dom}(h) \qquad [\![\hat{d}]\!]_{\theta,s} = d}{\theta, s, (h, d) \models (\hat{h}, \hat{d})}
$$

1.2.2 Compositional Concrete Rules

We define the helper functions `get` and `set` as:

$$
\begin{aligned}
\texttt{get} &: \textsc{PMap}(I, \mathbb{S}) \to I \to \mathbb{S}.\Sigma \\
\texttt{set} &: \textsc{PMap}(I, \mathbb{S}) \to I \to \mathbb{S}.\Sigma \to \textsc{PMap}(I, \mathbb{S})
\end{aligned}
$$

We pretty-print get and set as $\texttt{get}(\sigma, i) = \sigma_i$ and $\texttt{set}(\sigma, i, \sigma_i) = \sigma'$.

$$\text{Given } \mathit{wrap}(h, d) \stackrel{\text{def}}{=} \begin{cases} \bot & \text{if } \operatorname{dom}(h) = \emptyset \wedge d = \bot \\ (h, d) & \text{otherwise} \end{cases}$$

$$\mathit{unwrap}(\sigma) \stackrel{\text{def}}{=} \begin{cases} ([], \bot) & \text{if } \sigma = \bot \\ (h, d) & \text{if } \sigma = (h, d) \end{cases}$$

$$\textsc{CPMapGetMatch}\quad \frac{(h, d) = \mathit{unwrap}(\sigma) \qquad i \in \operatorname{dom}(h) \qquad \sigma_i = h(i)}{\texttt{get}(\sigma, i) = \sigma_i}$$

$$\textsc{CPMapGetAdd}\quad \frac{(h, d) = \mathit{unwrap}(\sigma) \qquad i \notin \operatorname{dom}(h) \qquad d \neq \bot \qquad i \in d}{\texttt{get}(\sigma, i) = \bot}$$

$$\textsc{CPMapGetBotDomain}\quad \frac{(h, d) = \mathit{unwrap}(\sigma) \qquad i \notin \operatorname{dom}(h) \qquad d = \bot}{\texttt{get}(\sigma, i) = \bot}$$

$$\textsc{CPMapSetSome}\quad \frac{(h, d) = \mathit{unwrap}(\sigma) \qquad \sigma_i \neq \bot \qquad h' = h[i \leftarrow \sigma_i] \qquad \sigma' = \mathit{wrap}(h', d)}{\texttt{set}(\sigma, i, \sigma_i) = \sigma'}$$

$$\textsc{CPMapSetNone}\quad \frac{(h, d) = \mathit{unwrap}(\sigma) \qquad \sigma_i = \bot \qquad h' = h[i \not\leftarrow] \qquad \sigma' = \mathit{wrap}(h', d)}{\texttt{set}(\sigma, i, \sigma_i) = \sigma'}$$

The action rules are then:

$$\textsc{CPMapAction}\quad \frac{\texttt{get}(\sigma, i) = \sigma_i \qquad \alpha(\sigma_i, \vec{v}_i) = (o, \sigma_i', \vec{v}_o) \qquad \texttt{set}(\sigma, i, \sigma_i') = \sigma'}{\alpha(\sigma, i :: \vec{v}_i) = (o, \sigma', i :: \vec{v}_o)}$$

$$\textsc{CPMapActionOutOfBounds}\quad \frac{d \neq \bot \qquad i \notin d}{\alpha((h, d), i :: \vec{v}_i) = (\texttt{Err}, (h, d), [])}$$

$$\textsc{CPMapAlloc}\quad \frac{d \neq \bot \qquad i = \texttt{fresh} \qquad \sigma_i = \texttt{instantiate}(\vec{v}_i) \qquad h' = h[i \leftarrow \sigma_i] \qquad d' = d \uplus \{i\}}{\texttt{alloc}((h, d), \vec{v}_i) = (\texttt{Ok}, (h', d'), [i])}$$

$$\textsc{CPMapAllocMiss}\quad \frac{(h, d) = \mathit{unwrap}(\sigma) \qquad d = \bot}{\texttt{alloc}(\sigma, \vec{v}_i) = (\texttt{Miss}, \sigma, [\text{`}\texttt{domainset}\text{'}])}$$

1.2.3 Compositional Symbolic Rules

We now re-define get and set, lifting them to the symbolic realm (note set is unchanged, as it does not perform any sort of matching). get now returns a state and an index, that may be different from the input index – it corresponds to the actual index of the state in the map. For example, for a map $[1 \mapsto \hat{x}]$, calling get with index $\hat{y}$ may return index 1, along with the state $\hat{x}$ and the path condition $[\hat{y} = 1]$.

$$\texttt{get} : \text{PM}{\small\text{AP}}(I, \mathbb{S}) \to I \to \mathcal{P}(I \times \mathbb{S}.\hat{\Sigma} \times \Pi)$$
$$\texttt{set} : \text{PM}{\small\text{AP}}(I, \mathbb{S}) \to I \to \mathbb{S}.\hat{\Sigma} \to \text{PM}{\small\text{AP}}(I, \mathbb{S})$$

We pretty-print $\texttt{get}$ and $\texttt{set}$ as $\texttt{get}(\hat{\sigma}, \hat{i}) \rightsquigarrow (\hat{i}', \hat{\sigma}_i, \pi)$ and $\texttt{set}(\hat{\sigma}, \hat{i}, \hat{\sigma}_i) = \hat{\sigma}'$.

$$\text{Given } \mathit{wrap}(\hat{h}, \hat{d}) \stackrel{\text{def}}{=} \begin{cases} \bot & \text{if } \operatorname{dom}(\hat{h}) = \emptyset \wedge \hat{d} = \bot \\ (\hat{h}, \hat{d}) & \text{otherwise} \end{cases}$$

$$\mathit{unwrap}(\hat{\sigma}) \stackrel{\text{def}}{=} \begin{cases} ([], \bot) & \text{if } \hat{\sigma} = \bot \\ (\hat{h}, \hat{d}) & \text{if } \hat{\sigma} = (\hat{h}, \hat{d}) \end{cases}$$

$$\frac{(\hat{h}, \hat{d}) = \mathit{unwrap}(\hat{\sigma}) \qquad \hat{i}' \in \operatorname{dom}(\hat{h}) \qquad \hat{\sigma}_i = \hat{h}(\hat{i}')}{\texttt{get}(\hat{\sigma}, \hat{i}) \rightsquigarrow (\hat{i}', \hat{\sigma}_i, [\hat{i} = \hat{i}'])}\ \text{PM}{\small\text{AP}}\text{G}{\small\text{ET}}\text{M}{\small\text{ATCH}}$$

$$\frac{(\hat{h}, \hat{d}) = \mathit{unwrap}(\hat{\sigma}) \qquad \hat{i} \notin \operatorname{dom}(\hat{h}) \qquad \hat{d} \neq \bot}{\texttt{get}(\hat{\sigma}, \hat{i}) \rightsquigarrow (\hat{i}, \bot, [\hat{i} \notin \operatorname{dom}(\hat{h}) \wedge \hat{i} \in \hat{d}])}\ \text{PM}{\small\text{AP}}\text{G}{\small\text{ET}}\text{A}{\small\text{DD}}$$

$$\frac{(\hat{h}, \hat{d}) = \mathit{unwrap}(\hat{\sigma}) \qquad \hat{i} \notin \operatorname{dom}(\hat{h}) \qquad \hat{d} = \bot}{\texttt{get}(\hat{\sigma}, \hat{i}) \rightsquigarrow (\hat{i}, \bot, [\hat{i} \notin \operatorname{dom}(\hat{h})])}\ \text{PM}{\small\text{AP}}\text{G}{\small\text{ET}}\text{B}{\small\text{OT}}\text{D}{\small\text{OMAIN}}$$

$$\frac{(\hat{h}, \hat{d}) = \mathit{unwrap}(\hat{\sigma}) \qquad \hat{\sigma}_i \neq \bot \qquad \hat{h}' = \hat{h}[\hat{i} \leftarrow \hat{\sigma}_i] \qquad \hat{\sigma}' = \mathit{wrap}(\hat{h}', \hat{d})}{\texttt{set}(\hat{\sigma}, \hat{i}, \hat{\sigma}_i) = \hat{\sigma}'}\ \text{PM}{\small\text{AP}}\text{S}{\small\text{ET}}\text{S}{\small\text{OME}}$$

$$\frac{(\hat{h}, \hat{d}) = \mathit{unwrap}(\hat{\sigma}) \qquad \hat{\sigma}_i = \bot \qquad \hat{h}' = \hat{h}[\hat{i} \not\leftarrow] \qquad \hat{\sigma}' = \mathit{wrap}(\hat{h}', \hat{d})}{\texttt{set}(\hat{\sigma}, \hat{i}, \hat{\sigma}_i) = \hat{\sigma}'}\ \text{PM}{\small\text{AP}}\text{S}{\small\text{ET}}\text{N}{\small\text{ONE}}$$

$$\text{Given } \mathit{lift\_if\_miss}(o, \hat{i}, \vec{\hat{v}}_i) \stackrel{\text{def}}{=} \begin{cases} \hat{i} :: \vec{\hat{v}}_i & \text{if } o = \texttt{Miss} \\ \vec{\hat{v}}_i & \text{otherwise} \end{cases}$$

$$\frac{\texttt{get}(\hat{\sigma}, \hat{i}) \rightsquigarrow (\hat{i}', \hat{\sigma}_i, \pi) \qquad \alpha(\hat{\sigma}_i, \vec{\hat{v}}_i) \rightsquigarrow (o, \hat{\sigma}_i', \vec{\hat{v}}_o, \pi') \qquad \texttt{set}(\hat{\sigma}, \hat{i}', \hat{\sigma}_i') = \hat{\sigma}'}{\alpha(\hat{\sigma}, \hat{i} :: \vec{\hat{v}}_i) \rightsquigarrow (o, \hat{\sigma}', \hat{i}' :: \vec{\hat{v}}_o, \pi :: \pi')}\ \text{PM}{\small\text{AP}}\text{A}{\small\text{CTION}}$$

$$\frac{\hat{d} \neq \bot}{\alpha((\hat{h}, \hat{d}), \hat{i} :: \vec{\hat{v}}_i) \rightsquigarrow (\texttt{Err}, (\hat{h}, \hat{d}), [], [\hat{i} \notin \hat{d}])}\ \text{PM}{\small\text{AP}}\text{A}{\small\text{CTION}}\text{O}{\small\text{UT}}\text{O}{\small\text{F}}\text{B}{\small\text{OUNDS}}$$

$$\frac{\hat{d} \neq \bot \qquad \hat{i} = \texttt{fresh} \qquad \hat{\sigma}_i = \texttt{instantiate}(\vec{\hat{v}}_i) \qquad \hat{h}' = \hat{h}[\hat{i} \leftarrow \hat{\sigma}_i] \qquad \hat{d}' = \hat{d} \uplus \{\hat{i}\}}{\texttt{alloc}((\hat{h}, \hat{d}), \vec{\hat{v}}_i) \rightsquigarrow (\texttt{Ok}, (\hat{h}', \hat{d}'), [\hat{i}], [\hat{i} = \hat{i}])}\ \text{PM}{\small\text{AP}}\text{A}{\small\text{LLOC}}$$

$$\frac{(\hat{h}, \hat{d}) = \mathit{unwrap}(\hat{\sigma}) \qquad \hat{d} = \bot}{\texttt{alloc}(\hat{\sigma}, \vec{\hat{v}}_i) \rightsquigarrow (\texttt{Miss}, \hat{\sigma}, [\text{‘}\texttt{domainset}\text{’}], [])}\ \text{PM}{\small\text{AP}}\text{A}{\small\text{LLOC}}\text{M}{\small\text{ISS}}$$

PMapCons

$$\frac{\begin{array}{c}\mathsf{get}(\hat\sigma,\hat\imath) \rightsquigarrow (\hat\imath',\hat\sigma_i,\pi)\\ \mathsf{consume}(\hat\sigma_i,\delta,\vec{\hat v}_i) \rightsquigarrow (o,\hat\sigma_i',\vec{\hat v}_o,\pi') \qquad \mathsf{set}(\hat\sigma,\hat\imath',\hat\sigma_i') = \hat\sigma' \qquad \vec{\hat v}_o' = \mathit{lift\_if\_miss}(o,\hat\imath',\vec{\hat v}_o)\end{array}}{\mathsf{consume}(\hat\sigma,\delta,\hat\imath :: \vec{\hat v}_i) \rightsquigarrow (o,\hat\sigma',\vec{\hat v}_o',\pi :: \pi')}$$

PMapConsIncompat

$$\frac{\hat d \neq \bot}{\mathsf{consume}((\hat h,\hat d),\delta,\hat\imath :: \vec{\hat v}_i) \rightsquigarrow (\mathsf{LFail},(\hat h,\hat d),[],[\hat\imath \notin \hat d])}$$

PMapConsDomainSet

$$\frac{\hat d \neq \bot}{\mathsf{consume}((\hat h,\hat d),\mathsf{domainset},[]) \rightsquigarrow (\mathsf{Ok},(\hat h,\bot),[\hat d],[])}$$

PMapConsDomainSetMiss

$$\frac{(\hat h,\hat d) = \mathit{unwrap}(\hat\sigma) \qquad \hat d = \bot}{\mathsf{consume}(\hat\sigma,\mathsf{domainset},[]) \rightsquigarrow (\mathsf{Miss},\hat\sigma,[\text{‘}\mathsf{domainset}\text{’}],[])}$$

PMapProd

$$\frac{\mathsf{get}(\hat\sigma,\hat\imath) \rightsquigarrow (\hat\imath',\hat\sigma_i,\pi) \qquad \mathsf{produce}(\hat\sigma_i,\delta,\vec{\hat v}_i,\vec{\hat v}_o) \rightsquigarrow (\hat\sigma_i',\pi') \qquad \mathsf{set}(\hat\sigma,\hat\imath',\hat\sigma_i') = \hat\sigma'}{\mathsf{produce}(\hat\sigma,\delta,\hat\imath :: \vec{\hat v}_i,\vec{\hat v}_o) \rightsquigarrow (\hat\sigma',\pi :: \pi')}$$

PMapProdDomainSet

$$\frac{(\hat h,\bot) = \mathit{unwrap}(\hat\sigma)}{\mathsf{produce}(\hat\sigma,\mathsf{domainset},[],[\hat d]) \rightsquigarrow ((\hat h,\hat d),[\mathrm{dom}(\hat h) \subseteq \hat d])}$$

PMapFix

$$\frac{\mathbb{S}.\mathsf{fix}\ \vec{\hat v}_i = a \qquad a' = \mathit{lift}(a,\hat\imath)}{\mathsf{fix}\ \hat\imath :: \vec{\hat v}_i = a'}$$

PMapFixDomainSet

$$\mathsf{fix}\ [\text{‘}\mathsf{domainset}\text{’}] = \exists \hat d.\ \langle\mathsf{domainset}\rangle(;\hat d)$$

Note in the above we define $\mathit{lift}(a,\hat\imath)$ as a function that traverses an assertion a and lifts all core predicate assertions by adding the value $\hat\imath$ at the start of its in-values, such that $\mathit{lift}(\langle\delta\rangle(\vec{\hat v}_i;\vec{\hat v}_o),\hat\imath) = \langle\delta\rangle(\hat\imath :: \vec{\hat v}_i;\vec{\hat v}_o)$.

1.2.4 Soundness Proofs

To facilitate the soundness proof for PMap, and due to its extensive use of the `get` and `set` helper methods to modify the elements in the heap, we first define and prove axioms about these auxiliary functions.

We introduce a pair of axioms for `get`, similar to OX and UX soundness of actions, to ensure that the symbolic function retrieves the right state if it exists in the concrete counterpart of the state.

$$\begin{array}{c}\theta,s,\sigma \models \hat\sigma \land \mathsf{get}(\sigma,i) = \sigma_i \land [\![\hat\imath]\!]_{\theta,s} = i \implies \exists \hat\sigma_i,\hat\imath',\pi.\\ \mathsf{get}(\hat\sigma,\hat\imath) \rightsquigarrow (\hat\imath',\hat\sigma_i,\pi) \land \theta,s,\sigma_i \models \hat\sigma_i \land [\![\hat\imath']\!]_{\theta,s} = i \land \mathsf{SAT}_{\theta,s}(\pi)\end{array} \tag{Get OX Soundness}$$

$$\begin{array}{c}\mathsf{get}(\hat\sigma,\hat\imath) \rightsquigarrow (\hat\imath',\hat\sigma_i,\pi) \land [\![\hat\imath]\!]_{\theta,s} = [\![\hat\imath']\!]_{\theta,s} = i \land \mathsf{SAT}_{\theta,s}(\pi) \implies\\ \forall\sigma.\,\theta,s,\sigma \models \hat\sigma \implies \exists\sigma_i.\,\mathsf{get}(\sigma,i) = \sigma_i \land \theta,s,\sigma_i \models \hat\sigma_i\end{array} \tag{Get UX Soundness}$$

Proof.

Proposition: Get OX Soundness

Assume

(H1) $\theta, s, \sigma \models \hat{\sigma} \wedge \mathtt{get}(\sigma, i) = \sigma_i \wedge [\![\hat{i}]\!]_{\theta,s} = i$

To prove

(G1) $\exists \hat{\sigma}_i, \hat{i}', \pi.\, \mathtt{get}(\hat{\sigma}, \hat{i}) \rightsquigarrow (\hat{i}', \hat{\sigma}_i, \pi) \wedge \theta, s, \sigma_i \models \hat{\sigma}_i \wedge [\![\hat{i}']\!]_{\theta,s} = i \wedge \mathsf{SAT}_{\theta,s}(\pi)$

We proceed by proving the property holds for all rules of the concrete `get`, resulting in three cases.

Case CPMapGetMatch**:**

(H2) Assume $(h, d) = \mathit{unwrap}(\sigma) \wedge i \in \mathrm{dom}(h) \wedge \sigma_i = h(i)$.

(H3) It follows from (H1) and the definition of $\models$ that $\hat{\sigma} = (\hat{h}, \hat{d})$ such that $\exists \hat{i}'.\, \hat{i}' \in \mathrm{dom}(\hat{h})$, $[\![\hat{i}']\!]_{\theta,s} = i$ and $\theta, s, \sigma_i \models \hat{h}(\hat{i}')$.

(H4) We can then apply PMapGetMatch, giving us $\mathtt{get}(\hat{\sigma}, \hat{i}) \rightsquigarrow (\hat{i}', \hat{h}(\hat{i}'), [\hat{i}' = \hat{i}])$.

(H5) From (H3) and (H1) it follows that $\mathsf{SAT}_{\theta,s}([\hat{i}' = \hat{i}])$, which completes our goal (G1).

Case CPMapGetAdd**:**

(H6) Assume $(h, d) = \mathit{unwrap}(\sigma) \wedge i \notin \mathrm{dom}(h) \wedge d \neq \perp \wedge i \in d$.

(H7) It follows from (H1) and the definition of $\models$ that $\hat{\sigma} = (\hat{h}, \hat{d})$ such that $\hat{i} \in \hat{d}$ and $\hat{i} \notin \mathrm{dom}(\hat{h})$.

(H8) We can then apply PMapGetAdd, giving us $\mathtt{get}(\hat{\sigma}, \hat{i}) \rightsquigarrow (\hat{i}, \perp, [\hat{i} \notin \mathrm{dom}(\hat{h}) \wedge \hat{i} \notin \hat{d}])$.

(H9) From (H8) and (H7) it follows that $\mathsf{SAT}_{\theta,s}([\hat{i} \notin \mathrm{dom}(\hat{h}) \wedge \hat{i} \notin \hat{d}])$, which completes our goal (G1).

Case CPMapGetBotDomain**:**

(H10) Assume $(h, d) = \mathit{unwrap}(\sigma) \wedge i \notin \mathrm{dom}(h) \wedge d = \perp$.

(H11) From (H10), (H1) and CPMapGetBotDomain, we have $\sigma_i = \perp$.

(H12) It follows from (H1) and the definition of $\models$ that $\hat{\sigma} = (\hat{h}, \perp)$ such that $\hat{i} \notin \mathrm{dom}(\hat{h})$.

(H13) We can then apply PMapGetBotDomain, giving us $\mathtt{get}(\hat{\sigma}, \hat{i}) \rightsquigarrow (\hat{i}, \perp, [\hat{i} \notin \mathrm{dom}(\hat{h})])$.

(H14) From (H13) and (H12) it follows that $\mathsf{SAT}_{\theta,s}([\hat{i} \notin \mathrm{dom}(\hat{h})])$, which completes our goal (G1).

Proposition: Get UX Soundness
Assume

(H1) $\mathtt{get}(\hat{\sigma}, \hat{i}) \rightsquigarrow (\hat{i}', \hat{\sigma}_i, \pi) \wedge [\![\hat{i}]\!]_{\theta,s} = [\![\hat{i}']\!]_{\theta,s} = i \wedge \mathsf{SAT}_{\theta,s}(\pi)$

(H2) $\theta, s, \sigma \models \hat{\sigma}$

To prove

(G1) $\exists \sigma_i.\, \mathtt{get}(\sigma, i) = \sigma_i \wedge \theta, s, \sigma_i \models \hat{\sigma}_i$

We proceed by proving the property holds for all rules of the symbolic `get`, resulting in three cases.

Case PMapGetMatch**:**

(H3) Assume $(\hat{h}, \hat{d}) = \mathit{unwrap}(\hat{\sigma}) \land \hat{i}' \in \mathrm{dom}(\hat{h}) \land \hat{\sigma}_i = \hat{h}(\hat{i}')$

(H4) From (H3), (H1) and PMapGetMatch, we have $\pi = [\hat{i} = \hat{i}']$

(H5) From (H3), we have $\hat{\sigma} \neq \bot$, thus from (H2) we have $\sigma = (h, d)$

(H6) From (H2), (H3) and the definition of $\models$, we have $i \in h$ such that $\sigma_i = h(i)$ and $\theta, s, \sigma_i \models \hat{\sigma}_i$

(H7) From (H6) we can apply CPMapGetMatch, thus $\mathtt{get}(\sigma, i) = \sigma_i$. This completes our goal (G1).

Case PMapGetAdd**:**

(H8) Assume $(\hat{h}, \hat{d}) = \mathit{unwrap}(\hat{\sigma}) \land \hat{i} \notin \mathrm{dom}(\hat{h}) \land \hat{d} \neq \bot$

(H9) From (H8), (H1) and PMapGetAdd, we have $\hat{i}' = \hat{i}$, $\hat{\sigma}_i = \bot$ and $\pi = [\hat{i} \notin \mathrm{dom}(\hat{h}) \land \hat{i} \in \hat{d}]$

(H10) From (H8) we have $\hat{d} \neq \bot$, thus $\hat{\sigma} = (\hat{h}, \hat{d})$ and from (H2) $\sigma = (h, d)$ such that $[\![\hat{d}]\!]_{\theta,s} = d$.

(H11) From (H1) $\mathsf{SAT}_{\theta,s}(\pi)$, thus from (H9) and (H2), $i \notin \mathrm{dom}(h) \land i \in d$

(H12) From (H11) and (H10) we can apply CPMapGetAdd, thus $\mathtt{get}(\sigma, i) = \bot$.

(H13) From **??**, $\theta, s, \bot \models \bot$, which along with (H12) completes our goal (G1).

Case PMapGetBotDomain**:**

(H14) Assume $(\hat{h}, \hat{d}) = \mathit{unwrap}(\hat{\sigma}) \land \hat{i} \notin \mathrm{dom}(\hat{h}) \land \hat{d} = \bot$

(H15) From (H14), (H1) and PMapGetBotDomain, we have $\hat{i}' = \hat{i}$, $\hat{\sigma}_i = \bot$ and $\pi = [\hat{i} \notin \mathrm{dom}(\hat{h})]$

(H16) From (H14) we have $\hat{d} = \bot$, so given $(h, d) = \mathit{unwrap}(\sigma)$ and (H2), we have $d = \bot$.

(H17) From (H1) $\mathsf{SAT}_{\theta,s}(\pi)$, thus from (H15) and (H2), $i \notin \mathrm{dom}(h)$.

(H18) From (H17) and (H16) we can apply CPMapGetBotDomain, thus $\mathtt{get}(\sigma, i) = \bot$.

(H19) From **??**, $\theta, s, \bot \models \bot$, which along with (H18) completes our goal (G1).

□

We can now proceed with the standard proofs.
Proof.

Proposition: OX Soudness
Assume

(H1) $\theta, s, \sigma \models \hat{\sigma} \land \alpha(\sigma, \vec{v}_i) = (o, \sigma', \vec{v}_o) \land [\![\vec{\hat{v}}_i]\!]_{\theta,s} = \vec{v}_i$

(H2) $\forall o', \hat{\sigma}', \vec{\hat{v}}'_o, \pi'. \, \alpha(\hat{\sigma}, \vec{\hat{v}}_i) \rightsquigarrow (o', \hat{\sigma}', \vec{\hat{v}}'_o, \pi') \Rightarrow o' \in \{\mathtt{Ok}, \mathtt{Err}\}$

(H3) The actions on $\mathbb{S}$ are OX sound.

To prove

(G1) $\exists \hat{\sigma}', \vec{\hat{v}}_o, \pi, \theta'. \, \hat{\alpha}(\hat{\sigma}, \vec{\hat{v}}_i) \rightsquigarrow (o, \hat{\sigma}', \vec{\hat{v}}_o, \pi) \wedge \theta', s, \sigma' \models \hat{\sigma}' \wedge \mathsf{SAT}_{\theta',s}(\pi) \wedge [\![\vec{\hat{v}}_o]\!]_{\theta',s} = \vec{v}_o$

There are two action cases to consider: $\alpha \in \mathcal{A}_{\mathbb{S}}$ and $\alpha = \mathtt{alloc}$.

Case $\alpha \in \mathcal{A}_{\mathbb{S}}$**:**

(H4) If the action gets executed succesfully (it is not an out of bounds error), from CPMAPACTION we have $\mathtt{get}(\sigma, i) = \sigma_i$, $\alpha(\sigma_i, \vec{v}'_i) = (o, \sigma'_i, \vec{v}'_o)$ and $\mathtt{set}(\sigma, i, \sigma'_i) = \sigma'$, with $\vec{v}_i = i :: \vec{v}'_i$ and $\vec{v}_o = i :: \vec{v}'_o$

(H5) From (H1) and (H4), we know we have $\vec{\hat{v}}_i = \hat{i} :: \vec{\hat{v}}'_i$ such that $[\![\hat{i}]\!]_{\theta,s} = i$ and $[\![\vec{\hat{v}}'_i]\!]_{\theta,s} = \vec{\hat{v}}_i$.

(H6) From (H1), (H4) and (H5), we can apply Get OX Soundness, giving $\exists \hat{\sigma}_i, \hat{i}', \pi. \, \mathtt{get}(\hat{\sigma}, \hat{i}) \rightsquigarrow (\hat{i}', \hat{\sigma}_i, \pi) \wedge \theta, s, \sigma_i \models \hat{\sigma}_i \wedge [\![\hat{i}']\!]_{\theta,s} = i \wedge \mathsf{SAT}_{\theta,s}(\pi)$

(H7) From (H4), (H6), (H2), (H5) and (H3), we can apply **??**, giving us $\exists \vec{\hat{v}}'_o, \hat{\sigma}'_i, \pi', \theta'. \, \alpha(\hat{\sigma}_i, \vec{\hat{v}}'_i) \rightsquigarrow (o, \hat{\sigma}'_i, \vec{\hat{v}}'_o, \pi') \wedge \theta', s, \sigma'_i \models \hat{\sigma}'_i \wedge \mathsf{SAT}_{\theta',s}(\pi') \wedge [\![\vec{\hat{v}}'_o]\!]_{\theta',s} = \vec{v}'_o$.

(H8) From (H1) and (H7), we have $\theta, s, \sigma \models \hat{\sigma}$ and $\theta', s, \sigma'_i \models \hat{\sigma}'_i$. From the definition of $\mathtt{set}$, it follows that only modifying the state at $\hat{i}'$ preserves symbolic interpretation, thus given $\mathtt{set}(\hat{\sigma}, \hat{i}', \hat{\sigma}'_i) = \hat{\sigma}'$, we have $\theta', s, \sigma' \models \hat{\sigma}'$.

(H9) By applying PMAPACTION and from (H7), (H6), (H8) our goal (G1) follows.

(H10) If this is an out of bounds access, we have $\sigma = \sigma' = (h, d)$ with $d \neq \bot$ and $\vec{v}_i = i :: \vec{v}'_i$, with $i \notin d$, as well as $\vec{\hat{v}}_i = \hat{i} :: \vec{\hat{v}}'_i$

(H11) From (H1) and the definition of $\models$, we have $\hat{\sigma} = (\hat{h}, \hat{d})$ such that $\pi = [\hat{i} \notin \hat{d}]$ and $\mathsf{SAT}_{\theta,s}(\pi)$.

(H12) From the rules of action execution and (H11), we get $\alpha(\hat{\sigma}, \vec{\hat{v}}_i) \rightsquigarrow (\mathtt{Err}, \hat{\sigma}, [], \pi)$. The rest of our goal (G1) follows.

Case $\alpha = \mathtt{alloc}$**:**
Both cases (when $d \neq \bot$ and $d = \bot$) are trivially lifted from the concrete to the symbolic realm, with $\pi = [i = i]$ and $\pi = []$ respectively. It follows that $\mathsf{SAT}_{\theta,s}(\pi)$ in both cases, giving our goal (G1).
For the successful allocation case, we note there is an additional requirement for soundness of state instantiation to be able to prove the resulting states $\hat{\sigma}'$ and σ' are compatible.

$$[\![\vec{\hat{v}}_i]\!]_{\theta,s} = \vec{v}_i \implies (\forall \sigma, \hat{\sigma}. \, \sigma = \mathtt{instantiate} \; \vec{v}_i \wedge \hat{\sigma} = \mathtt{instantiate} \; \vec{\hat{v}}_i \implies \theta, s, \sigma \models \hat{\sigma})$$

Proposition: UX Soundness
Assume

(H1) $\hat{\alpha}(\hat{\sigma}, \vec{\hat{v}}_i) \rightsquigarrow (o, \hat{\sigma}', \vec{\hat{v}}_o, \pi) \wedge \mathsf{SAT}_{\theta,s}(\pi) \wedge \theta, s, \sigma' \models \hat{\sigma}' \wedge$
$[\![\vec{\hat{v}}_o]\!]_{\hat{s},\pi} \rightsquigarrow (\vec{v}_o, \pi') \wedge [\![\vec{\hat{v}}_i]\!]_{\hat{s},\pi'} \rightsquigarrow (\vec{v}_i, \pi'')$

(H2) The actions on $\mathbb{S}$ are UX sound.

To prove

(G1) $\exists \sigma. \theta, s, \sigma \models \hat{\sigma} \wedge \alpha(\sigma, \vec{v}_i) = (o, \sigma', \vec{v}_o)$

Case $\alpha \in \mathcal{A}_{\mathbb{S}}$**:**

(H3) If the action gets executed succesfully (it is not an out of bounds error), from PMAPACTION we have $\texttt{get}(\hat{\sigma}, \hat{i}) \rightsquigarrow (\hat{i}', \hat{\sigma}_i, \pi')$, $\alpha(\hat{\sigma}_i, \vec{\hat{v}}_i') \rightsquigarrow (o, \hat{\sigma}_i', \vec{\hat{v}}_o', \pi'')$ and $\texttt{set}(\hat{\sigma}, \hat{i}', \hat{\sigma}_i') = \hat{\sigma}'$, with $\vec{\hat{v}}_i = \hat{i} :: \vec{\hat{v}}_i'$, $\vec{\hat{v}}_o = \hat{i}' :: \vec{\hat{v}}_o'$ and $\pi = \pi' :: \pi''$

(H4) From (H2), (H3) and (H1), we get $\exists \sigma_i. \theta, s, \sigma_i \models \hat{\sigma}_i \wedge \alpha(\sigma_i, \vec{v}_i') = (p, \sigma', \vec{v}_o')$

(H5) From the definition of $\models$, it follows that if $\theta, s, \sigma' \models \hat{\sigma}'$ where $\texttt{set}(\hat{\sigma}, \hat{i}', \hat{\sigma}_i') = \hat{\sigma}'$, then if $\theta, s, \sigma_i \models \hat{\sigma}_i$ there exists an σ and i such that $\texttt{set}(\sigma, i, \sigma_i') = \sigma'$, giving us $\theta, s, \sigma \models \hat{\sigma}$

(H6) From (H3), (H1) and (H5) we can use Get UX Soundness, giving us $\exists \sigma_i. \texttt{get}(\sigma, i) = \sigma_i \wedge \theta, s, \sigma_i \models \hat{\sigma}_i$

(H7) From (H6), (H4) and (H5) we can apply CPMAPACTION, giving us $\alpha(\sigma, \vec{v}_i) = (o, \sigma', \vec{v}_o)$, which together with (H5) gives our goal (G1).

(H8) If the action fails as it is out of bounds, from PMAPACTIONOUTOFBOUNDS we have $\vec{\hat{v}}_i = \hat{i} :: \vec{\hat{v}}_i'$ and $\pi = [\hat{i} \notin \hat{d}]$.

(H9) From (H1) and the definition of $\models$ we thus have $\sigma = (h, d)$, with $\vec{v}_i = i :: \vec{v}_i'$, $d \neq \bot$ and $i \notin d$

(H10) From (H9), we can apply CPMAPACTIONOUTOFBOUNDS, which along with the fact the state is unmodified gives our goal (G1).

Case $\alpha = \texttt{alloc}$**:**
Again, both cases (when the domains set d is owned and when it is $\bot$) are directly lifted from the concrete cases; it follows that if $\theta, s, \sigma' \models \hat{\sigma}'$ then $\theta, s, \sigma \models \hat{\sigma}$ where σ is the state without the added binding.

Proposition: Frame subtraction is satisfied
Assume

(H1) $\sigma \# \sigma_f \wedge \alpha(\sigma \cdot \sigma_f, \vec{v}_i) = (o, \sigma', \vec{v}_o)$

(H2) The actions on $\mathbb{S}$ satisfy frame subtraction.

(H3) $\forall \alpha, \vec{v}_i, \vec{v}_o. \alpha(\bot, \vec{v}_i) = (\texttt{Miss}, \sigma', \vec{v}_o)$

To prove

(G1) $$\begin{aligned} &\exists \sigma'', o', \vec{v}_o'. \alpha(\sigma, \vec{v}_i) = (o', \sigma'', \vec{v}_o') \wedge \\ &\quad (o' \neq \texttt{Miss} \implies o' = o \wedge \vec{v}_o' = \vec{v}_o \wedge \sigma' = \sigma'' \cdot \sigma_f) \end{aligned}$$

Case $\alpha \in \mathcal{A}_{\mathbb{S}}$**:**

(H4) If the action causes an out of bounds error, from CPMAPACTIONOUTOFBOUNDS we have $\sigma \cdot \sigma_f = (h, d)$ and $\vec{v}_i = i :: \vec{v}_i'$ such that $i \notin d$.

(H5) From composition, the domain set d is either part of σ or σ_f. If it is part of σ such that $\sigma = (h_a, d)$ and $\sigma_f = (h_b, \bot)$, then we still have $i \notin d$, resulting in the same error and outcome, giving our goal (G1).

(H6) If the domain is in σ_f such that $\sigma = (h_a, \bot)$ and $\sigma_f = (h_b, d)$, then the action is executed on the underlying state model. Because we have $\mathrm{dom}(h) \subseteq d$, from (H4) we have $i \notin \mathrm{dom}(h_a)$, so from the rules of `get` and CPMapAction the action is executed on $\bot$. From (H3), we have $o' = \texttt{Miss}$, satisfying (G1).

(H7) If the action is not out of bounds, we have $\alpha(\sigma_i, \vec{v}_i') = (o, \sigma_i', \vec{v}_o')$ with $\vec{v}_i = i :: \vec{v}_i'$ and $\vec{v}_o = i :: \vec{v}_o'$.

(H8) If σ_f only includes the domain set and indices different than i, then the action is executed on the same σ_i and gives the same outcomes – it can then be composed with the result, giving our goal (G1).

(H9) Let σ_f such that $\sigma_f = (h_b, d_b)$ and $i \in \mathrm{dom}(h_b)$. We denote $\sigma_{b,i} = h_b(i)$. For $\sigma = (h_a, d_a)$, we also have $\sigma_{a,i} = \bot$ if $i \notin \mathrm{dom}(h_a)$, and $\sigma_{a,i} = h_a(i)$ otherwise, such that $\sigma_{a,i} \# \sigma_{b,i}$.

(H10) With (H7) and (H9), from (H2), we can apply **??** to the action on the state at index i, which allows re-applying CPMapAction and completes our goal (G1).

Case $\alpha = \texttt{alloc}$**:**

(H11) `alloc` is always defined, so we know $\exists \sigma'', o', \vec{v}_o.\, \alpha(\sigma, \vec{v}_i) = (o', \sigma'', \vec{v}_o')$.

(H12) Assume $o' \neq \texttt{Miss}$.

(H13) From (H12) and the rules of `alloc`, $\sigma = (h, d)$ with $d \neq \bot$.

(H14) From (H13) and (H1), if $\sigma \# \sigma_f$, then $\sigma_f = (h_f, \bot)$.

(H15) From the definition of `alloc`, the heap is not modified to the exception of the added entry, so from the definition of composition we have $\sigma' = \sigma'' \cdot \sigma_f$. The returns value $\vec{v}_o$ are the same (a fresh value), and the outcome is `Ok` in both cases, giving our goal (G1).

Proposition: Frame addition is satisfied
Assume

(H1) $\alpha(\sigma, \vec{v}_i) = (o, \sigma', \vec{v}_o)$

(H2) $\sigma_f \# \sigma'$

(H3) $o \neq \texttt{Miss}$

(H4) The actions on $\mathbb{S}$ satisfy frame addition.

(H5) $\forall \alpha, \vec{v}_i, \vec{v}_o.\, \alpha(\bot, \vec{v}_i) = (\texttt{Miss}, \sigma', \vec{v}_o)$

To prove

(G1) $\sigma \# \sigma_f \wedge \alpha(\sigma \cdot \sigma_f, \vec{v}_i) = (o, \sigma' \cdot \sigma_f, \vec{v}_o)$

Case $\alpha \in \mathcal{A}_{\mathbb{S}}$**:**

(H6) From the rules of concrete actions we have $\vec{v}_i = i :: \vec{v}'_i$ and $\vec{v}_o = i :: \vec{v}'_o$.

(H7) If the execution of the action is an out of bounds error, by CPMapActionOutOfBounds we have $\sigma = (h, d)$ such that $d \neq \bot$ and $i \notin d$.

(H8) From composition rules, we know $\sigma_f = (h_f, \bot)$, meaning that we still get $\alpha(\sigma \cdot \sigma_f, \vec{v}_i) = (\mathsf{Err}, \sigma' \cdot \sigma_f, [])$ as we still have $i \notin d$. This satisfies our goal (G1).

(H9) We now look at the case where we do not have an out of bounds error. Assume we have $\sigma = (h, d)$ such that $i \notin \mathrm{dom}(h)$. From the rules for `get`, we know we executed the action on $\bot$. From (H5), we thus have $o = \mathsf{Miss}$, which is a contradiction with (H3). Thus we know $i \in \mathrm{dom}(h)$.

(H10) From (H9) and the rules for `get`, we know we have $\sigma_i = h(i)$ and $\alpha(\sigma_i, \vec{v}'_i) = (o, \sigma'_i, \vec{v}'_o)$.

(H11) If we have $\sigma_f = (h_f, d_f)$ such that $i \notin \mathrm{dom}(h_f)$, then it follows from (H2) that $\sigma \# \sigma_f$ as the action does not modify any other cell. The action will also have the same outcome, completing our goal (G1).

(H12) If we have $\sigma_f = (h_f, d_f)$ such that $i \in \mathrm{dom}(h_f)$, it follows from (H2) that $\sigma'_i \# h_f(i)$. We also know the outcome is not `Miss` from the rules of action execution and (H3).

(H13) From (H12), (H10) and (H4) we can apply **??** and get $\sigma_i \# h_f(i)$ and $\alpha(\sigma_i \cdot h_f(i), \vec{v}'_i) = (o, \sigma'_i \cdot h_f(i), \vec{v}'_o)$.

(H14) From (H13) and CPMapAction, we get $\sigma \# \sigma_f$ and $\alpha(\sigma \cdot \sigma_f, \vec{v}_i) = (o, \sigma' \cdot \sigma_f, \vec{v}_o)$, giving our goal (G1).

Case $\alpha = \texttt{alloc}$**:**

(H15) From (H1), (H3) and the rules for `alloc`, we have $\sigma' = (h', d')$ such that $d' \neq \bot$.

(H16) From (H15), (H2) and the rules of composition it follows that $\sigma_f = (h_f, \bot)$.

(H17) Because `alloc` does not modify any state aside from adding a cell, and because σ_f has no domain set, it follows from (H2) that $\sigma \# \sigma_f$ and that the action has the saame outcome, thus $\alpha(\sigma \cdot \sigma_f, \vec{v}_i) = (o, \sigma' \cdot \sigma_f, \vec{v}_o)$, giving our goal (G1).

Proposition: Consume soundness
Assume

(H1) $\mathsf{consume}(\hat{\sigma}, \delta, \vec{\hat{v}}_i) \rightsquigarrow (\mathsf{Ok}, \hat{\sigma}_f, \vec{\hat{v}}_o, \pi)$

(H2) $\theta, s, \sigma \models \hat{\sigma} \wedge \mathsf{SAT}_{\theta,s}(\pi)$

(H3) `consume` on $\mathbb{S}$ is sound.

To prove

(G1) $\exists \sigma_\delta, \sigma_f . \sigma_\delta \# \sigma_f \wedge \sigma = \sigma_\delta \cdot \sigma_f \wedge \theta, s, \sigma_\delta \models_\Delta \langle\delta\rangle(\vec{\hat{v}}_i; \vec{\hat{v}}_o) \wedge \theta, s, \sigma_f \models \hat{\sigma}_f$

Case $\delta \in \Delta_{\mathbb{S}}$**:**

(H4) From (H1) and PMapCons, given $\vec{\hat{v}}_i = \hat{i} :: \vec{\hat{v}}'_i$ and $\pi = \pi' :: \pi''$ we have $\texttt{get}(\hat{\sigma}, \hat{i}) \rightsquigarrow (\hat{i}', \hat{\sigma}_i, \pi')$, $\texttt{consume}(\hat{\sigma}_i, \delta, \vec{\hat{v}}'_i) \rightsquigarrow (o, \hat{\sigma}'_i, \vec{\hat{v}}_o, \pi'')$ and $\texttt{set}(\hat{\sigma}, \hat{i}', \hat{\sigma}'_i) = \hat{\sigma}_f$.

(H5) From (H4) and (H2), we have $\vec{\hat{v}}_i = \hat{i} :: \vec{\hat{v}}'_i \wedge [\![\hat{i}]\!]_{\theta,s} = i$

(H6) From (H2) and Get UX Soundness, we know $\exists \sigma_i.\, \texttt{get}(\sigma, i) = \sigma_i \wedge \theta, s, \sigma_i \models \hat{\sigma}_i$.

(H7) From (H2) and (H4), we have $\mathsf{SAT}_{\theta,s}(\pi'')$.

(H8) From (H6), (H7) and (H3) we can apply **??**, giving us $\exists \sigma_{\delta,i}, \sigma_{f,i}.\, \sigma_{\delta,i} \# \sigma'_i \wedge \sigma_i = \sigma_{\delta,i} \cdot \sigma'_i \wedge \theta, s, \sigma_{\delta,i} \models_\Delta \langle\delta\rangle(\vec{\hat{v}}'_i; \vec{\hat{v}}_o) \wedge \theta, s, \sigma_{f,i} \models \hat{\sigma}_{f,i}$.

(H9) We can then pick $\sigma_\delta = ([i \mapsto \sigma_{\delta,i}], \bot)$, and from (H8) and the definition of composition we know $\sigma_\delta \# \sigma$ and (H4) and the definition of $\texttt{set}$, $\sigma_f = \sigma \cdot \sigma_\delta$.

(H10) From the definition of $\models_\Delta$ and (H8) we then have $\theta, s, \sigma_\delta \models \langle\delta\rangle(\vec{v}_i; \vec{v}_o)$, which with (H9) gives our goal (G1).

Case $\delta = \texttt{domainset}$**:**

(H11) From (H1) and the rules for $\texttt{consume}$, we have $\hat{\sigma} = (\hat{h}, \hat{d})$ and $\hat{\sigma}_f = (\hat{h}, \bot)$ such that $\hat{d} \neq \bot$, as well as $\vec{\hat{v}}_i = []$ and $\vec{\hat{v}}_o = [\hat{d}]$

(H12) From the definition of $\models$ and (H11) we have $\sigma = (h, d)$ and $\sigma_f = (h, \bot)$ such that $d \neq \bot$ and $[\![\hat{d}]\!]_{\theta,s} = d$.

(H13) We can pick $\sigma_\delta = ([], d)$ – it follows that $\sigma_f \# \sigma_\delta$ and from the composition rules, $\sigma = \sigma_f \cdot \sigma_\delta$.

(H14) From the definition of $\models_\Delta$ and (H12) it follows that $\theta, s, \sigma_\delta \models_\Delta \langle\texttt{domainset}\rangle(\vec{\hat{v}}_i; \vec{\hat{v}}_o)$. Along with (H13), this gives our goal (G1).

Proposition: Consume completeness
Assume

(H1) $\texttt{consume}(\hat{\sigma}, \delta, \vec{\hat{v}}_i) \rightsquigarrow (\texttt{Ok}, \hat{\sigma}_f, \vec{\hat{v}}_o, \pi)$

(H2) $\theta, s, \sigma_f \models \hat{\sigma}_f \wedge \theta, s, \sigma_\delta \models_\Delta \langle\delta\rangle(\vec{\hat{v}}_i; \vec{\hat{v}}_o) \wedge \sigma_f \# \sigma_\delta$

(H3) $\texttt{consume}$ on $\mathbb{S}$ is complete.

To prove

(G1) $\exists \sigma.\, \sigma = \sigma_\delta \cdot \sigma_f \wedge \theta, s, \sigma \models \hat{\sigma} \wedge \mathsf{SAT}_{\theta,s}(\pi)$

Case $\delta \in \Delta_{\mathbb{S}}$**:**

(H4) From (H1) and PMapCons, given $\vec{\hat{v}}_i = \hat{i} :: \vec{\hat{v}}'_i$ and $\pi = \pi' :: \pi''$ we have $\texttt{get}(\hat{\sigma}, \hat{i}) \rightsquigarrow (\hat{i}', \hat{\sigma}_i, \pi')$, $\texttt{consume}(\hat{\sigma}_i, \delta, \vec{\hat{v}}'_i) \rightsquigarrow (o, \hat{\sigma}'_i, \vec{\hat{v}}_o, \pi'')$ and $\texttt{set}(\hat{\sigma}, \hat{i}', \hat{\sigma}'_i) = \hat{\sigma}_f$.

(H5) There are two cases here, $\sigma_\delta = \bot$ and $\sigma_\delta \neq \bot$. If $\sigma_\delta = \bot$, we trivially get $\sigma = \bot \cdot \sigma_f$ and the rest of our goal (G1).

(H6) Assume now $\sigma_\delta \neq \bot$. From the definition of $\models_\Delta$ and (H2), we have $\exists i, \sigma_{\delta,i}.\, [\![\hat{i}]\!]_{\theta,s} = i \wedge \sigma_\delta = ([i \mapsto \sigma_{\delta,i}], \bot) \wedge \sigma_{\delta,i} \models_\Delta \langle\delta\rangle(\vec{\hat{v}}'_i; \vec{\hat{v}}_o)$.

(H7) From (H2), (H4) and the definitions of $\models$ and composition we have $\exists \sigma_i'.\, \theta, s, \sigma_i' \models \hat{\sigma}_i' \wedge \sigma_i' \# \sigma_{\delta,i}$.

(H8) From (H7), (H6) and (H3), we can apply **??**, giving us $\exists \sigma_i.\, \sigma_i = \sigma_{\delta,i} \cdot \sigma_i' \wedge \theta, s, \sigma_i \models \hat{\sigma}_i \wedge \mathsf{SAT}_{\theta,s}(\pi'')$.

(H9) From the definition of $\models$, because consume doesn't affect any other cell, given $\theta, s, \sigma_i \models \hat{\sigma}_i$ and $\theta, s, \sigma_f \models \hat{\sigma}_f$ and $\mathtt{set}(\hat{\sigma}, \hat{i}', \hat{\sigma}_i') = \hat{\sigma}_f$, we obtain $\theta, s, \sigma \models \hat{\sigma}$.

(H10) From (H6), we have σ_δ of the form $\sigma_\delta = (h_\delta, \bot)$ where $i \in \mathrm{dom}(h_\delta)$. From composition rules, it follows that if $\sigma = \sigma_\delta \cdot \sigma_f$ we have $\sigma = (h, d)$ where $i \in \mathrm{dom}(h)$ too.

(H11) From (H10) and (H9), it follows that we can apply CPMapGetMatch such that $\mathtt{get}(\sigma, i) = \sigma_i$. We can then use Get OX Soundness, giving us, from (H4) that $\mathsf{SAT}_{\theta,s}(\pi')$.

(H12) It follows from (H11) and (H8) that $\mathsf{SAT}_{\theta,s}(\pi)$. This along with (H9) gives our goal (G1).

Case $\delta = \mathtt{domainset}$:

(H13) From (H1) and the rules for consume, we have $\hat{\sigma} = (\hat{h}, \hat{d})$ and $\hat{\sigma}_f = (\hat{h}, \bot)$ such that $\hat{d} \neq \bot$, as well as $\vec{\hat{v}}_i = []$, $\vec{\hat{v}}_o = [\hat{d}]$ and $\pi = []$

(H14) From the definition of $\models_\Delta$ and (H2) we have $\sigma_\delta = (\emptyset, d)$ such that $[\![\hat{d}]\!]_{\theta,s} = d$.

(H15) It follows from (H13), (H2) and the definition of $\models$ that σ_f is of the form $\sigma_f = (h, \bot)$.

(H16) From (H14), (H15) and the definition of $\models$ it follows that for $\sigma = \sigma_\delta \cdot \sigma_f$, we have $\theta, s, \sigma \models \hat{\sigma}$. This with the fact $\mathsf{SAT}_{\theta,s}([])$ gives our goal (G1).

Proposition: Consume: OX sound
Assume

(H1) $\forall o, \hat{\sigma}_f, \vec{\hat{v}}_o, \pi.\, \mathtt{consume}(\mathtt{OX}, \hat{\sigma}, \delta, \vec{\hat{v}}_i) \rightsquigarrow (o, \hat{\sigma}_f, \vec{\hat{v}}_o, \pi) \Rightarrow o_c = \mathtt{Ok}$

(H2) consume on $\mathbb{S}$ are OX sound.

To prove

(G1) $\exists \hat{\sigma}_f', \vec{\hat{v}}_o', \pi'.\, \mathtt{consume}(\mathtt{OX}, \hat{\sigma}, \delta, \vec{\hat{v}}_i, \pi) \rightsquigarrow (\mathtt{Ok}, \hat{\sigma}_f', \vec{\hat{v}}_o', \pi')$

Case $\delta \in \Delta_\mathbb{S}$:
From PMapProd we know consume for $\delta \in \Delta_\mathbb{S}$ only returns Ok if the underlying's consume returns Ok. It follows that if that is the case, then by (H2) there exists an execution of consume that succeeds for $\mathbb{S}$, that is lifted by PMap, giving us our goal (G1).

Case $\delta = \mathtt{domainset}$:
From the consume rules, we never vanish, so if all consumptions are Ok we must have $\hat{\sigma} = (\hat{h}, \hat{d})$ with $\hat{d} \neq \bot$, $\vec{\hat{v}}_i = []$, $\hat{\sigma}_f' = (\hat{h}, \bot)$, $\vec{\hat{v}}_o' = [\hat{d}]$, $\pi' = []$, which gives our goal (G1).

Proposition: Produce soundness
Assume

(H1) $\mathsf{produce}(\hat{\sigma}_f, \delta, \vec{\hat{v}}_i, \vec{\hat{v}}_o) \rightsquigarrow (\hat{\sigma}, \pi)$

(H2) $\mathsf{SAT}_{\theta,s}(\pi) \wedge \theta, s, \sigma \models \hat{\sigma}$

(H3) $\mathsf{produce}$ on $\mathbb{S}$ is sound.

To prove

(G1) $\exists \sigma_\delta, \sigma_f.\, \sigma_\delta \# \sigma_f \wedge \sigma = \sigma_\delta \cdot \sigma_f \wedge \theta, s, \sigma_\delta \models_\Delta \langle\delta\rangle(\vec{\hat{v}}_i; \vec{\hat{v}}_o) \wedge \theta, s, \sigma_f \models \hat{\sigma}_f$

Case $\delta \in \Delta_{\mathbb{S}}$**:**

(H4) Given (H1), from PMapProd we have $\mathsf{get}(\hat{\sigma}_f, \hat{i}) \rightsquigarrow (\hat{i}', \hat{\sigma}_{f,i}, \pi') \wedge \mathsf{produce}(\hat{\sigma}_{f,i}, \delta, \vec{\hat{v}}'_i, \vec{\hat{v}}_o) \rightsquigarrow (\hat{\sigma}_i, \pi'') \wedge \mathsf{set}(\hat{\sigma}_f, \hat{i}', \hat{\sigma}_i) = \hat{\sigma}$, given $\vec{\hat{v}}_i = \hat{i} :: \vec{\hat{v}}'_i$ and $\pi = \pi' :: \pi''$.

(H5) From (H2) and $\pi = \pi' :: \pi''$, it follows that $\mathsf{SAT}_{\theta,s}(\pi')$ and $\mathsf{SAT}_{\theta,s}(\pi'')$.

(H6) From the definition of set, (H4) and (H2), we have $\exists \sigma_i.\, \theta, s, \sigma_i \models \hat{\sigma}_i$.

(H7) From (H4), (H6), (H5) and (H3), we can apply **??**, giving us $\exists \sigma_{\delta,i}, \sigma_{f,i}.\, \sigma_{\delta,i} \# \sigma_{f,i} \wedge \sigma_i = \sigma_{\delta,i} \cdot \sigma_{f,i} \wedge \theta, s, \sigma_{\delta,i} \models_\Delta \langle\delta\rangle(\vec{\hat{v}}'_i; \vec{\hat{v}}_o) \wedge \theta, s, \sigma_{f,i} \models \hat{\sigma}_{f,i}$.

(H8) Given $[\![\hat{i}']\!]_{\theta,s} = i$, we can then have $\sigma_\delta = ([i \mapsto \sigma_{\delta,i}], \bot)$, or $\sigma_\delta = \bot$ if $\sigma_{\delta,i} = \bot$. By the definition of $\models_\Delta$, $\theta, s, \sigma_\delta \models_\Delta \langle\delta\rangle(\vec{\hat{v}}_i; \vec{\hat{v}}_o)$ in both cases.

(H9) Given set only modifies the state at a specific location leaving the rest untouched, from (H2) and (H7) we have $\theta, s, \sigma \models \hat{\sigma}$ and $\theta, s, \sigma_{f,i} \models \hat{\sigma}_{f,i}$, thus it follows that $\exists \sigma_f.\, \theta, s, \sigma_f \models \sigma_f$.

(H10) From (H7) and by the rules of composition, we know that if the two states $\sigma_{\delta,i}$ and $\sigma_{f,i}$ at index i are disjoint then the entire state σ_f and the singleton σ_δ are disjoint. Thus $\sigma_\delta \# \sigma_f \wedge \sigma = \sigma_\delta \cdot \sigma_f$, giving our goal (G1).

Case $\delta = \mathsf{domainset}$**:**

(H11) Given (H1), from PMapProdDomainSet we have $\hat{\sigma}_f = (\hat{h}, \bot)$, $\hat{\sigma} = (\hat{h}, \hat{d})$ and $\pi = [\mathrm{dom}(\hat{h}) \subseteq \hat{d}]$, with $\vec{\hat{v}}_i = []$ and $\vec{\hat{v}}_o = [\hat{d}]$.

(H12) We can pick σ_δ such that $\sigma_\delta = (\emptyset, d)$ with $[\![\hat{d}]\!]_{\theta,s} = d$, which from the definition of $\models_\Delta$ gives $\theta, s, \sigma_\delta \models_\Delta \langle\mathsf{domainset}\rangle(\vec{\hat{v}}_i; \vec{\hat{v}}_o)$.

(H13) We can pick σ_f such that $\sigma_f = (h, \bot)$ and $[\![\hat{h}]\!]_{\theta,s} = h$. From the definition of $\models$, we have $\theta, s, \sigma_f \models \hat{\sigma}_f$.

(H14) From (H12), (H13) and composition rules we have $\sigma_f \# \sigma_\delta$ and $\sigma = \sigma_f \cdot \sigma_\delta$. This completes our goal (G1).

Proposition: Produce completeness
Assume

(H1) $\theta, s, \sigma_f \models \hat{\sigma}_f \wedge \theta, s, \sigma_\delta \models_\Delta \langle\delta\rangle(\vec{\hat{v}}_i; \vec{\hat{v}}_o) \wedge \sigma_f \# \sigma_\delta$

(H2) $\mathsf{produce}$ on $\mathbb{S}$ is complete.

To prove

(G1) $\exists \hat{\sigma}.\, \mathtt{produce}(\hat{\sigma}_f, \delta, \vec{\hat{v}}_i, \vec{\hat{v}}_o) \rightsquigarrow (\hat{\sigma}, \pi) \wedge \mathsf{SAT}_{\theta,s}(\pi) \wedge \theta, s, (\sigma_f \cdot \sigma_\delta) \models \hat{\sigma}$

Case $\delta \in \Delta_{\mathbb{S}}$**:**

(H3) From (H1) and the definition of $\models_\Delta$, we have $\exists \hat{i}, i, \sigma_{\delta,i}.\, \vec{\hat{v}}_i = \hat{i} :: \vec{\hat{v}}'_i \wedge [\![\hat{i}]\!]_{\theta,s} = i \wedge \theta, s, \sigma_{\delta,i} \models_\Delta \langle\delta\rangle(\vec{\hat{v}}'_i; \vec{\hat{v}}_o) \wedge \sigma_\delta = ([i \mapsto \sigma_{\delta,i}], \perp)$.

(H4) Given the fact $\sigma_f \# \sigma_\delta$ and $\sigma_\delta \models_\Delta \langle\delta\rangle(\hat{i} :: \vec{\hat{v}}'_i; \vec{\hat{v}}_o)$, it means the index i is compatible with σ_f, so the state located at i is obtainable via `get`. We thus have $\exists \sigma_{f,i}.\, \mathtt{get}(\sigma_f, i) = \sigma_{f,i}$.

(H5) From (H4), (H3) and (H1), we can use Get OX Soundness, giving us $\exists \hat{\sigma}_{f,i}, \hat{i}', \pi'.\, \mathtt{get}(\hat{\sigma}_f, \hat{i}) \rightsquigarrow (\hat{i}', \hat{\sigma}_{f,i}, \pi) \wedge \theta, s, \sigma_{f,i} \models \hat{\sigma}_{f,i} \wedge [\![\hat{i}']\!]_{\theta,s} = i \wedge \mathsf{SAT}_{\theta,s}(\pi')$.

(H6) From (H1), because $\sigma_{f,i}$ and $\sigma_{\delta,i}$ are at the same index and $\sigma_f \# \sigma_\delta$, from the rules for composition we have $\sigma_{f,i} \# \sigma_{\delta,i}$.

(H7) From (H3), (H5) and (H6) we can apply **??** for $\mathbb{S}$, giving us $\exists \hat{\sigma}_i.\, \mathtt{produce}(\hat{\sigma}_{f,i}, \delta, \vec{\hat{v}}'_i, \vec{\hat{v}}_o) \rightsquigarrow (\hat{\sigma}_i, \pi'') \wedge \mathsf{SAT}_{\theta,s}(\pi'') \wedge \theta, s, (\sigma_{f,i} \cdot \sigma_{\delta,i}) \models \hat{\sigma}_i$.

(H8) We may define $\hat{\sigma}$ such that $\mathtt{set}(\hat{\sigma}_f, \hat{i}', \hat{\sigma}_i) = \hat{\sigma}$.

(H9) We have $\theta, s, \sigma_f \models \hat{\sigma}_f$ and $\theta, s, (\sigma_{f,i} \cdot \sigma_{\delta,i}) \models \hat{\sigma}_i$ from (H1) and (H7). Furthermore, we know $\sigma_\delta = ([i \mapsto \sigma_{\delta,i}], \perp)$ from (H3). From (H8) and the rules for `set` we know no state is lost and only the state at i is extended by $\sigma_{\delta,i}$, it thus follows that $\theta, s, (\sigma_f \cdot \sigma_\delta) \models \hat{\sigma}$.

(H10) From and (H5), (H7) and this we can apply PMapProd, giving us $\mathtt{produce}(\hat{\sigma}_f, \delta, \vec{\hat{v}}_i, \vec{\hat{v}}_o) \rightsquigarrow (\hat{\sigma}, \pi' :: \pi'')$.

(H11) From (H5) and (H7), we have $\mathsf{SAT}_{\theta,s}(\pi' :: \pi'')$. This, along with (H9) and (H10), gives our goal (G1).

Case $\delta = \mathtt{domainset}$**:**

(H12) From (H1) and the definition of $\models_\Delta$ we have $\sigma_\delta = (\emptyset, d)$ such that $\vec{\hat{v}}_i = []$, $\vec{\hat{v}}_o = [\hat{d}]$ and $[\![\hat{d}]\!]_{\theta,s} = d$.

(H13) From (H12) and (H1), by the definition of composition, we must have $\hat{\sigma}_f = (\hat{h}, \perp)$, $\sigma_f = (h, \perp)$ and $\mathrm{dom}(h) \subseteq d$.

(H14) From (H12) and (H13) we may apply PMapProdDomainSet, resulting in $\mathtt{produce}(\hat{\sigma}_f, \mathtt{domainset}, \vec{\hat{v}}_i, \vec{\hat{v}}_o) \rightsquigarrow ((\hat{h}, \hat{d}), [\mathrm{dom}(\hat{h}) \subseteq \hat{d}])$.

(H15) From (H1) and (H13) we have $\mathsf{SAT}_{\theta,s}([\mathrm{dom}(\hat{h}) \subseteq \hat{d}])$.

(H16) From (H12), (H13) and composition rules we have we have $\sigma_f \cdot \sigma_\delta = (h, d)$. It follows from (H14) that we have $\hat{\sigma} = (\hat{h}, \hat{d})$ such that $\theta, s, (\sigma_f \cdot \sigma_\delta) \models \hat{\sigma}$. This along with (H14) and (H15) gives our goal (G1).

□

1.3 Syntactic Partial Map

For the partial map with syntactic checks, we re-use the RA of PMap, as well as the concrete and symbolic action rules, and the `produce` and `consume` rules, as we only modify the behaviour of the `get` helper function. From this, we can thus re-use the soundness proofs for all of the axioms; the only part that needs to be proved again are the axioms for symbolic `get`, with regards to the concrete version that is taken from PMap.

1.3.1 `get` rules

$$\frac{(\hat{h}, \hat{d}) = \mathit{unwrap}(\hat{\sigma}) \qquad \hat{\imath} \in \mathrm{dom}(\hat{h}) \qquad \hat{\sigma}_i = \hat{h}(\hat{\imath})}{\mathtt{get}(\hat{\sigma}, \hat{\imath}) \rightsquigarrow (\hat{\imath}, \hat{\sigma}_i, [])}\ \textsc{SyntacticPMapGetMatch}$$

$$\frac{(\hat{h}, \hat{d}) = \mathit{unwrap}(\hat{\sigma}) \qquad \hat{\imath} \notin \mathrm{dom}(\hat{h}) \qquad \hat{\imath}' \in \mathrm{dom}(\hat{h}) \qquad \hat{\sigma}_i = \hat{h}(\hat{\imath}')}{\mathtt{get}(\hat{\sigma}, \hat{\imath}) \rightsquigarrow (\hat{\imath}', \hat{\sigma}_i, [\hat{\imath} = \hat{\imath}'])}\ \textsc{SyntacticPMapGetBranch}$$

$$\frac{(\hat{h}, \hat{d}) = \mathit{unwrap}(\hat{\sigma}) \qquad \hat{\imath} \notin \mathrm{dom}(\hat{h}) \qquad \hat{d} \neq \bot}{\mathtt{get}(\hat{\sigma}, \hat{\imath}) \rightsquigarrow (\hat{\imath}, \bot, [\hat{\imath} \notin \mathrm{dom}(\hat{h}) \land \hat{\imath} \in \hat{d}])}\ \textsc{SyntacticPMapGetAdd}$$

$$\frac{(\hat{h}, \hat{d}) = \mathit{unwrap}(\hat{\sigma}) \qquad \hat{\imath} \notin \mathrm{dom}(\hat{h}) \qquad \hat{d} = \bot}{\mathtt{get}(\hat{\sigma}, \hat{\imath}) \rightsquigarrow (\hat{\imath}, \bot, [\hat{\imath} \notin \mathrm{dom}(\hat{h})])}\ \textsc{SyntacticPMapGetBotDomain}$$

1.3.2 Soundness Proofs

Proof.

Proposition: Get OX Soundness
Assume

(H1) $\theta, s, \sigma \models \hat{\sigma} \land \mathtt{get}(\sigma, i) = \sigma_i \land [\![\hat{\imath}]\!]_{\theta,s} = i$

To prove

(G1) $\exists \hat{\sigma}_i, \hat{\imath}', \pi.\, \mathtt{get}(\hat{\sigma}, \hat{\imath}) \rightsquigarrow (\hat{\imath}', \hat{\sigma}_i, \pi) \land \theta, s, \sigma_i \models \hat{\sigma}_i \land [\![\hat{\imath}']\!]_{\theta,s} = i \land \mathsf{SAT}_{\theta,s}(\pi)$

We proceed by proving the property holds for all rules of the concrete `get`, resulting in three cases.

Case **CPMapGetMatch:**

(H2) Assume $(h, d) = \mathit{unwrap}(\sigma) \land i \in \mathrm{dom}(h) \land \sigma_i = h(i)$.

(H3) It follows from (H1) and the definition of $\models$ that $\hat{\sigma} = (\hat{h}, \hat{d})$.

(H4) Either $\hat{\imath} \in \hat{h}$ or not. If $\hat{\imath} \in \hat{h}$, we apply SyntacticPMapGetMatch, giving us $\mathtt{get}(\hat{\sigma}, \hat{\imath}) \rightsquigarrow (\hat{\imath}, \hat{h}(\hat{\imath}), [])$. It also follows from the definition of $\models$ that $\theta, s, \sigma_i \models \hat{h}(\hat{\imath})$, completing our goal (G1).

(H5) Otherwise $\hat{i} \notin \hat{h}$. From the definition of $\models$ we know $\exists \hat{i}'. [\![\hat{i}']\!]_{\theta,s} = i \wedge \hat{i}' \in \hat{h} \wedge \theta, s, \sigma_i \models \hat{h}(\hat{i}')$. We can apply SyntacticPMapGetBranch, giving us $\mathtt{get}(\hat{\sigma}, \hat{i}) \rightsquigarrow (\hat{i}', \hat{\sigma}_i, [\hat{i} = \hat{i}'])$. It follows that $\mathsf{SAT}_{\theta,s}([\hat{i} = \hat{i}'])$, which completes our goal (G1).

Case CPMapGetAdd**:** The rule SyntacticPMapGetAdd is exactly the same as PMapGetAdd – we omit the proof.

Case CPMapGetBotDomain**:** The rule SyntacticPMapGetBotDomain is exactly the same as PMapGetBotDomain – we omit the proof.

Proposition: Get UX Soundness
Assume

(H1) $\mathtt{get}(\hat{\sigma}, \hat{i}) \rightsquigarrow (\hat{i}', \hat{\sigma}_i, \pi) \wedge [\![\hat{i}]\!]_{\theta,s} = [\![\hat{i}']\!]_{\theta,s} = i \wedge \mathsf{SAT}_{\theta,s}(\pi)$

(H2) $\theta, s, \sigma \models \hat{\sigma}$.

To prove

(G1) $\exists \sigma_i. \mathtt{get}(\sigma, i) = \sigma_i \wedge \theta, s, \sigma_i \models \hat{\sigma}_i$

We proceed by proving the property holds for all rules of the symbolic get, resulting in four cases.

Case SyntacticPMapGetMatch**:**

(H3) Assume $(\hat{h}, \hat{d}) = \mathit{unwrap}(\hat{\sigma}) \wedge \hat{i} \in \mathrm{dom}(\hat{h}) \wedge \hat{\sigma}_i = \hat{h}(\hat{i})$.

(H4) From (H3), (H1) and SyntacticPMapGetMatch we have $\pi = []$ and $\hat{i}' = \hat{i}$.

(H5) From (H3), we know $\hat{\sigma} \neq \bot$, thus σ is of the form $\sigma = (h, d)$.

(H6) From (H2), (H3) and the definition of $\models$, we have $i \in h$ such that $\sigma_i = h(i)$ and $\theta, s, \sigma_i \models \hat{\sigma}_i$.

(H7) From (H6) we can apply CPMapGetMatch, thus $\mathtt{get}(\sigma, i) = \sigma_i$. This completes our goal (G1).

Case SyntacticPMapGetBranch**:**

(H8) Assume $(\hat{h}, \hat{d}) = \mathit{unwrap}(\hat{\sigma}) \wedge \hat{i}' \in \mathrm{dom}(\hat{h}) \wedge \hat{\sigma}_i = \hat{h}(\hat{i}')$

(H9) From (H8), (H1) and PMapGetMatch, we have $\pi = [\hat{i} = \hat{i}']$

(H10) From (H8), we have $\hat{\sigma} \neq \bot$, thus from (H2) we have $\sigma = (h, d)$

(H11) From (H2), (H8) and the definition of $\models$, we have $i \in h$ such that $\sigma_i = h(i)$ and $\theta, s, \sigma_i \models \hat{\sigma}_i$

(H12) From (H11) we can apply CPMapGetMatch, thus $\mathtt{get}(\sigma, i) = \sigma_i$. This completes our goal (G1).

Case SyntacticPMapGetAdd**:** The rule SyntacticPMapGetAdd is exactly the same as PMapGetAdd – we omit the proof.

Case SyntacticPMapGetBotDomain**:** The rule SyntacticPMapGetBotDomain is exactly the same as PMapGetBotDomain – we omit the proof. $\square$

1.4 Split Partial Map

For the split partial map, we define a new RA, $\text{PMap}_{\text{Split}}$, which we only use for the symbolic representation; the concrete compositional RA is still that defined in PMap. We re-use the definitions for action execution, `consume` and `produce`, as the only part of them that is modified is `get` and `set`. From this, we can thus re-use the soundness proofs for all of the axioms; the only part that needs to be proved again are the axioms for symbolic `get`, with regards to the concrete version that is taken from PMap.

1.4.1 Resource Algebra

While we keep the default PMap RA for the concrete compositional states, we need to define a new set of states for the symbolic compositional states, as defined below.

$$\text{PMap}_{\text{Split}}(I, \mathbb{S}) \stackrel{\text{def}}{=} I \xrightharpoonup{\textit{fin}} \mathbb{S}.\Sigma \times I \xrightharpoonup{\textit{fin}} \mathbb{S}.\Sigma \times \mathcal{P}(I)^?$$

Predicate satisfiability is defined with regards to the concrete compositional states, so we re-use it from PMap. We however need to re-define symbolic interpretation, as follows. We denote $\hat{h}_c$ and $\hat{h}_s$ the concrete and symbolic parts of the heap respectively.

$$\textsc{SplitPMapSymInterpretation}\quad \frac{\begin{array}{c}\forall \hat{i} \in \text{dom}(\hat{h}_c).\, \hat{i} \notin \hat{h}_s \wedge [\![\hat{i}]\!]_{\theta,s} = i \wedge i \in \text{dom}(h) \wedge \theta, s, h(i) \models \hat{h}_c(\hat{i}) \\ \forall \hat{i} \in \text{dom}(\hat{h}_s).\, \hat{i} \notin \hat{h}_c \wedge [\![\hat{i}]\!]_{\theta,s} = i \wedge i \in \text{dom}(h) \wedge \theta, s, h(i) \models \hat{h}_s(\hat{i}) \\ [\![\text{dom}(\hat{h}_c) \uplus \text{dom}(\hat{h}_s)]\!]_{\theta,s} = \text{dom}(h) \qquad [\![\hat{d}]\!]_{\theta,s} = d\end{array}}{\theta, s, (h, d) \models (\hat{h}_c, \hat{h}_s, \hat{d})}$$

1.4.2 `get` and `set` rules

$$\text{Given } \textit{wrap}(\hat{h}_c, \hat{h}_s, \hat{d}) \stackrel{\text{def}}{=} \begin{cases} \bot & \text{if } \text{dom}(\hat{h}_c) = \emptyset \wedge \text{dom}(\hat{h}_s) = \emptyset \wedge \hat{d} = \emptyset \\ (\hat{h}_c, \hat{h}_s, \hat{d}) & \text{otherwise} \end{cases}$$

$$\textit{unwrap}(\hat{\sigma}) \stackrel{\text{def}}{=} \begin{cases} ([], [], \emptyset) & \text{if } \hat{\sigma} = \bot \\ (\hat{h}_c, \hat{h}_s, \hat{d}) & \text{if } \hat{\sigma} = (\hat{h}_c, \hat{h}_s, \hat{d}) \end{cases}$$

$$\textsc{SplitPMapGetMatchCon}\quad \frac{(\hat{h}_c, \hat{h}_s, \hat{d}) = \textit{unwrap}(\hat{\sigma}) \qquad \texttt{is\_concrete}\ \hat{i} \qquad \hat{i} \in \text{dom}(\hat{h}_c) \qquad \hat{\sigma}_i = \hat{h}_c(\hat{i})}{\texttt{get}(\hat{\sigma}, \hat{i}) \rightsquigarrow (\hat{i}, \hat{\sigma}_i, [])}$$

$$\textsc{SplitPMapGetMatchSym}\quad \frac{(\hat{h}_c, \hat{h}_s, \hat{d}) = \textit{unwrap}(\hat{\sigma}) \qquad \hat{i} \in \text{dom}(\hat{h}_s) \qquad \hat{\sigma}_i = \hat{h}_s(i)}{\texttt{get}(\hat{\sigma}, \hat{i}) \rightsquigarrow (\hat{i}, \hat{\sigma}_i, [])}$$

$$\textsc{SplitPMapGetBranch}\quad \frac{\begin{array}{c}(\hat{h}_c, \hat{h}_s, \hat{d}) = \textit{unwrap}(\hat{\sigma}) \\ \hat{h}_{all} = \hat{h}_c \cup \hat{h}_s \qquad \hat{i} \notin \text{dom}(\hat{h}_{all}) \qquad \hat{i}' \in \text{dom}(\hat{h}_{all}) \qquad \hat{\sigma}_i = \hat{h}_{all}(\hat{i}')\end{array}}{\texttt{get}(\hat{\sigma}, \hat{i}) \rightsquigarrow (\hat{i}', \hat{\sigma}_i, [\hat{i} = \hat{i}'])}$$

$$\frac{\textsc{SplitPMapGetAdd} \quad (\hat{h}_c, \hat{h}_s, \hat{d}) = \mathit{unwrap}(\hat{\sigma}) \qquad \hat{h}_{all} = \hat{h}_c \cup \hat{h}_s \qquad \hat{i} \notin \mathrm{dom}(\hat{h}_{all}) \qquad \hat{d} \neq \bot}{\mathtt{get}(\hat{\sigma}, \hat{i}) \rightsquigarrow (\hat{i}, \bot, [\hat{i} \notin \mathrm{dom}(\hat{h}_{all}) \wedge \hat{i} \in \hat{d}])}$$

$$\frac{\textsc{SplitPMapGetBotDomain} \quad (\hat{h}_c, \hat{h}_s, \hat{d}) = \mathit{unwrap}(\hat{\sigma}) \qquad \hat{h}_{all} = \hat{h}_c \cup \hat{h}_s \qquad \hat{i} \notin \mathrm{dom}(\hat{h}_{all}) \qquad \hat{d} = \bot}{\mathtt{get}(\hat{\sigma}, \hat{i}) \rightsquigarrow (\hat{i}, \bot, [\hat{i} \notin \mathrm{dom}(\hat{h}_{all})])}$$

$$\frac{\textsc{SplitPMapSetSomeCon} \quad (\hat{h}_c, \hat{h}_s, \hat{d}) = \mathit{unwrap}(\hat{\sigma}) \qquad \hat{\sigma}_i \neq \bot \qquad \mathtt{is\_concrete}_\Sigma\ \hat{\sigma}_i \qquad \mathtt{is\_concrete}\ \hat{i} \qquad \hat{h}'_c = \hat{h}_c[\hat{i} \leftarrow \hat{\sigma}_i] \qquad \hat{h}'_s = \hat{h}_s[\hat{i} \not\leftarrow] \qquad \hat{\sigma}' = \mathit{wrap}(\hat{h}'_c, \hat{h}'_s, \hat{d})}{\mathtt{set}(\hat{\sigma}, \hat{i}, \hat{\sigma}_i) = \hat{\sigma}'}$$

$$\frac{\textsc{SplitPMapSetSomeSym} \quad (\hat{h}_c, \hat{h}_s, \hat{d}) = \mathit{unwrap}(\hat{\sigma}) \qquad \hat{s}_i \neq \bot \qquad \neg(\mathtt{is\_concrete}_\Sigma\ \hat{\sigma}_i \vee \mathtt{is\_concrete}\ \hat{i}) \qquad \hat{h}'_c = \hat{h}_c[\hat{i} \not\leftarrow] \qquad \hat{h}'_s = \hat{h}_s[\hat{i} \leftarrow \hat{\sigma}_i] \qquad \hat{\sigma}' = \mathit{wrap}(\hat{h}'_c, \hat{h}'_s, \hat{d})}{\mathtt{set}(\hat{\sigma}, \hat{i}, \hat{\sigma}_i) = \hat{\sigma}'}$$

$$\frac{\textsc{SplitPMapSetNone} \quad (\hat{h}_c, \hat{h}_s, \hat{d}) = \mathit{unwrap}(\hat{\sigma}) \qquad \hat{\sigma}_i = \bot \qquad \hat{h}'_c = \hat{h}_c[\hat{i} \not\leftarrow] \qquad \hat{h}'_s = \hat{h}_s[\hat{i} \not\leftarrow] \qquad \hat{\sigma}' = \mathit{wrap}(\hat{h}'_c, \hat{h}'_s, \hat{d})}{\mathtt{set}(\hat{\sigma}, \hat{i}, \hat{\sigma}_i) = \hat{\sigma}'}$$

1.4.3 Soundness Proofs

Proof.

Proposition: Get OX Soundness
Assume

(H1) $\theta, s, \sigma \models \hat{\sigma} \wedge \mathtt{get}(\sigma, i) = \sigma_i \wedge [\![\hat{i}]\!]_{\theta,s} = i$

To prove

(G1) $\exists \hat{\sigma}_i, \hat{i}', \pi.\ \mathtt{get}(\hat{\sigma}, \hat{i}) \rightsquigarrow (\hat{i}', \hat{\sigma}_i, \pi) \wedge \theta, s, \sigma_i \models \hat{\sigma}_i \wedge [\![\hat{i}']\!]_{\theta,s} = i \wedge \mathsf{SAT}_{\theta,s}(\pi)$

We proceed by proving the property holds for all rules of the concrete `get`, resulting in three cases.

Case CPMapGetMatch**:**

(H2) Assume $(h, d) = \mathit{unwrap}(\sigma) \wedge i \in \mathrm{dom}(h) \wedge \sigma_i = h(i)$.

(H3) It follows from (H1) and the definition of $\models$ that $\hat{\sigma} = (\hat{h}_c, \hat{h}_s, \hat{d})$. From here we have three cases: $\hat{i}$ is present directly in $\hat{h}_c$, or in $\hat{h}_s$, or neither.

(H4) From (H3), assume the binding is in the concrete part of the heap: $\hat{i} \in \hat{h}_c$. From the definition of `set`, we know that if an entry is in the concrete part it must be concrete; thus `is_concrete` $\hat{i}$. From this, we can apply SplitPMapGetMatchCon, giving us $\mathtt{get}(\hat{\sigma}, \hat{i}) \rightsquigarrow (\hat{i}, \hat{h}_c(\hat{i}), [])$. From the definition of $\models$, it also follows that $\theta, s, \sigma_i \models \hat{h}_c(\hat{i})$. This, along with the fact $\mathsf{SAT}_{\theta,s}([])$, gives our goal (G1).

(H5) Assume now from (H3) that the binding is in the symbolic part of the heap, $\hat{h}_s$. From this, we can apply SplitPMapGetMatchSym, giving us $\texttt{get}(\hat{\sigma}, \hat{\imath}) \rightsquigarrow (\hat{\imath}, \hat{h}_s(\hat{\imath}), [])$. From the definition of $\models$, it also follows that $\theta, s, \sigma_i \models \hat{h}_s(\hat{\imath})$. This again gives our goal (G1).

(H6) Finally, it may be that $\hat{\imath} \notin \hat{h}_c \wedge \hat{\imath} \notin \hat{h}_s$. From the definition of $\models$, there must however still exist a $\hat{\imath}'$ such that $[\![\hat{\imath}']\!]_{\theta,s} = i$ and given $\hat{h}_{all} = \hat{h}_c \cup \hat{h}_s$, $\theta, s, \sigma_i \models \hat{h}_{all}(\hat{\imath}')$. From this we may apply SplitPMapGetBranch, giving us $\texttt{get}(\hat{\sigma}, \hat{\imath}) \rightsquigarrow (\hat{\imath}', \hat{\sigma}_i, [\hat{\imath} = \hat{\imath}'])$. Because we have $[\![\hat{\imath}]\!]_{\theta,s} = i$, it follows that $\mathsf{SAT}_{\theta,s}([\hat{\imath} = \hat{\imath}'])$, completing our goal (G1).

Case CPMapGetAdd**:**

(H7) Assume $(h, d) = \mathit{unwrap}(\sigma) \wedge i \notin \mathrm{dom}(h) \wedge d \neq \bot \wedge i \in d$.

(H8) It follows from (H1) and the definition of $\models$ that $\hat{\sigma} = (\hat{h}_c, \hat{h}_s, \hat{d})$ such that $[\![\hat{\imath}]\!]_{\theta,s} = i$, $\hat{\imath} \in \hat{d}$ and $\hat{\imath} \notin \mathrm{dom}(\hat{h}_c \cup \hat{h}_s)$.

(H9) We can then apply SplitPMapGetAdd, giving us $\texttt{get}(\hat{\sigma}, \hat{\imath}) \rightsquigarrow (\hat{\imath}, \bot, [\hat{\imath} \notin \mathrm{dom}(\hat{h}_c \cup \hat{h}_s) \wedge \hat{\imath} \notin \hat{d}])$.

(H10) From (H8) it follows that $\mathsf{SAT}_{\theta,s}([\hat{\imath} \notin \mathrm{dom}(\hat{h}) \wedge \hat{\imath} \notin \hat{d}])$, which completes our goal (G1).

Case CPMapGetBotDomain**:**

(H11) Assume $(h, d) = \mathit{unwrap}(\sigma) \wedge i \notin \mathrm{dom}(h) \wedge d = \bot$.

(H12) From (H11), (H1) and CPMapGetBotDomain, we have $\sigma_i = \bot$.

(H13) It follows from (H1) and the definition of $\models$ that $\hat{\sigma} = (\hat{h}_c, \hat{h}_s, \bot)$ such that $[\![\hat{\imath}]\!]_{\theta,s} = i$ and $\hat{\imath} \notin \mathrm{dom}(\hat{h}_c \cup \hat{h}_s)$.

(H14) We can then apply SplitPMapGetBotDomain, giving us $\texttt{get}(\hat{\sigma}, \hat{\imath}) \rightsquigarrow (\hat{\imath}, \bot, [\hat{\imath} \notin \mathrm{dom}(\hat{h}_c \cup \hat{h}_s)])$.

(H15) From (H13) it follows that $\mathsf{SAT}_{\theta,s}([\hat{\imath} \notin \mathrm{dom}(\hat{h}_c \cup \hat{h}_s)])$, which completes our goal (G1).

Proposition: Get UX Soundness
Assume

(H1) $\texttt{get}(\hat{\sigma}, \hat{\imath}) \rightsquigarrow (\hat{\imath}', \hat{\sigma}_i, \pi) \wedge [\![\hat{\imath}]\!]_{\theta,s} = [\![\hat{\imath}']\!]_{\theta,s} = i \wedge \mathsf{SAT}_{\theta,s}(\pi)$

(H2) $\sigma. \theta, s, \sigma \models \hat{\sigma}$

To prove

(G1) $\exists \sigma_i. \texttt{get}(\sigma, i) = \sigma_i \wedge \theta, s, \sigma_i \models \hat{\sigma}_i$

We proceed by proving the property holds for all rules of the symbolic `get`, resulting in five cases.

Case SplitPMapGetMatchCon**:**

(H3) Assume $(\hat{h}_c, \hat{h}_s, \hat{d}) = \mathit{unwrap}(\hat{\sigma}) \wedge \texttt{is\_concrete}\ \hat{\imath} \wedge \hat{\imath} \in \mathrm{dom}(\hat{h}_c) \wedge \hat{\sigma}_i = \hat{h}_c(\hat{\imath})$.

(H4) From (H3), (H1) and SPLITPMAPGETMATCHCON we have $\pi = []$ and $\hat{i}' = \hat{i}$.

(H5) From (H3), we know $\hat{\sigma} \neq \bot$, thus σ is of the form $\sigma = (h, d)$.

(H6) From (H2), (H3) and the definition of $\models$, we have $i \in \mathrm{dom}(h)$ such that $\sigma_i = h(i)$ and $\theta, s, \sigma_i \models \hat{\sigma}_i$.

(H7) From (H6) we can apply CPMAPGETMATCH, thus $\mathtt{get}(\sigma, i) = \sigma_i$. This completes our goal (G1).

Case SPLITPMAPGETMATCHSYM**:** This is proved analogously to the above.

Case SPLITPMAPGETBRANCH**:**

(H8) Assume $(\hat{h}_c, \hat{h}_s, \hat{d}) = \mathit{unwrap}(\hat{\sigma}) \wedge \hat{h}_{all} = \hat{h}_c \cup \hat{h}_s \wedge \hat{i} \notin \mathrm{dom}(\hat{h}_{all}) \wedge \hat{i}' \in \mathrm{dom}(\hat{h}_{all}) \wedge \hat{\sigma}_i = \hat{h}_{all}(\hat{i}')$.

(H9) From (H8), (H1) and SPLITPMAPGETBRANCH we have $\pi = [\hat{i} = \hat{i}']$.

(H10) From (H8) we know $\hat{\sigma} \neq \bot$, thus σ is of the form $\sigma = (h, d)$.

(H11) From (H8), (H2) and the definition of $\models$, we have $i \in \mathrm{dom}(h)$ such that $\sigma_i = h(i)$ and $\theta, s, \sigma_i \models \hat{\sigma}_i$.

(H12) From (H11) we can apply CPMAPGETMATCH, thus $\mathtt{get}(\sigma, i) = \sigma_i$. This completes our goal (G1).

Case SPLITPMAPGETADD**:**

(H13) Assume $(\hat{h}_c, \hat{h}_s, \hat{d}) = \mathit{unwrap}(\hat{\sigma}) \wedge \hat{h}_{all} = \hat{h}_c \cup \hat{h}_s \wedge \hat{i} \notin \mathrm{dom}(\hat{h}_{all}) \wedge \hat{d} \neq \bot$.

(H14) From (H13), (H1) and SPLITPMAPGETADD we have $\pi = [\hat{i} \notin \mathrm{dom}(\hat{h}_{all}) \wedge \hat{i} \in \hat{d}]$, $\hat{i}' = \hat{i}$ and $\hat{\sigma}_i = \bot$.

(H15) From (H13) we know $\hat{\sigma} \neq \bot$, thus σ is of the form $\sigma = (h, d)$.

(H16) From (H1) and (H14), we know $\mathsf{SAT}_{\theta,s}(\pi)$, such that from (H2) we have $i \notin \mathrm{dom}(h) \wedge i \in d$.

(H17) From (H16) we can straightforwardly apply CPMAPGETADD, giving us $\mathtt{get}(\sigma, i) = \bot$ – together with (H14) and **??** this completes our goal (G1).

Case SPLITPMAPGETBOTDOMAIN**:**

(H18) Assume $(\hat{h}_c, \hat{h}_s, \hat{d}) = \mathit{unwrap}(\hat{\sigma}) \wedge \hat{h}_{all} = \hat{h}_c \cup \hat{h}_s \wedge \hat{i} \notin \mathrm{dom}(\hat{h}_{all}) \wedge \hat{d} = \bot$.

(H19) From (H18), (H1) and SPLITPMAPGETBOTDOMAIN we have $\pi = [\hat{i} \notin \mathrm{dom}(\hat{h}_{all})]$ and $\hat{i} = \hat{i}$.

(H20) From (H18) we have $\hat{d} = \bot$, so given $(h, d) = \mathit{unwrap}(\sigma)$ and (H2), we have $d = \bot$.

(H21) From (H1) $\mathsf{SAT}_{\theta,s}(\pi)$, thus from (H19) and (H2), $i \notin \mathrm{dom}(h)$.

(H22) From (H21) and (H20) we can apply CPMAPGETBOTDOMAIN, thus $\mathtt{get}(\sigma, i) = \bot$ – together with (H19) and **??** this completes our goal (G1).

$\square$

1.5 Abstract Location Partial Map

For the abstract location partial map, we define a new set of states, $\text{PMap}_{\text{ALoc}}$, which we only use for the symbolic representation; the concrete compositional RA is still that defined in PMap.

For the PMap with syntactic matching and $\text{PMap}_{\text{Split}}$ we considered compatiblity with any "regular" PMap that has the same domain I. Here however, because $\text{PMap}_{\text{ALoc}}$ enforces the domain be abstract locations (as strings), we must consider compatiblity with regards to $\text{PMap}(\text{Loc}, \mathbb{S})$ only.

Concrete locations (or just locations) are values of the form $\text{loc}(a)$, where a is the name of the location – it always holds that given two locations $\text{loc}(a)$ and $\text{loc}(b)$, $\text{loc}(a) = \text{loc}(b) \iff a = b$. They are uninterpreted values, and do not allow any operations; as such, they can be used as sorts of pointers, where they represent address in a map, to the difference that they do not allow pointer arithmetics.

Abstract locations are *symbolic* values of the form $\text{aloc}(a)$, where a is the name of the abstract location. Unlike with concrete locations, their name does not uniquely identify them: we may have a substitution θ such that for $\text{aloc}(a)$ and $\text{aloc}(b)$ with $a \neq b$, we have $[\![\text{aloc}(a)]\!]_{\theta,s} = [\![\text{aloc}(b)]\!]_{\theta,s}$.

We also introduce the function `to_aloc`, that returns the name of an abstract location associated with a symbolic value if it exists and is found, and $\bot$ otherwise. This is a best effort function, that may not find an abstract location that exists.

$$\texttt{to\_aloc}\ i = a \implies \exists a.\, [\![i]\!]_{\theta,s} = [\![\text{aloc}(a)]\!]_{\theta,s}$$

1.5.1 Resource Algebra

$$\text{PMap}_{\text{ALoc}}(\mathbb{S}) \stackrel{\text{def}}{=} \text{Str} \xrightarrow{fin} \mathbb{S}.\Sigma \times \mathcal{P}(\text{LVal})^{?}$$

$$\textsc{ALocPMapSymInterpretation}\ \frac{\begin{array}{c}\forall a \in \text{dom}(\hat{h}).\, [\![\text{aloc}(a)]\!]_{\theta,s} = i \land i \in \text{dom}(h) \land \theta, s, h(i) \models \hat{h}(a) \\ [\![\{\text{aloc}(a) : a \in \text{dom}(\hat{h})\}]\!]_{\theta,s} = \text{dom}(h) \qquad [\![\hat{d}]\!]_{\theta,s} = d\end{array}}{\theta, s, (h, d) \models (\hat{h}, \hat{d})}$$

1.5.2 `get` and `set` rules

We now present the rules for this state model; in particular, we again only need to concern ourselves with the `get` and `set` internal methods. We also extend `get` to receive a mode $M = \{\texttt{MATCH}, \texttt{NO\_MATCH}\}$; it is `MATCH` in `consume` and `produce`, and `NO_MATCH` during action execution. The rules for `execute_action`, `consume` and `produce` are omitted as they are analogous to those in PMap, to the exception that `get` receives a matching mode. To simplify the rules, we also add the notation $\in^{?}$, that checks for membership in a possibly $\bot$ set, in which case the result is `true`.

$$\text{Given } \mathit{wrap}(\hat{h},\hat{d}) \stackrel{\text{def}}{=} \begin{cases} \bot & \text{if } \operatorname{dom}(\hat{h}) = \emptyset \wedge \hat{d} = \bot \\ (\hat{h},\hat{d}) & \text{otherwise} \end{cases}$$

$$\mathit{unwrap}(\hat{\sigma}) \stackrel{\text{def}}{=} \begin{cases} ([],\bot) & \text{if } \hat{\sigma} = \bot \\ (\hat{h},\hat{d}) & \text{if } \hat{\sigma} = (\hat{h},\hat{d}) \end{cases}$$

$$a \in^{?} \hat{d} \stackrel{\text{def}}{=} \begin{cases} \texttt{true} & \text{if } \hat{d} = \bot \\ a \in \hat{d} & \text{otherwise} \end{cases}$$

$$\textsc{ALocPMapGetMatch}\quad \frac{(\hat{h},\hat{d}) = \mathit{unwrap}(\hat{\sigma}) \qquad a = \texttt{to\_aloc}\ \hat{\imath} \qquad a \neq \bot \qquad a \in \operatorname{dom}(\hat{h}) \qquad \hat{\sigma}_i = \hat{h}(a)}{\texttt{get}(\hat{\sigma},\hat{\imath},m) \rightsquigarrow (\operatorname{aloc}(a),\hat{\sigma}_i,[])}$$

$$\textsc{ALocPMapGetNoMatchNotFound}\quad \frac{(\hat{h},\hat{d}) = \mathit{unwrap}(\hat{\sigma}) \qquad a = \texttt{to\_aloc}\ \hat{\imath} \qquad a \neq \bot \qquad a \notin \operatorname{dom}(\hat{h})}{\texttt{get}(\hat{\sigma},\hat{\imath},\texttt{NO\_MATCH}) \rightsquigarrow (\operatorname{aloc}(a),\bot,[\hat{\imath} \in^{?} \hat{d}])}$$

$$\textsc{ALocPMapGetNoMatchNew}\quad \frac{(\hat{h},\hat{d}) = \mathit{unwrap}(\hat{\sigma}) \qquad a = \texttt{to\_aloc}\ \hat{\imath} \qquad a = \bot \qquad a' = \texttt{fresh\_aloc ()}}{\texttt{get}(\hat{\sigma},\hat{\imath},\texttt{NO\_MATCH}) \rightsquigarrow (\operatorname{aloc}(a'),\bot,[\hat{\imath} = \operatorname{aloc}(a') \wedge \hat{\imath} \in^{?} \hat{d}])}$$

$$\textsc{ALocPMapGetMatchNotFound}\quad \frac{(\hat{h},\hat{d}) = \mathit{unwrap}(\hat{\sigma}) \qquad a = \texttt{to\_aloc}\ \hat{\imath} \qquad a \neq \bot \qquad a \notin \operatorname{dom}(\hat{h})}{\texttt{get}(\hat{\sigma},\hat{\imath},\texttt{MATCH}) \rightsquigarrow (\operatorname{aloc}(a),\bot,[\hat{\imath} \notin \{\operatorname{aloc}(a') : a' \in \operatorname{dom}(\hat{h})\} \wedge \hat{\imath} \in^{?} \hat{d}])}$$

$$\textsc{ALocPMapGetMatchNew}\quad \frac{(\hat{h},\hat{d}) = \mathit{unwrap}(\hat{\sigma}) \qquad a = \texttt{to\_aloc}\ \hat{\imath} \qquad a = \bot \qquad a' = \texttt{fresh\_aloc ()}}{\texttt{get}(\hat{\sigma},\hat{\imath},\texttt{MATCH}) \rightsquigarrow (\operatorname{aloc}(a'),\bot,[\hat{\imath} = \operatorname{aloc}(a') \wedge \hat{\imath} \notin \{\operatorname{aloc}(a'') : a'' \in \operatorname{dom}(\hat{h})\} \wedge \hat{\imath} \in^{?} \hat{d}])}$$

$$\textsc{ALocPMapMatching}\quad \frac{\begin{array}{c}(\hat{h},\hat{d}) = \mathit{unwrap}(\hat{\sigma}) \\ a = \texttt{to\_aloc}\ \hat{\imath} \qquad a \neq \bot \qquad a \notin \operatorname{dom}(\hat{h}) \qquad a' \in \operatorname{dom}(\hat{h}) \qquad \hat{\sigma}_i = \hat{h}(a')\end{array}}{\texttt{get}(\hat{\sigma},\hat{\imath},\texttt{MATCH}) \rightsquigarrow (\operatorname{aloc}(a'),\hat{\sigma}_i,[\operatorname{aloc}(a) = \operatorname{aloc}(a')])}$$

$$\textsc{ALocPMapMatchingBot}\quad \frac{(\hat{h},\hat{d}) = \mathit{unwrap}(\hat{\sigma}) \qquad a = \texttt{to\_aloc}\ \hat{\imath} \qquad a = \bot \qquad a' \in \operatorname{dom}(\hat{h}) \qquad \hat{\sigma}_i = \hat{h}(a')}{\texttt{get}(\hat{\sigma},\hat{\imath},\texttt{MATCH}) \rightsquigarrow (\operatorname{aloc}(a'),\hat{\sigma}_i,[\hat{\imath} = \operatorname{aloc}(a')])}$$

$$\textsc{ALocPMapSetSome}\quad \frac{\begin{array}{c}(\hat{h},\hat{d}) = \mathit{unwrap}(\hat{\sigma}) \\ a = \texttt{to\_aloc}\ \hat{\imath} \qquad a \neq \bot \qquad \hat{\sigma}_i \neq \bot \qquad \hat{h}' = \hat{h}[a \leftarrow \hat{\sigma}_i] \qquad \hat{\sigma}' = \mathit{wrap}(\hat{h}',\hat{d})\end{array}}{\texttt{set}(\hat{\sigma},\hat{\imath},\hat{\sigma}_i) = \hat{\sigma}'}$$

$$\textsc{ALocPMapSetNone}\quad \frac{\begin{array}{c}(\hat{h},\hat{d}) = \mathit{unwrap}(\hat{\sigma}) \\ a = \texttt{to\_aloc}\ \hat{\imath} \qquad a \neq \bot \qquad \hat{\sigma}_i = \bot \qquad \hat{h}' = \hat{h}[a \not\leftarrow] \qquad \hat{\sigma}' = \mathit{wrap}(\hat{h}',\hat{d})\end{array}}{\texttt{set}(\hat{\sigma},\hat{\imath},\hat{\sigma}_i) = \hat{\sigma}'}$$

1.5.3 Soundness Proofs

Because we extend the signature of get with a matching mode, we proceed with the proofs for `get` and `set` by doing no assumption on the value of the mode m. Interestingly, we will see that *we cannot prove OX soundness* with $m = \texttt{NO\_MATCH}$, as it only holds for $m = \texttt{MATCH}$. This is central to the difference in behaviour between PMap(Loc, $\mathbb{S}$) and PMap$_{\text{ALoc}}$($\mathbb{S}$).
Proof.

Proposition: Get OX Soundness
Assume

(H1) $\theta, s, \sigma \models \hat{\sigma} \wedge \texttt{get}(\sigma, i) = \sigma_i \wedge [\![\hat{i}]\!]_{\theta,s} = i$

To prove

(G1) $\exists \hat{\sigma}_i, \hat{i}', \pi.\, \texttt{get}(\hat{\sigma}, \hat{i}) \rightsquigarrow (\hat{i}', \hat{\sigma}_i, \pi) \wedge \theta, s, \sigma_i \models \hat{\sigma}_i \wedge [\![\hat{i}']\!]_{\theta,s} = i \wedge \mathsf{SAT}_{\theta,s}(\pi)$

We proceed by proving the property holds for all rules of the concrete `get`, resulting in three cases.

Case CPMapGetMatch**:**

(H2) Assume $(h, d) = \textit{unwrap}(\sigma) \wedge i \in \text{dom}(h) \wedge \sigma_i = h(i)$.

(H3) It follows from (H1) and the definition of $\models$ that $\hat{\sigma} = (\hat{h}, \hat{d})$ such that $\exists \hat{i}'.\, \hat{i}' = \text{aloc}(a') \wedge a' \in \text{dom}(\hat{h})$, $[\![\hat{i}']\!]_{\theta,s} = i$ and $\theta, s, \sigma_i \models \hat{h}(a')$.

(H4) From here, multiple cases are possible: either `to_aloc` $\hat{i} = a'$, or `to_aloc` $\hat{i} = a$ such that $a \neq a' \wedge [\![\text{aloc}(a)]\!]_{\theta,s} = [\![\text{aloc}(a')]\!]_{\theta,s}$, or `to_aloc` $\hat{i} = \bot$. Indeed, `to_aloc` is a best effort function, that may not find a match despite there being one; it can also be that $\hat{i}$ is a fresh symbolic variable, such that in the current path condition nothing binds it to aloc(a).

(H5) In the first case, `to_aloc` $\hat{i} = a'$ we can apply ALocPMapGetMatch, giving us our goal (G1).

(H6) In the second case, `to_aloc` $\hat{i} = a \wedge a \neq a' \wedge [\![\text{aloc}(a)]\!]_{\theta,s} = [\![\text{aloc}(a')]\!]_{\theta,s}$, we may apply ALocPMapMatching, *but only when* $m = \texttt{MATCH}$. This gives our goal (G1). Otherwise, when $m = \texttt{NO\_MATCH}$, the only rule that suits is ALocPMapGetNoMatchNotFound, which gives us $\hat{\sigma}_i = \bot$, from which it follows that $\theta, s, \sigma_i \not\models \hat{\sigma}_i$.

(H7) In the last case, `to_aloc` $\hat{i} = \bot$. Again, if $m = \texttt{MATCH}$, we can apply ALocPMapMatchingBot and get our goal (G1); otherwise, the only applicable rule is ALocPMapGetNoMatchNew, which invalidates our goal, as $\theta, s, \sigma_i \not\models \hat{\sigma}_i$.

Case CPMapGetAdd**:**

(H8) Assume $(h, d) = \textit{unwrap}(\sigma) \wedge i \notin \text{dom}(h) \wedge d \neq \bot \wedge i \in d$.

(H9) From (H14), (H1) and CPMapGetAdd, we have $\sigma_i = \bot$.

(H10) It follows from (H1) and the definition of $\models$ that $\hat{\sigma} = (\hat{h}, \hat{d})$ such that $\hat{i} = \text{aloc}(a') \wedge \hat{i} \in \hat{d}$ and $a' \notin \text{dom}(\hat{h})$.

(H11) We again get three possible cases, depending on `to_aloc`: either `to_aloc` $\hat{i} = a'$, or `to_aloc` $\hat{i} = a$ such that $a \neq a' \wedge [\![\text{aloc}(a)]\!]_{\theta,s} = [\![\text{aloc}(a')]\!]_{\theta,s}$, or `to_aloc` $\hat{i} = \bot$.

(H12) In the two first cases, from the definition of $\models$ and `to_aloc`, it still holds from (H10) that $a' \notin \hat{h}$ and $a \notin \hat{h}$, respectively. Depending on the mode m, we thus apply ALocPMapGetNoMatchNotFound or ALocPMapGetMatchNotFound, giving us $\hat{\sigma}_i = \bot$ and $\pi = [\hat{i} \in^? \hat{d}]$ or $\pi = [\hat{i} \notin \{\text{aloc}(a') : a' \in \text{dom}(\hat{h})\} \wedge \hat{i} \in^? \hat{d}]$, which from (H10) we know $\mathsf{SAT}_{\theta,s}(\pi)$ in both cases, giving our goal (G1).

(H13) In the last case, we have `to_aloc` $\hat{i} = \bot$, we thus apply ALocPMapGetNoMatch-New or ALocPMapGetMatchNew depending on m, giving us $a' =$ `fresh_aloc ()`, $\hat{\sigma}_i = \bot$, $\hat{i}' = \text{aloc}(a')$ and $\pi = [\hat{i} = \text{aloc}(a') \wedge \hat{i} \in^? \hat{d}]$ or $[\hat{i} = \text{aloc}(a') \wedge \hat{i} \notin \{\text{aloc}(a'') : a'' \in \text{dom}(\hat{h})\} \wedge \hat{i} \in^? \hat{d}]$. From (H9) and **??**, we have $\theta, s, \sigma_i \models \hat{\sigma}_i$. Because a' is a fresh abstract location, it's equality to one term can always be satisfied; and from (H10) we know $\hat{i} \in^? \hat{d}$ holds – thus $\mathsf{SAT}_{\theta,s}(\pi)$, completing our goal (G1).

Case **CPMapGetBotDomain:**

(H14) Assume $(h, d) = \mathit{unwrap}(\sigma) \wedge i \notin \text{dom}(h) \wedge d = \bot$.

(H15) From (H14), (H1) and CPMapGetBotDomain, we have $\sigma_i = \bot$.

(H16) It follows from (H1) and the definition of $\models$ that $\hat{\sigma} = (\hat{h}, \bot)$ such that $\hat{i} \notin \text{dom}(\hat{h})$.

(H17) The remainder of the proof is analogous to the above; three cases are possible depending on the result of `to_aloc` $\hat{i}$ – in all three cases, we can apply one of the four [...]NotFound and [...]New rules, such that $\hat{\sigma}_i = \bot$, $[\![\hat{i}']\!]_{\theta,s} = i$ and $\mathsf{SAT}_{\theta,s}(\pi)$, giving our goal (G1).

Proposition: Get UX Soundness
Assume

(H1) $\mathtt{get}(\hat{\sigma}, \hat{i}) \rightsquigarrow (\hat{i}', \hat{\sigma}_i, \pi) \wedge [\![\hat{i}]\!]_{\theta,s} = [\![\hat{i}']\!]_{\theta,s} = i \wedge \mathsf{SAT}_{\theta,s}(\pi)$

(H2) $\theta, s, \sigma \models \hat{\sigma}$

To prove

(G1) $\exists \sigma_i.\, \mathtt{get}(\sigma, i) = \sigma_i \wedge \theta, s, \sigma_i \models \hat{\sigma}_i$

We proceed by proving the property holds for all rules of the symbolic `get`, resulting in five cases.

Case **ALocPMapGetMatch:**

(H3) Assume $(\hat{h}, \hat{d}) = \mathit{unwrap}(\hat{\sigma}) \wedge a =$ `to_aloc` $\hat{i} \wedge a \neq \bot \wedge a \in \text{dom}(\hat{h}) \wedge \hat{\sigma}_i = \hat{h}(a)$

(H4) From (H3), (H1) and ALocPMapGetMatch, we have $\pi = []$

(H5) From (H3), we have $\hat{\sigma} \neq \bot$, thus from (H2) we have $\sigma = (h, d)$

(H6) From (H2), (H3) and the definition of $\models$, we have $i \in h$ such that $\sigma_i = h(i)$ and $\theta, s, \sigma_i \models \hat{\sigma}_i$

(H7) From (H6) we can apply CPMapGetMatch, thus $\texttt{get}(\sigma, i) = \sigma_i$. This completes our goal (G1).

Case ALocPMapGetNoMatchNotFound:

(H8) Assume $(\hat{h}, \hat{d}) = \mathit{unwrap}(\hat{\sigma}) \wedge a = \texttt{to\_aloc}\ \hat{i} \wedge a \neq \bot \wedge a \notin \mathrm{dom}(\hat{h}) \wedge m = \texttt{NO\_MATCH}$

(H9) From (H21), (H1) and ALocPMapGetNoMatchNotFound, we have $\hat{\sigma}_i = \bot$ and $\pi = [\mathrm{aloc}(a) \in^? \hat{d}]$

(H10) Let $(h, d) = \mathit{unwrap}(\sigma)$.

(H11) From (H1) $\mathsf{SAT}_{\theta,s}(\pi)$, thus from (H22), either $d \neq \bot \wedge i \in d$, or $d = \bot$.

(H12) From this, we get two cases: either $i \in \mathrm{dom}(h)$, or $i \notin \mathrm{dom}(h)$.

(H13) If $i \notin h$, we can apply either CPMapGetAdd or CPMapGetBotDomain depending on d, in both cases giving us $\texttt{get}(\sigma, i) = \bot$. Together with **??**, this completes our goal.

(H14) However, if $i \in \mathrm{dom}(h)$, we can only apply CPMapGetMatch, which however gives us $\texttt{get}(\sigma, i) = \sigma_i$ where $\sigma_i \neq \bot$ – this however *is not compatible with* $\hat{\sigma}_i$.

Case ALocPMapGetNoMatchNew:

(H15) Assume $(\hat{h}, \hat{d}) = \mathit{unwrap}(\hat{\sigma}) \wedge a = \texttt{to\_aloc}\ \hat{i} \wedge a = \bot \wedge a' = \texttt{fresh\_aloc ()} \wedge m = \texttt{NO\_MATCH}$

(H16) From (H26), (H1) and ALocPMapGetNoMatchNew, we have $\hat{i}' = \mathrm{aloc}(a')$, $\hat{\sigma}_i = \bot$ and $\pi = [\hat{i} = \mathrm{aloc}(a') \wedge \hat{i} \in^? \hat{d}]$

(H17) Let $(h, d) = \mathit{unwrap}(\sigma)$.

(H18) From here, we get two cases: either $i \in \mathrm{dom}(h)$, or $i \notin \mathrm{dom}(h)$ – as indeed it may be the case that $[\![\hat{i}]\!]_{\theta,s} = i$, but that due to the nature of $\texttt{to\_aloc}$, $a \notin \hat{h}$.

(H19) If $i \notin h$, we can apply either CPMapGetAdd or CPMapGetBotDomain depending on d, in either cases giving us $\texttt{get}(\sigma, i) = \bot$. Together with **??**, this completes our goal.

(H20) However, if $i \in \mathrm{dom}(h)$, we can only apply CPMapGetMatch, which however gives us $\texttt{get}(\sigma, i) = \sigma_i$ where $\sigma_i \neq \bot$ – this however *is not compatible with* $\hat{\sigma}_i$.

Case ALocPMapGetMatchNotFound:

(H21) Assume $(\hat{h}, \hat{d}) = \mathit{unwrap}(\hat{\sigma}) \wedge a = \texttt{to\_aloc}\ \hat{i} \wedge a \neq \bot \wedge a \notin \mathrm{dom}(\hat{h}) \wedge m = \texttt{MATCH}$

(H22) From (H21), (H1) and ALocPMapGetNoMatchNotFound, we have $\hat{\sigma}_i = \bot$ and $\pi = [\hat{i} \notin \{\mathrm{aloc}(a') : a' \in \mathrm{dom}(\hat{h})\} \wedge \hat{i} \in^? \hat{d}]$

(H23) Let $(h, d) = \mathit{unwrap}(\sigma)$.

(H24) From (H1) $\mathsf{SAT}_{\theta,s}(\pi)$, thus from (H22), either $d \neq \bot \wedge i \in d$, or $d = \bot$. We also have, crucially, that $i \notin \mathrm{dom}(h)$.

(H25) We can thus apply either CPMapGetAdd or CPMapGetBotDomain depending on d, in both cases giving us $\texttt{get}(\sigma, i) = \bot$. Together with **??**, this completes our goal.

Case ALocPMapGetMatchNew**:**

(H26) Assume $(\hat{h}, \hat{d}) = \mathit{unwrap}(\hat{\sigma}) \wedge a = \texttt{to\_aloc}\ \hat{i} \wedge a = \bot \wedge a' = \texttt{fresh\_aloc ()} \wedge m = \texttt{MATCH}$

(H27) Let $(h, d) = \mathit{unwrap}(\sigma)$.

(H28) From (H26), (H1) and ALocPMapGetNoMatchNew, we have $\hat{i}' = \mathrm{aloc}(a')$, $\hat{\sigma}_i = \bot$ and $\pi = [\hat{i} = \mathrm{aloc}(a') \wedge \hat{i} \notin \{\mathrm{aloc}(a'') : a'' \in \mathrm{dom}(\hat{h})\} \wedge \hat{i} \in^? \hat{d}]$

(H29) From (H1) $\texttt{SAT}_{\theta,s}(\pi)$, thus from (H28), either $d \neq \bot \wedge i \in d$, or $d = \bot$. We also have, crucially, that $[\![\mathrm{aloc}(a')]\!]_{\theta,s} = i$, and thus $i \notin \mathrm{dom}(h)$.

(H30) From (H29), we can apply either CPMapGetAdd or CPMapGetBotDomain depending on d, in either cases giving us $\texttt{get}(\sigma, i) = \bot$. Together with **??**, this completes our goal.

Case ALocPMapMatching**:**

(H31) Assume $(\hat{h}, \hat{d}) = \mathit{unwrap}(\hat{\sigma}) \wedge a = \texttt{to\_aloc}\ \hat{i} \wedge a \neq \bot \wedge a \notin \mathrm{dom}(\hat{h}) \wedge a' \in \mathrm{dom}(\hat{h}) \wedge \hat{\sigma}_i = \hat{h}(a')$

(H32) From (H31), (H1) and ALocPMapMatching, we have $\hat{i}' = \mathrm{aloc}(a')$, $\hat{\sigma}_i = \hat{h}(a')$, $\pi = [\mathrm{aloc}(a) = \mathrm{aloc}(a')]$ and $m = \texttt{MATCH}$.

(H33) From (H2), (H31) and the definition of $\models$, we have $i \in h$ such that $\sigma_i = h(i)$ and $\theta, s, \sigma_i \models \hat{\sigma}_i$

(H34) From (H33) we can apply CPMapGetMatch, thus $\texttt{get}(\sigma, i) = \sigma_i$. This completes our goal (G1).

Case ALocPMapMatchingBot**:**

(H35) Assume $(\hat{h}, \hat{d}) = \mathit{unwrap}(\hat{\sigma}) \wedge a = \texttt{to\_aloc}\ \hat{i} \wedge a = \bot \wedge a' \in \mathrm{dom}(\hat{h}) \wedge \hat{\sigma}_i = \hat{h}(a')$

(H36) From (H35), (H1) and ALocPMapMatchingBot, we have $\hat{i}' = \mathrm{aloc}(a')$, $\hat{\sigma}_i = \hat{h}(a')$, $\pi = [\hat{i} = \mathrm{aloc}(a')]$ and $m = \texttt{MATCH}$.

(H37) From (H2), (H35) and the definition of $\models$, we have $i \in h$ such that $\sigma_i = h(i)$ and $\theta, s, \sigma_i \models \hat{\sigma}_i$

(H38) From (H37) we can apply CPMapGetMatch, thus $\texttt{get}(\sigma, i) = \sigma_i$. This completes our goal (G1).

$\square$