Verifying Correct Usage of Atomic Blocks and Typestate: Technical Companion

Nels E. Beckman† Jonathan Aldrich†

August 2008
CMU-ISR-08-126

Institute for Software Research
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

†Institute for Software Research, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA

This work was supported by a University of Coimbra Joint Research Collaboration Initiative, DARPA grant #HR00110710019, Army Research Office grant #DAAD19-02-1-0389 entitled “Perpetually Available and Secure Information Systems”, the Department of Defense, and the Software Industry Center at CMU and its sponsors, especially the Alfred P. Sloan Foundation.
In this technical report, we present a static and dynamic semantics as well as a proof of soundness for a programming language presented in the paper entitled, Verifying Correct Usage of Atomic Blocks and Typestate [1]. The proof of soundness consists of a proof of preservation, which shows that well-typed expressions evaluate to other well-typed expressions, and a proof of progress, which shows that well-typed expressions are either values or can take an evaluation step in the dynamic semantics. The notion of progress is complicated by a specific notion of a well-typed heap, which ensures that only one reference in the entire thread-pool can know the exact state of an object of share or pure permission.
Keywords: transactional memory, typestate, proof
Abstract

In this technical report, we present a static and dynamic semantics as well as a proof of soundness for a programming language presented in the paper entitled, Verifying Correct Usage of Atomic Blocks and Typestate [1]. The proof of soundness consists of a proof of preservation, which shows that well-typed expressions evaluate to other well-typed expressions, and a proof of progress, which shows that well-typed expressions are either values or can take an evaluation step in the dynamic semantics. The notion of progress is complicated by a specific notion of a well-typed heap, which ensures that only one reference in the entire thread-pool can know the exact state of an object of share or pure permission.
1 Proof of Soundness

Our soundness criterion is as follows: It is either the case that all of the threads in a program are values, or their exists one thread such that the expression this thread is evaluating is well-typed and can take a step to another well-typed expression. If one of the threads in the thread-pool is currently executing within a transaction, then that thread must step, and if no threads in the thread-pool are currently executing within a transaction, then any threads that is not a value must be able to step. The dynamic semantics track typestate, and there is no evaluation rule to allow a method call if method preconditions are not met. In order to prove that method preconditions are always met for well-typed programs, our store typing judgment requires the invariant that only one thread can pinpoint the state of a share or pure object at a time.

The language of proof differs from the language used in the paper in a few ways. We have restored the original effects system used by Bierhoff and Aldrich [2]. This system was removed from the paper for purposes of clarity. The effects system keeps track of the fields that are modified in a subexpression to ensure that only the fields of the unpacked object are modified, and no field permissions “escape” beyond the packing of that object. Otherwise, our proof language resembles their proof language in most ways. As it was for Bierhoff and Aldrich, we have simplified the language of proof by removing linear disjunction ($\oplus$) and additive conjunction ($\&$). In the paper, an object is known to be unpacked if there is a $\text{unpacked}(k, s)$ permission inside of the linear context. In the system shown here, we use a separate context $u$. One $u$ appears on the left-hand side of the judgment. This shows us which object is unpacked before the expression takes an evaluation step. The other $u$ appears on the right-hand side, and shows us which object is unpacked after the expression has finished an evaluation step.

For the majority of the proof, things proceed much as they did in the proof of soundness presented by Bierhoff and Aldrich [2] with many of the multi-threaded features coming from Moore and Grossman [3]. Our system is different in a few ways. In Bierhoff and Aldrich the stack permissions, that is the dynamic representation of permissions that are currently available for use by the evaluating expression, were actually stored inside the heap. Because we have many threads, we have a separate environment $S_p$ attached to each thread expression which holds these stack permissions. Additionally, when typing a pool of threads, $T$ (essentially a list of expressions and their stack permissions), we associate each with their own linear context $\Delta$ and incoming unpacking flag $u$. We often must refer to the entire collection of linear contexts and packing flags, and this will usually be written $\overline{\Delta}$ and $\overline{u}$. Keep in mind that each linear context and unpacking flag is associated with one specific thread. This would most accurately be written as a list of tuples except that our $\Delta$ and $u$ usually appear on the left-hand side of the rule, while the thread itself will appear on the right-hand side, and so treating them as a tuple would be notationally awkward.

When type-checking the top-level thread pool, the members of $\overline{\Delta}$ and $\overline{u}$ are tagged with an additional bit of information, and are written $\overline{\Delta^e}$ and $\overline{u^e}$. At most one $\Delta$ and $u$ pair are allowed to contain specific state information about pure and share permissions. If this is the case, that $\Delta$ and $u$ will be tagged with $\text{wt}$, whereas others may not be. The fact that at most one linear context and unpacking flag is allowed to contain state information about share and pure permissions is checked by the $a; (\overline{\Delta^e}, \overline{u^e}) \text{ ok}$ judgment.
### 1.1 Proof Language

**program** \( \text{PG} ::= (\text{CL}, e) \)

**class decls.** \( \text{CL} ::= \text{class } C \{ \text{F I N M} \} \)

**methods** \( \text{M} ::= C_r m(C \? x) : P_1 \rightarrow \exists \text{result} : C_r.P_2 = e \)

**terms** \( t ::= x, y, z \mid o \)

**expressions** \( e ::= k \cdot t \mid k \cdot t.f \mid t_1.f := k \cdot t_2 \)
\( \mid \text{new } C(k \cdot t) \mid k \cdot t.m(k \cdot t) \)
\( \mid \text{inatomic } (e) \)
\( \mid \text{let } x = e_1 \text{ in } e_2 \)
\( \mid \text{spawn } (k \cdot t.m(k \cdot t)) \mid \text{atomic } e \)
\( \mid \text{unpack}_k k \cdot t@s \text{ in } e \mid \text{pack } t \text{ to } s' \text{ in } e \)

**expression types** \( E ::= \exists x : C.P \)

**I ::= \text{init}(\exists f : C.P, s) \)

**atomic** \( \mathcal{E} ::= \text{wt} \mid \text{ot} \mid \text{emp} \)

**states** \( S ::= s \mid \text{unpacked}(k) \mid \text{unpacked}(s) \)

**Predicates** \( P ::= k \cdot r@\$ \mid P_1 \odot P_2 \)
\( \$ ::= s \mid ? \)
\( N ::= s = P \)

**valid contexts** \( \Gamma ::= \cdot \mid \Gamma, x : C \)

**linear contexts** \( \Delta \mathcal{E} ::= \cdot \mid \Delta \mathcal{E}, P \)

**stores** \( \Sigma ::= \cdot \mid \Sigma, o : C \)

**heaps** \( H ::= \cdot \mid H, o \mapsto C(f = k \cdot o)@S \)
\( k ::= \text{full} \mid \text{pure} \mid \text{share} \mid \text{immutable} \mid \text{unique} \)
\( u ::= - \mid k \cdot t@s \)
\( \omega ::= \emptyset \mid \{ t.f \} \mid \omega_1 \cup \omega_2 \)
### 1.2 Judgment Forms

<table>
<thead>
<tr>
<th>Judgment</th>
<th>Judgment form</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Top-Level Evaluation</td>
<td>( a; H; T \to a'; H'; T' )</td>
<td>Under transaction state ( a ) and heap ( H ) the thread-pool ( T ) evaluates to ( T' ), which may modify an expression and add a new expression, while possibly modifying the heap ( H' ) and changing the transaction state ( a' ).</td>
</tr>
<tr>
<td>Expr. Evaluation</td>
<td>( a; H; \langle e, S_p \rangle \to a'; H'; \langle e', S_p' \rangle; T )</td>
<td>In heap ( H ), with transaction state ( a ) and stack permissions ( S_p ), the expression ( e ) takes a step to ( e' ), potentially modifying each and potentially adding a new thread.</td>
</tr>
<tr>
<td>Expression typing</td>
<td>( \Gamma; \Sigma; \Delta; \mathcal{E}; u \vdash e : E \setminus \omega\mid u' )</td>
<td>In variable context ( \Gamma ), store ( \Sigma ), linear context ( \Delta ), transaction effect ( \mathcal{E} ), and unpacking flag ( u ), expression ( e ) has type ( E ) and may assign to fields in ( \omega ) and changes unpacking to ( u' ).</td>
</tr>
<tr>
<td>Store typing (definition 1.5.1)</td>
<td>( \Sigma; \Delta\mathcal{E}; u\mathcal{E} \vdash H; S_p )</td>
<td>In store context ( \Sigma ) with lists of linear contexts ( \Delta\mathcal{E} ) and packing flags ( u\mathcal{E} ), each tagged with a transaction effect, the heap ( H ) and the list of all stack permissions ( S_p ) is well-typed.</td>
</tr>
<tr>
<td>Linear logic entailment (figure 6)</td>
<td>( \Gamma; \Sigma; \Delta \vdash P )</td>
<td>In variable context ( \Gamma ) and store ( \Sigma ), linear context ( \Delta ) proves ( P ).</td>
</tr>
<tr>
<td>Runtime property check (definition 1.5.2)</td>
<td>( H; S_p [k \cdot o] \vdash P )</td>
<td>Heap ( H ) with stack permissions ( S_p ) restricted to stack permissions ( k \cdot o ) satisfies property ( P ).</td>
</tr>
</tbody>
</table>

\[
\mathcal{E} = \text{ot|emp} \quad \Gamma; \Sigma; \Delta \vdash P
\]
\[
k \cdot o \notin \Delta, P \text{ where } k = \text{pure|share} \quad \Gamma; \Sigma; \Delta \vdash P
\]
\[
\Gamma; \Sigma; \Delta \vdash P \quad \Gamma; \Sigma; \Delta \vdash P
\]

\[
\Gamma; \Sigma; \Delta \vdash P
\]

Figure 1: Transaction-aware linear judgement

### 1.3 Thread Pool and Expression Typing

3
Figure 2: Well-formedness of all linear contexts.
\[
\text{not-active}(e)
\]

\[
\text{not-active}(\text{mbody}(C, m) = \overline{x}.e_m)
\]

\[
\text{not-active}(e_m)
\]

\[
\text{not-active}(k \cdot t)
\]

\[
\text{not-active}(k \cdot t.f)
\]

\[
\text{not-active}(t_1.f := k \cdot t_2)
\]

\[
\text{not-active}(\text{new } C(k \cdot t))
\]

\[
\text{not-active}(k \cdot t.m(k \cdot t))
\]

\[
\text{not-active}(e_1) \quad \text{not-active}(e_2)
\]

\[
\text{not-active}(\text{let } x = e_1 \text{ in } e_2)
\]

\[
\text{not-active}(\text{unpack}_E k \cdot t@s \text{ in } e)
\]

\[
\text{not-active}(e)
\]

\[
\text{not-active}(\text{pack}_E t \text{ to } s \text{ in } e)
\]

\[
\text{not-active}(e)
\]

\[
\text{not-active}(\text{atomic } e)
\]

Figure 3: Expressions with no active subexpressions.

\[
\text{active}(e)
\]

\[
\text{active}(e_1) \quad \text{not-active}(e_2)
\]

\[
\text{active}(\text{inatomic } e) \quad \text{active}(\text{let } x = e_1 \text{ in } e_2)
\]

Figure 4: Expressions with an active subexpression.

\[
\text{forget}(P) = P'
\]

\[
k = \text{immutable}|\text{unique}|\text{full} \quad k = \text{pure}|\text{share}
\]

\[
\text{forget}(k \cdot o@s) = k \cdot o@s \quad \text{forget}(k \cdot o@s) = k \cdot o@s'
\]

\[
\text{forget}(P_1) = P'_1 \quad \text{forget}(P_2) = P'_2
\]

\[
\text{forget}(P_1 \otimes P_2) = P'_1 \otimes P'_2
\]

Figure 5: The forget judgement.
\[
\text{forget}_{\mathcal{E}}(P) = P'
\]

\[
\begin{array}{c c}
\mathcal{E} = \text{wt} & \text{forget}_{\mathcal{E}}(P) = P \\
\mathcal{E} \neq \text{wt} & \text{for} \text{get}(P) = P'
\end{array}
\]

\[
\begin{array}{c}
\text{writes}(k) \\
\text{writes(uniq} \text{ue)} \quad \text{writes}(\text{full}) \quad \text{writes}(\text{sh} \text{are})
\end{array}
\]

\[
\begin{array}{c}
\text{readonly}(k) \\
\text{readonly}(\text{pure}) \quad \text{readonly}(\text{im} \text{mutable})
\end{array}
\]

\[
S \leq S'
\]

\[
\begin{array}{c c}
S \leq S' & S \leq S \\
S \leq S & S \leq ?
\end{array}
\]

\[
\text{unpacked}(s) \leq s \quad s \leq \text{unpacked}(s)
\]

\[
\begin{array}{c}
k \leq k'
\end{array}
\]

\[
\begin{array}{c}
k \cdot o@s \Rightarrow k' \cdot o@s \\
k \cdot o@s \Rightarrow k' \cdot o@s \otimes k'' \cdot o@s \\
k \leq k'
\end{array}
\]

6
Figure 6: Linear logic for permission reasoning

\[ \Gamma; P \vdash P \] LinHyp

\[ \Gamma; \Delta \vdash P_1 \quad \Gamma; \Delta \vdash P_2 \]
\[ \Gamma; (\Delta_1, \Delta_2) \vdash P_1 \otimes P_2 \] \( \otimes I \)

\[ \Gamma; \Delta \vdash 1 \] 1I

\[ \Gamma; \Delta \vdash P_1 \quad \Gamma; \Delta \vdash P_2 \]
\[ \Gamma; \Delta \vdash P_1 \& P_2 \] \( \& I \)

\[ \Gamma; \Delta \vdash \top \] \( \top I \)

\[ \Gamma; \Delta \vdash P_1 \]
\[ \Gamma; \Delta \vdash P_2 \]
\[ \Gamma; \Delta \vdash P_1 \oplus P_2 \] \( \oplus I_L \)

\[ \Gamma; \Delta \vdash P_1 \oplus P_2 \]
\[ \Gamma; \Delta \vdash P_2 \] \( \oplus I_R \)

no \( \top \) elimination

\[ \Gamma; \Delta \vdash \top \]
\[ \Gamma; (\Delta', P_1) \vdash P \]
\[ \Gamma; (\Delta', P_2) \vdash P \]
\[ \Gamma; (\Delta, \Delta') \vdash P \] \( \oplus E \)

no 0 introduction

\[ (\Gamma, z : H); \Delta \vdash P \]
\[ (\Gamma, z : H); \Delta \vdash \forall z : H. P \] \( \forall I \)

\[ \Gamma \vdash h : H \]
\[ \Gamma; \Delta \vdash \forall z : H. P \]
\[ \Gamma; \Delta \vdash [h/z]P \] \( \forall E \)

\[ \Gamma; \Delta \vdash \exists z : H. P \]
\[ (\Gamma, z : H); (\Delta', P) \vdash P' \]
\[ \Gamma; (\Delta, \Delta') \vdash P' \] \( \exists E \)

Figure 7: Top-level typing rules

\[ \vdash a; H; T \]

\[ ; \Sigma; \overline{\Delta^2}; \overline{\overline{\alpha}}; \overline{\overline{\alpha}}; T \vdash H; S_p \]
\[ \Sigma; \overline{\Delta^2}; \overline{\overline{\alpha}}; \overline{\overline{\alpha}}; T \vdash T \]

\[ \text{correct-atomic}(a, T) \quad \text{where } T = \langle e, S_p, \rangle \]
\[ \vdash a; H; T \]
Figure 8: Well-typed thread-pool

\[
\begin{align*}
\Sigma; \Delta; \bar{u} &\vdash T \\
\Sigma; \Delta_1^\xi, \Xi_1; u_1 \vdash e_1 : E_1 \setminus \omega \| u &\quad \Sigma; \Delta_2^\xi, \ldots, \Delta_n^\xi; u_2, \ldots, u_n \vdash T
\end{align*}
\]

\[(o : C) \in \Sigma \quad \vdash \Sigma; \Delta \vdash \mathcal{P} \quad \text{T-Loc}
\]

\[
\begin{align*}
\text{readonly}(k_u) \implies \text{readonly}(k) &\quad \vdash \Sigma; \Delta; \Xi; u \vdash k \cdot o : \exists x : C \cdot [x/o]P \setminus \emptyset \| u \\
\text{localFields}(C) = \overline{f : C} &\quad \vdash \Sigma; \Delta \vdash \mathcal{P} \quad \text{T-READ}
\end{align*}
\]

\[
\begin{align*}
\text{localFields}(C") = \overline{f : C} \quad (o' : C') \in \Sigma &\quad \text{writes}(k') \\
\Sigma; \Delta \vdash \mathcal{P} &\quad \vdash \Sigma; \Delta' \vdash o' / x' \| P' \\
\Sigma; \Delta, \Delta'; \Xi; k' \cdot o' @ s' \vdash o' . f_i' := k \cdot o : \exists x' : C \cdot P' \otimes [o'. f_i / x]P \setminus \emptyset \| [o_i, f] | k' \cdot o' @ s' &\quad \text{T-ASSIGN}
\end{align*}
\]

\[
\begin{align*}
\Sigma; \Delta \vdash \mathcal{P} &\quad [\bar{f} / f]P \quad \overline{o : C} \subseteq \Sigma \quad \text{init}(C) = \langle \exists \bar{f} : C \cdot P, s \rangle \\
\vdash \Sigma; \Delta, \Delta'; \Xi; u \vdash \text{new}(C(k \cdot o)) : \exists x : C \cdot \text{unique} \cdot x @ s \setminus \emptyset \| u &\quad \text{T-NEW}
\end{align*}
\]

\[
\text{forget}_\mathcal{P}(k \cdot o @ s) = k \cdot o @ s \\
k = \text{immutable} \quad | \quad \text{pure} \implies s = s' \quad \vdash \Sigma; \Delta', k \cdot o @ s; \Xi; - \vdash e' : E \setminus \emptyset \setminus -
\]

\[
\text{localFields}(C) = \overline{f : C} \quad (o : C) \in \Sigma \quad \vdash \Sigma; \Delta \vdash [o / \text{this}] \text{inv}_{C}(s, k) \\
\text{No temporary permissions for o}\bar{f} \text{ in } \Delta' &\quad \text{T-PACK}
\]

\[
\begin{align*}
\vdash \Sigma; (\Delta, \Delta'); \Xi; k \cdot o @ s \vdash \text{pack } o \text{ to } s' \text{ in } e' : E \setminus \{ o \bar{f} \} &\quad \text{T-UNPACK}
\end{align*}
\]

\[
\begin{align*}
\vdash \Sigma; (\Delta, \Delta'); \Xi; k \cdot o @ s \vdash \text{unpack}_\mathcal{P} k \cdot o @ s \text{ in } e' : E \setminus \emptyset &\quad \text{T-UNPACK-WT}
\end{align*}
\]
\[
\begin{array}{c}
(o : C) \in \Sigma \quad o : C \subseteq \Sigma \\
\vdots \Sigma; \Delta \vdash_{\mathcal{E}} [o/this][\sigma/\emptyset]P \quad \text{mtype}(C, m) = \forall x : C. P \rightarrow \exists x : C. P_r \quad \text{forget}_{\mathcal{E}}(P_r) = P'_r \\
\vdots \Sigma; \Delta; \mathcal{E}; \vdash_{\mathcal{E}} \forall k \cdot o.m(k \cdot o) : \exists x : C. P'_r \setminus \emptyset \\
\end{array}
\]

<table>
<thead>
<tr>
<th>T-CALL</th>
</tr>
</thead>
</table>

\[
\begin{array}{c}
o : C \in \Sigma \quad o : C \in \Sigma \quad \text{mtype}(C, m) = \forall x : C. P \rightarrow E \\
\vdots \Sigma; \Delta; \mathcal{E}; \vdash_{\mathcal{E}} [o/this][\sigma/\emptyset]P \\
\vdots \Sigma; \Delta; \mathcal{E}; \vdash_{\mathcal{E}} \forall o.m(k \cdot o) : \exists x : C. P'_r \setminus \emptyset \\
\end{array}
\]

<table>
<thead>
<tr>
<th>T-SPAWN</th>
</tr>
</thead>
</table>

\[
\begin{array}{c}
\Sigma; \Delta; \mathcal{E}; \vdash_{\mathcal{E}} \forall k \cdot o.m(k \cdot o) : \exists x : C. \text{immutable} \cdot o_d \not\in s_d \setminus \emptyset \\
\vdots \Sigma; \Delta; \mathcal{E}; \vdash_{\mathcal{E}} \forall k \cdot o.m(k \cdot o) : \exists x : C. P'_r \setminus \emptyset \\
\end{array}
\]

<table>
<thead>
<tr>
<th>T-LET</th>
</tr>
</thead>
</table>

\[
\begin{array}{c}
\vdots \Sigma; \Delta; \mathcal{E}; \vdash_{\mathcal{E}} \forall x : T.P \setminus \omega_1 | u_2 \quad x : C; \Sigma; P'; \mathcal{E}; \vdash_{\mathcal{E}} \forall x : T.P \setminus \omega_2 | u' \\
\vdots \Sigma; (\Delta_1, \Delta_2); \mathcal{E}; \vdash_{\mathcal{E}} \forall x : T.P \setminus \omega_1 \cup \omega_2 | u' \\
\vdots \Sigma; \Delta; \mathcal{E}; \vdash_{\mathcal{E}} \forall x : T.P \setminus \omega | u' \\
\vdots \Sigma; \Delta; \mathcal{E}; \vdash_{\mathcal{E}} \forall x : T.P \setminus \omega | u' \\
\end{array}
\]

<table>
<thead>
<tr>
<th>T-INATOMIC</th>
</tr>
</thead>
</table>

\[
\begin{array}{c}
\vdots \Sigma; \Delta; \mathcal{E}; \vdash_{\mathcal{E}} \forall x : T.P \setminus \omega | u' \quad \text{forget}_{\mathcal{E}}(P) = P' \\
\vdots \Sigma; \Delta; \mathcal{E}; \vdash_{\mathcal{E}} \forall x : T.P \setminus \omega | u' \\
\end{array}
\]

<table>
<thead>
<tr>
<th>T-ATOMIC</th>
</tr>
</thead>
</table>

\[
\begin{array}{c}
\vdots \Sigma; \Delta; \mathcal{E}; \vdash_{\mathcal{E}} \forall x : T.P \setminus \omega | u' \quad \text{forget}_{\mathcal{E}}(P) = P' \\
\vdots \Sigma; \Delta; \mathcal{E}; \vdash_{\mathcal{E}} \forall x : T.P \setminus \omega | u' \\
\end{array}
\]
1.4 Dynamic Semantics

\[
\begin{array}{c}
a; H; T \rightarrow a'; H'; T' \\
\end{array}
\]

\[
\begin{array}{c}
a; H; e \rightarrow a'; H'; e'; T' \\
a; H; T_a, e, T_b \rightarrow a'; H'; T_a, e', T_b, T' \\
\end{array}
\]

Figure 9: Top-level Dynamic Semantics
\[ k = \text{pure} \mid \text{immutable} \quad o \mapsto C(\ldots, f_i = k' \cdot o')@\text{unpacked}(s'') \in H \]
\[ a; H; \langle k \cdot o.f_i, S_p \rangle \rightarrow a; H[o \mapsto C(\ldots, f_i = (k' - k) \cdot o)]; \langle k \cdot o', (S_p + k \cdot o') \rangle \]

\[ k \leq k' \quad o \mapsto C(\ldots, f_i = k' \cdot o')@\text{unpacked}(k'') \in H \]
\[ E - \text{Read-R} \]

\[ k \cdot o_1 \in S_p \quad o_1 \mapsto C(\ldots, f = k' \cdot o', \ldots)@\text{unpacked}(k'') \in H \]
\[ k_2 \cdot o_2 \in S_p \quad o_2 \mapsto C(\ldots)@S_2 \in H \]
\[ H' = H[o_1 \mapsto C(\ldots, f = k \cdot o_2, \ldots)@\text{unpacked}(k'')] \quad S'_p = S_p[(k_2 - k) \cdot o_2, k' \cdot o''] \]
\[ E - \text{Assign} \]

\[ H; S_p \vdash [\overline{E}]P \quad \text{init}(C) = (\exists E : C, P, S) \quad S'_p = S_P - k \cdot o_o \not\in \text{dom}(H) \]
\[ a; H; (\text{new } C(k \cdot o), S_p) \rightarrow a; H, o_n \mapsto C(f = k \cdot o)@s; (\text{unique } o_n, (S'_p, \text{unique } o_n)) \]

\[ E = \text{ot} \mid \text{emp} \quad k' \cdot o \in S_p \quad \text{readonly}(k) \quad o \mapsto C(\ldots)@S \in H \quad k \leq k' \]
\[ k = \text{mutable} \quad S = (\text{unpacked}(s)|s), k = \text{pure} \quad S = s \]
\[ E - \text{Unpack-R} \]

\[ o; H; (\text{unpack}_w k \cdot o@s \in e', S_p) \rightarrow \]
\[ o; H[o \mapsto C(\ldots)@\text{unpacked}(s)]; (e', S_p[(k' - k) \cdot o]) \]

\[ E - \text{Unpack-R-Wt} \]

\[ E = \text{ot} \mid \text{emp} \quad k' \cdot o \in S_p \quad \text{writes}(k) \quad o \mapsto C(\ldots)@s \in H \quad k \leq k' \]
\[ a; H; (\text{unpack}_w k \cdot o@s \in e', S_p) \rightarrow \]
\[ a; H[o \mapsto C(\ldots)@\text{unpacked}(k)]; (e', S_p[(k' - k) \cdot o]) \]

\[ E - \text{Unpack-R-Wt} \]

\[ \text{inv}_C(s) \text{ satisfied by } o \text{'s fields} \]
\[ k_o \cdot o \in S_p \quad o \mapsto C(f = k \cdot o)@\text{unpacked}(s) \in H \]
\[ a; H; (\text{pack } o \text{ to } s \text{ in } e', S_p) \rightarrow a; H[o \mapsto C(f = k \cdot o)@s]; (e', S_p) \]

\[ E - \text{Pack-R} \]

\[ \text{inv}_C(s) \text{ satisfied by } o \text{'s fields} \]
\[ k_o \cdot o \in S_p \quad o \mapsto C(f = k \cdot o)@\text{unpacked}(k) \in H \]
\[ a; H; (\text{pack } o \text{ to } s \text{ in } e', S_p) \rightarrow a; H[o \mapsto C(f = k \cdot o)@s]; (e', S_p[(k + k_o) \cdot o]) \]

\[ E - \text{Pack-R} \]
\[ \text{mbody}(C, m) = \overline{\text{e}}_m \quad \text{mtime}(C, m) = \forall x : C.P \rightarrow E \]

\[ a; H; (k \cdot o.m(k \cdot o), S_p) \rightarrow a; H; ([o/this][\overline{o}/\overline{x}]e_m, S_p); . \quad \text{E-CALL} \]

\[ \text{mbody}(C, m) = \overline{\text{e}}_m \quad \text{mtime}(C, m) = \forall x : C.P \rightarrow E \]

\[ o; H; (\text{spawn}(k \cdot o.m(k \cdot o)), (S_{p1}, S_{p2})) \rightarrow o; H; (o_d, S_{p1}); ([o/this][\overline{o}/\overline{x}]e_m, S_{p2}) \quad \text{E-SPAWN} \]

\[ a; H; (e_1, S_p) \rightarrow a'; H'; (e_1', S_{p'}); T \]

\[ a; H; (\text{let } x = e_1 \text{ in } e_2, S_p) \rightarrow a'; H'; (\text{let } x = e'_1 \text{ in } e_2, S_{p'}); T \quad \text{E-LET-E} \]

\[ k' \cdot o \in S_p \quad o \mapsto C(...) \overline{S} \in H \quad k \leq k' \]

\[ a; H; (\text{let } x = k \cdot o \text{ in } e_2, S_p) \rightarrow a; H; ([o/x]e_2, S_p); . \quad \text{E-LET-V} \]

\[ o; H; (\text{atomic}(e), S_p) \rightarrow \bullet; H; (\text{inatomic}(e), S_p); . \quad \text{E-ATOMIC-BEGIN} \]

\[ \bullet; H; (\text{inatomic}(k \cdot o), S_p) \rightarrow o; H; (k \cdot o, S_p); . \quad \text{E-ATOMIC-EXIT} \]

\[ a; H; (e, S_p) \rightarrow a'; H'; (e', S_{p'}); T \]

\[ \bullet; H; (\text{inatomic}(e), S_p) \rightarrow \bullet; H'; (\text{inatomic}(e'), S_{p'}); T \quad \text{E-INATOMIC} \]

\[ 1.5 \quad \text{Preservation} \]

\[ 1.5.1 \quad \text{Definition of Store Typing} \]

\[ \Sigma; \overline{\Delta e}; u^\overline{e} \vdash H; \overline{S_p} \]

The above judgement is true if:

1. \[ \Sigma; \overline{\Delta e} \vdash \overline{S_p} \]

2. \[ \Sigma; \overline{\Delta e}; \overline{S_p}; u^\overline{e} \vdash H \]
\[
\Sigma; \Delta^k \vdash S_p
\]

\[
\Sigma; \Delta^k \vdash S_{p1}, \Sigma; \Delta^k, \ldots, \Delta^k \vdash S_{p2}, \ldots, S_{pn}
\]

\[
\Sigma; \Delta^k, \Delta^k, \ldots, \Delta^k \vdash S_{p1}, S_{p2}, \ldots, S_{pn}
\]

\[
\Sigma; \Delta^k \vdash S_p
\]

\[
\{o|k \cdot o \mathbin{@}$ \in \Delta\} \subseteq \{o|k \cdot o \in S_p\} \quad \forall k \cdot o \in S_p, \Sigma; \Delta \vdash k' \cdot o \mathbin{@}$ \otimes \top \supset k' \leq k
\]

Where the above rule ignores permissions on fields.

The above judgement is true if:

1. \( \text{dom}(\Sigma) = \text{dom}(H) \)
2. \( a; (\Delta^k, \bar{u}^k) \) ok
3. \( \forall u^k \in \bar{u}^k, u^k = k \cdot o@s \supset E = \text{wt}(k = \text{immutable}|\text{unique}|\text{full} \text{ and } o \mapsto C(\ldots)@S \in H, \)
   where \( S = \text{unpacked}(s) \) if \( \text{readonly}(k) \) or \( S = \text{unpacked}(k) \) if \( \text{writes}(k) \). Also, \( o \notin \overline{u} \supset \)
   \( o \) is packed in \( H \) and \( \text{inv}_C(o, \text{unique}) \).
4. \( \forall o \in \text{dom}(\Sigma), \forall \Delta \in \overline{\Delta}: \text{if } o \mapsto C(f = k \cdot o)@S \in H \text{ then} \)
   (a) \( (o : C) \in \Sigma \)
   (b) Either \( S = \text{unpacked}(k) \) or \( S = \text{unpacked}(s) \) and \( [o/\text{this}]\text{inv}_C(s, \text{immutable}) \) is
       satisfied by \( o \)'s fields, or \( S = s \) and \( [o/\text{this}]\text{inv}_C(s, \text{unique}) \) is satisfied by \( o \)'s fields.
   (c) If \( \vdash; \Sigma; \Delta \vdash k \cdot o@s \otimes \top \text{ then } S \leq \$ \).
   (d) If \( \vdash; \Sigma; \Delta \vdash k'_i \cdot o, f_i@s \otimes \top, \text{ then } k'_i \leq k_i \) (and \( o = o_{unp} \) and \( o_i \mapsto C(o(\ldots))@s_o \in H \)
       and either \( S = \text{unpacked}(s) \), which implies \( \text{readonly}(k'_i) \), or \( S = \text{unpacked}(k') \).
       If \( S = \text{unpacked}(s) \) then \( \$ = s_o \text{ or } \$ = ? \).
   (e) \( \text{unique} \cdot o@s \in \Delta, u \supset k \cdot o@s \) not in any other \( \Delta \) or \( u \) in \( \overline{\Delta} \) or \( \overline{\pi} \). Also, \text{full} \cdot o@s \in \Delta, u \supset \text{full} \cdot o@s \text{ and } k \cdot o@s \) not in any other \( \Delta \) or \( u \) in \( \overline{\Delta} \) or \( \overline{\pi} \).
   (f) \( \text{immutable} \cdot o@s \in \Delta, u \supset (k \cdot o@s \in \overline{\Delta}, \overline{\pi} \supset k = \text{immutable} \& \$(s = s|\$ = ?) \))
   (g) Where \( k_i = \text{unique} \) implies \( k \cdot o_i \notin \overline{\Delta}, \overline{\pi} \supset k = \text{immutable} \) and \( k \cdot o_i@s \notin \overline{\Delta}, \overline{\pi} \) and where \( k_i = \text{immutable} \) and \( o \mapsto C(\ldots)@S \), where \( S = s|\text{unpacked}(s) \) implies \( k' \cdot o_i@s' \notin \overline{\Delta}, \overline{\pi} \), where \( k' \neq \text{immutable} | s' \neq s \).
1.5.2 Property Satisfied at Runtime

If

\[ \overline{o \mapsto C(\ldots)}@s \subseteq H \text{ and } k' \cdot o @ s \in S_p \]

\[ \cdot | o : C | k \cdot o @ s \vdash P \text{ (an instance of } \Gamma | \Sigma | \Delta \vdash P) \]

\[ k \leq k' \]

then \( H; S_p | k \cdot o \vdash P \)

1.5.3 Lemma: Compositionality

If \( \Sigma; \Delta_i \vdash H; S_p \) and \( \Delta_i = \Delta_{i1}, \Delta_{i2} \) then \( \Sigma; \Delta_{i1} \vdash H; S_p \) where \( \Delta_i \) is replaced with \( \Delta_{i1} \) and \( \Sigma; \Delta_{i2} \vdash H; S_p \) where \( \Delta_i \) is replaced with \( \Delta_{i2} \).

Proof: Immediate from the definition of store typing. We are always allowed to know less statically about permissions than what is true at run-time, so long as what we know statically is consistent with the run-time information.

1.5.4 Lemma: Packing Flag

If \( \Gamma; \Sigma; \Delta; \varepsilon; u \vdash e : E \setminus \omega | u' \) then either (a) \( u = - \) and \( \omega = \emptyset \) or (b) \( u = k \cdot t@s \) and \( \omega \) contains only fields of \( t \).

Proof: (a) \( u = - \) is not a valid precondition for producing effects (using assignment or packing).

(b) By induction on typing derivations, using (a). Only one object can be unpacked at a time, permission for unpacked object is needed for assignments and packing, and effect of `unpack` expression is \( \emptyset \).

1.5.5 Object Weight

\[ w(o, \Delta) = \Sigma_{k \cdot o \in \Delta} k, \text{ ignoring fields.} \]

\[ w(o, u) = k, \text{ if } u = k \cdot o@s, \text{ and } 0 \text{ otherwise.} \]

\[ w(o, S_p) = \Sigma_{k \cdot o \in S_p} k \]

Where:

\[ k + k' \]

is defined as:

- full + pure = full
- share + pure = share, share + share = share
- immutable + immutable = immutable
- pure + pure = pure
1.5.6 Preservation for Thread Pools

If

- $a; H; T \rightarrow a'; H'; T'$
- $\vdash a; H; T$

Then there exists

- $\Sigma' \supseteq \Sigma$
- $\Delta e'$
- $\omega e'$
- $\omega$

such that

- $\Sigma'; \Delta; \omega; \cdot \vdash e' : E \setminus \omega | u''$
- $\Sigma'; \Delta e' \vdash \omega e' ; H'; S_p'$, where $T' = < e', S_p'>$.
- $\text{correct-atomic}(a', T')$
- $\forall o \in \text{dom}(H) : w(o, S_p) - w(a, \Delta) - w(a, u) \leq w(o, S_p') - w(a, \Delta') - w(o, u')$, for each $\Delta$ in $\Delta$, $S_p$ in $S_p'$, $\Delta$ in $\Delta'$, and $S_p$ in $S_p'$.

**Proof:** By structural induction on the derivation of $a; H; T \rightarrow a'; H'; T'$.

**Case Top-Level**

$\vdash a; H; T$ Assumption
$a; H; e \rightarrow a'; H'; e'; T'$ where $T_a || e || T_b$ Inversion of only eval rule.

$\vdash; \Sigma; \Delta e' ; H; S_p$ Inversion of only typing rule. $\vdash; \Sigma; \Delta; \omega; \cdot; u \vdash e : E \setminus \omega | u''$
$\Sigma; \Delta e' ; \cdot \vdash T$ From well-typed thread pool.

Correct atomicity ($a, T$) where $T = \langle e, S_p \rangle$

Invoke preservation for single threads.

$\Sigma', \Delta e', \omega', a', \cdot$ s.t.

$\vdash; \Sigma'; \Delta; \omega; \cdot; u' \vdash e' : E \setminus \omega' | u''$

Single-threaded lemma.

If $T \neq \cdot$
\[ \vdash \Sigma'; \Delta^o; \text{ot}; - \vdash e_t : E_t \setminus \omega | u'' \]  

Single-threaded lemma.

\[ \Sigma; \overline{\Delta^e}; \overline{u^e} \vdash H; \overline{S'_p} \]  

Single-threaded lemma.

\[ \forall o \in \text{dom}(H) : w(o, S_p) - w(o, \Delta) - w(o, u) \leq w(o, S'_p) - w(o, \Delta') - w(o, u'), \]  

for each \( \Delta \) in \( \overline{\Delta} \), \( S_p \) in \( \overline{S_p} \), \( \Delta \) in \( \overline{\Delta'} \), and \( S_p \) in \( \overline{S'_p} \)  

Single-threaded lemma.

\( \text{not-active}(T') \) by single-threaded lemma.

If \( a = \circ \) implies \( \text{not-active}(T) \). If \( a' = \bullet \), then by single-threaded lemma \( \text{active}(e') \). If \( a' = \circ \) the by single-threaded lemma \( \text{not-active}(e') \). Thus, \( \text{correct-atomic}(a', T') \).

If \( a = \bullet \) and \( \text{active}(e) \) implies \( \text{not-active}(T_a || T_b) \). If \( a' = \bullet \) then by single-threaded lemma \( \text{active}(e') \). If \( a' = \circ \) then by the single-threaded lemma \( \text{not-active}(e') \). Thus, \( \text{correct-atomic}(a', T') \).

If \( a = \bullet \) and \( \text{not-active}(e) \) implies \( \text{active}(T_a) \) or \( \text{active}(T_b) \). Only one may be active but neither will change during \( e \)'s step, so \( a' = \bullet \). Single-threaded lemma gives us \( \text{not-active}(e') \). Thus, \( \text{correct-atomic}(a', T') \).

### 1.5.7 Preservation for Single Threads

If

- \( \Sigma; \Delta; E; u \vdash e : E \setminus \omega | u'' \)
- \( \Sigma; \overline{\Delta^e}; \overline{u^e} \vdash H; \overline{S'_p} \), where \( \overline{\Delta^e} = (\Delta_1, \Delta^*_1)^E, (\Delta_2, \Delta^*_2)^E, \ldots (\Delta_n, \Delta^*_n)^E \), where \( \Delta^*_i \) contains extra permissions that contain no temporary state information and no permissions for fields in \( \omega \).
- \( a; H; \langle e, S_p \rangle \rightarrow a'; H'; \langle e', S'_p \rangle; T \)
- And exactly one of the following:
  - \( a = \circ \) and \( \text{not-active}(e) \)
  - \( a = \bullet \) and \( \text{not-active}(e) \)
  - \( a = \bullet \) and \( \text{active}(e) \)

Then there exists

- \( \Sigma' \supseteq \Sigma \)
- \( u' \) tagged with \( E \), written \( u'E' \).
- \( \omega' \), where either (a) \( a; H; \langle e, S_p \rangle \rightarrow a'; H'; \langle e', S'_p \rangle; T \) unpacks an object \( o \), i.e., \( u^E = - \) and \( u'E' = k \cdot o@s \) and \( \omega' - \omega \) only mentions fields of \( o \), or (b) \( \omega' \subseteq \omega \).
• \( \Delta' \) tagged with \( \mathcal{E} \), written \( \Delta^{\mathcal{E}'} \).
• \( S_{pt}, \Delta^{\mathcal{E}'}_t \) and \( u^{\mathcal{E}}_t \).

such that

• \( T \) is either \( e_t \) or \( \circ \).
• \( \vdash \Sigma'; \Delta'; \mathcal{E}; u' \vdash e' : E \setminus \omega' \mid u'' \)
• \( \Sigma; \Delta^{\mathcal{E}}; u^{\mathcal{E}} \vdash H; S_p \), where \( \overline{\Delta} \) and \( \overline{S_p} \) are \( \Delta \) and \( S_p \) with \( (\Delta', \Delta^\ast) \) swapped for \( (\Delta, \Delta^\ast) \) and \( S'_p \) swapped for \( S_p \) (and including \( S_{pt} \) and \( \Delta_t \) if \( T = e_t \)).
• \( \forall o \in \text{dom}(H) : w(o, S_p) - w(o, \Delta) - w(o, u) \leq w(o, S'_p) - w(o, \Delta') - w(o, u'), \) for each \( \Delta \) in \( \overline{\Delta} \), \( S_p \) in \( \overline{S_p} \), \( \Delta \) in \( \overline{\Delta} \), and \( S_p \) in \( \overline{S_p} \).
• If \( T = e_t \) then \( \vdash \Sigma'; \Delta_t; \mathcal{E}; - \vdash e_t : E_t \setminus \omega - \)

As well as all of the following, although exactly one will not be vacuous:

- if \( a = a' \) and not-active\( (e) \) then not-active\( (e') \)
- if \( a = a' \) and active\( (e) \) then active\( (e') \)
- if \( a = \circ \) and \( a' = \bullet \) and not-active\( (e) \) then active\( (e') \)
- if \( a = \bullet \) and \( a' = \circ \) and active\( (e) \) then not-active\( (e') \)

**Proof:** By structural induction on the derivation of \( a; H; \langle e, S_p \rangle \rightarrow a'; H'; \langle e', S'_p \rangle; T \).

**Case E-Unpack-RW-Wt**

So \( e = \text{unpack}\_\text{wt} \ k \cdot o@s \) in \( e_2 \), \( e' = e_2 \), \( a = a' = o \), \( H' = H[o \mapsto C(\ldots)@\text{unpacked}(k)] \), \( S'_p = S_p[(k' - k) \cdot o] \) and \( T = \cdot \).

\[
\vdash \Sigma; \Delta; \mathcal{E}; u \vdash e : E \setminus \omega \mid u'' \\
\Sigma; \Delta^{\mathcal{E}}; u^{\mathcal{E}} \vdash H; \overline{S_p} \\
\text{writes}(k) \quad o \mapsto C(\ldots)@s \in H \quad k' \cdot o \in S_p \; k \leq k'
\]

Assumption

Inversion of only eval rule

Let \( \Sigma' = \Sigma, \quad \Delta^{\mathcal{E}} = \Delta_2, [o/\text{this}]\text{inv}_C(s, k); \text{wt}; k \cdot o@s \vdash e_2 : E \setminus \omega_2 - \\
\vdash \Sigma; \Delta^{\mathcal{E}}; u^{\mathcal{E}} \vdash H; \overline{S_p} \quad \text{Inversion of only typing rule} \quad u^{\mathcal{E}} = k \cdot o@s, \quad \omega' = \omega_2.

\[
\vdash \Sigma'; \Delta^{\mathcal{E}}; u^{\mathcal{E}} \vdash H; \overline{S_p} \\
\text{We have removed } k \cdot o \text{ from } \Delta \text{ and } S_p, \text{ and added field perms to } \Delta \text{ which are ignored.}
\]

\[
\Sigma'; \overline{\Delta} \vdash \overline{S_p}
\]
No other $\Delta$ or $S_p$ changed.

Must show $\Sigma'; \overline{\Delta e'}; S'_p; u e' \vdash H'$

1.) ok

No change to $\text{dom}(\Sigma)$ or $\text{dom}(H)$

2.) ok

$\Delta_{w'}$ and $u_{w'}$ were and remain the only $wt$ elements.

3.) ok

For $u_{w'}$, $E = \text{wt}$. $o \mapsto C(\ldots)@\text{unpack}(k) \in H'$ and writes($k$).

4.a.) ok

No change

4.b.) ok

$S = \text{unpack}(k)$

4.c.) ok

No new stack perms in $\Delta'$.

4.d.) ok

4.b. was true before step. Fields added to $\Delta'$ are given by $\text{inv}_C(s, k)$.

4.e.) ok

4.g. was true before step. Any unique or full fields cannot be in other $\Delta s$ and $u$.

4.f.) ok

4.g. was true before step. Other permissions to fields must agree with state.

4.g.) ok

No fields altered.

For $o$ was unpacked.

$\omega' - \omega = \omega$ only contains fields of $o$.

$\forall o \in \text{dom}(H) : w(o, S_p) - w(o, \Delta) - w(o, u) \leq w(o, S'_p) - w(o, \Delta') - w(o, u')$

Net is unchanged. Permission moved from $\Delta$ to $u'$.

$T = $.

$\text{not-active}(\text{unpack})$ implies $\text{not-active}(e_2)$  Inversion of $\text{not-active}$.

$\text{active}(\text{unpack})$ cannot be derived.

$a = a'$ and $\text{not-active}(e)$ implies $\text{not-active}(e')$  Above

**Case E-Unpack-RW**

So $e = \text{unpack}_E k \cdot o@s$ in $e_2$, $e' = e_2$, $a = a'$, $H' = H[o \mapsto C(\ldots)@\text{unpack}(k)]$, $S'_p = S_p[(k' - k) \cdot o]$ and $T = $.

$\vdash \Sigma; \Delta; \epsilon; u \vdash e : E \setminus \omega|u''$

Assumption

$\Sigma; \overline{\Delta e}; u \overline{e} \vdash H; S_p$

Assumption

writes($k$) $o \mapsto C(\ldots)@s \in H$ $k' \cdot o \in S_p$ $k \leq k'$ $E = \text{ot} \text{| emp}$

Inversion of only eval rule

$\vdash \Sigma; \Delta e = (\Delta_1, \Delta_2)$ $u = u'' = - \omega = \emptyset$

$\vdash \Sigma; \Delta_1 \vdash k \cdot o@s u = u'' = - \omega = \emptyset$

$k = \text{unique}$|$\text{full}$|$\text{immutable}$ $\vdash \Sigma; \Delta_2, [o/\text{this}]i\text{nv}_C(s, k); \epsilon; k \cdot o@s \vdash e_2 : E \setminus \omega_2|-$

Inversion of only typing rule

Let $\Sigma' = \Sigma$, $\Delta e' = \Delta_2, [o/\text{this}]i\text{nv}_C(s, k), u e' = k \cdot o@s, \omega' = \omega_2$.

$\vdash \Sigma'; \Delta e'; \epsilon; u' \vdash e' : E \setminus \omega'|-$

Substitution

Must show $\Sigma; \overline{\Delta e}; u \overline{e} \vdash H; S_p$

$\Sigma'; (\Delta e', \Delta^*) \vdash S'_p$

We have removed $k \cdot o$ from $\Delta$ and $S_p$, and added field perms to $\Delta$ which are ignored.

$\Sigma'; \overline{\Delta} \vdash S'_p$

No other $\Delta$ or $S_p$ changed.

Must show $\Sigma'; \overline{\Delta e'}; S'_p; u e' \vdash H'$

1.) ok

No change to $\text{dom}(\Sigma)$ or $\text{dom}(H)$
2.) ok We have not changed the number of \( \text{wt} \) elements from before.
If \( E \neq \text{wt} \), then not-\( \text{wt}(\Delta') \) because invariants cannot contain pure and shared information.

3.) ok \( k = \text{immutable}\|\text{full}\|\text{unique}. o \mapsto C(\ldots)@\text{unpack}(k) \in H' \) and writes(\( k \)).

4.a.) ok No change
4.b.) ok \( S = \text{unpack}(k) \)
4.c.) ok No new stack perms in \( \Delta' \).
4.d.) ok 4.b. was true before step. Fields added to \( \Delta' \) are given by \( \text{inv}_C(s, k) \).
4.e.) ok 4.g. was true before step. Any unique or full fields cannot be in other \( \Delta s \) and \( u \).
4.f.) ok 4.g. was true before step. Other permissions to fields must agree with state.
4.g.) ok No fields altered.

\( o \) was unpacked.
\( u = - \) and \( u' = k \cdot o@s. \)

\( \omega' = \omega \) only contains fields of \( o \).
packing flag lemma
\( \forall o \in \text{dom}(H) : w(o, S_p) - w(o, \Delta) - w(o, u) = w(o, S'_p) - w(o, \Delta') - w(o, u') \)
Net is unchanged. Permission moved from \( \Delta \) to \( u' \).

\[ T = . \]

not-active(unpack) implies not-active(\( e_2 \))
Inversion of not-active.

active(unpack) cannot be derived.
a = \( a' \) and not-active(\( e \)) implies not-active(\( e' \))
Above

\section*{CASE E-UNPACK-R}
So \( e = \text{unpack}_E k \cdot o@s \) in \( e_2 \), \( e' = e_2 \), \( a = a' \), \( H' = H[o \mapsto C(\ldots)@\text{unpack}(s)] \), \( S'_p = S_p[(k' - k) \cdot o] \) and \( T = . \).

\[ ;; \Sigma; \Delta; E; u \vdash e : E \setminus \omega|u'' \]
Assumption
\[ \Sigma; \Delta; \overline{\Delta'}; u \vdash H; \overline{S_p} \]
Assumption
\[ E = \text{ot} | \text{emp} k' \cdot o \in S_p \text{ readonly}(k) o \mapsto C(\ldots)@S \in H \ k \leq k' \]
\[ k = \text{immutable} \supset S = (\text{unpack}(s)|s), k = \text{pure} \supset S = s \]
Inversion of only eval rule

\[ ;; \Sigma; (\Delta_1, \Delta_2); E; \vdash \text{unpack}_E k \cdot o@s \text{ in } e_2 : E \setminus \emptyset|{-} \]
\( k = \text{unique} \| \text{full} \| \text{immutable} \ (o : C) \in \Sigma ;; \Sigma; \Delta_1 \vdash_E k \cdot o@s \)
\[ E = \text{emp}|\text{ot} ;; \Sigma; \Delta_2, [o/\text{this}]\text{inv}_C(s, k); E; k \cdot o@s \vdash e_2 : E \setminus \omega_2|{-} \]
Inversion of only typing rule

Let \( \Sigma' = \Sigma, \Delta^{E'} = \Delta_2, [o/\text{this}]\text{inv}_C(s, k), u'^e = k \cdot o@s, \omega' = \omega_2 \).
Substitution

\[ ;; \Sigma'; \Delta'; E; u' \vdash e' : E \setminus \omega'|{-} \]
Must show \( \Sigma; \Delta; \overline{\Delta'}; u \vdash H; \overline{S_p} \)
\[ \Sigma'; (\Delta^{E'}, \Delta^*); \vdash S'_p \]
We have removed \( k \cdot o \) from \( \Delta \) and \( S_p \), and added field perms to \( \Delta \) which are ignored.
\[ \Sigma'; \overline{\Delta'} \vdash \overline{S'_p} \]
No other \( \Delta \) or \( S_p \) changed.

Must show \( \Sigma'; \overline{\Delta^{E'}}; S'_p; u^{E'} \vdash H' \)
1.) ok No change to \( \text{dom}(\Sigma) \) or \( \text{dom}(H) \)
2.) ok We have not changed the number of \( \text{wt} \) elements from before.
If $\mathcal{E} \neq \text{wt}$, then not-wt($\Delta'$) because invariants cannot contain pure and shared information.

3.) ok \hspace{1cm} k = \text{immutable|full|unique}. \ o \mapsto C(\ldots)@\text{unpacked}(k) \in H' \text{ and writes}(k).

4.a.) ok \hspace{1cm} \text{No change}

4.b.) ok \hspace{1cm} \text{Before step, either } S = \text{unpacked}(s) \text{ and invariant holds by this rule, or } S = s \text{ and invariant held by this rule. Fields have not changed.}

4.c.) ok \hspace{1cm} \text{No new stack perms in } \Delta'.

4.d.) ok \hspace{1cm} 4.b. \text{ was true before step. Fields added to } \Delta' \text{ are given by } \text{inv}_C(s, k).

4.e.) ok \hspace{1cm} 4.g. \text{ was true before step. Any unique or full fields cannot be in other } \Delta s \text{ and } u.

4.f.) ok \hspace{1cm} 4.g. \text{ was true before step. Other permissions to fields must agree with state.}

4.g.) ok \hspace{1cm} \text{No fields altered.}

$\omega' - \omega = \omega'$ only contains fields of $o$.

Packing Flag lemma

$\Delta'$ does not contain any fields in $\omega - \omega'$

\[ \forall o \in \text{dom}(H) : w(o, S_p) - w(o, \Delta) - w(o, u) \leq w(o, S'_p) - w(o, \Delta') - w(o, u') \]

Net is unchanged. Permission moved from $\Delta$ to $u'$.

\[ T = . \]

not-active(unpack) implies not-active($e_2$)

active(unpack) cannot be derived.

\[ a = a' \text{ and not-active}(e) \text{ implies not-active}(e') \]

Above

CASE E-UNPACK-R-Wt

So $e = \text{unpack}_\text{wt} \ k \cdot o@s \text{ in } e_2, a' = e_2, a = a', H' = H[o \mapsto C(\ldots)@\text{unpacked}(s)], S'_p = S_p[(k' - k) \cdot o]$ and $T = .$

\[ \vdash \Sigma; \Delta; \mathcal{E}; u \vdash e : E \setminus \omega|u'' \]

Assumption

\[ \Sigma; \Delta; \mathcal{E}; u \vdash e : E \setminus \omega|u'' \]

Assumption

$\Sigma; \Delta; \mathcal{E}; u \vdash e : E \setminus \omega|u''$

Inversion of only evaluation rule.

$\Sigma; \Delta; \mathcal{E}; u \vdash e : E \setminus \omega|u''$

Let $\Sigma'' = \Sigma,$ $\Delta'' = \Delta,$ $[o/this]\text{inv}_C(s, k), u'' = k \cdot o@s, \ \omega' = \omega_2.$

Substitution

$\Sigma; \Delta''; \mathcal{E}; u'' \vdash e : E \setminus \omega_2|-$

Only typing rule and its inversion.

We have removed $k \cdot o$ from $\Delta$ and $S_p$, and added field perms to $\Delta$ which are ignored.

$\Sigma''; \Delta''; \mathcal{E}; u'' \vdash e : E \setminus \omega_2|-$

No other $\Delta$ or $S_p$ changed.

Must show $\Sigma; \Delta''; \mathcal{E}; u'' \vdash H'$

No change to $\text{dom}(\Sigma)$ or $\text{dom}(H)$

1.) ok
2.) ok  
Still the only wt, no need to prove not-active.

3.) ok  
\( E = wt. \)

4.a.) ok  
No change

4.b.) ok  
Before step, either \( S = \text{unpacked}(s) \) and invariant holds by this rule, or \( S = s \) and invariant held by this rule.

Fields have not changed.

4.c.) ok  
No new stack perms in \( \Delta' \).

4.d.) ok  
4.b. was true before step. Fields added to \( \Delta' \) are given by \( \text{inv}_C(s, k) \).

4.e.) ok  
4.g. was true before step. Any unique or full fields cannot be in other \( \Delta \)s and \( u \).

4.f.) ok  
4.g. was true before step. Other permissions to fields must agree with state.

4.g.) ok  
No fields altered.

o was unpacked.

\( \omega' - \omega = \omega' \) only contains fields of \( o \).  
\( \omega' - \omega = \omega' = \emptyset \)

\( \Delta' \) does not contain any fields in \( \omega - \omega' \)

\( \forall o \in \text{dom}(H) : w(o, S_p) - w(o, \Delta) - w(o, u) \leq w(o, S'_p) - w(o, \Delta') - w(o, u') \)

Net is unchanged. Permission moved from \( \Delta \) to \( u' \).

\( T = . \)

not-active(unpack) implies not-active(e_2)

Inversion of not-active.

active(unpack) cannot be derived.

\( a = a' \) and not-active(e) implies not-active(e')

Above

**CASE E-PACK-R**

So \( e = \text{pack} \ o \ to \ s \ in \ e_2, e' = e_2, a = a', H' = H[o \mapsto C(\ldots)s], S'_p = S_p[(k + k_o) \cdot o] \) and \( T = . \)

\[ \vdash \Sigma; \Delta; E; u \vdash e : E \setminus \omega|u'' \]  
Assumption

\[ \Sigma; \Delta; E; u \vdash e : E \setminus \omega|u'' \]

\( \Sigma; \Delta; E; u \vdash e : E \setminus \omega' \)

Assumption

\( \text{inv}_C(s) \) satisfied by \( o \)'s fields

\[ k_o \cdot o \in S_p \ o \mapsto C(f = k \cdot o) @ \text{unpacked}(s) \in H \]

Inversion of only evaluation rule

\[ o \in \Sigma \quad \Delta^E = (\Delta_1, \Delta_2) \]
\[ \Sigma; \Delta_1 \vdash \text{[o/this]} \text{inv}_C(s, k) \]
\[ \Sigma; k \cdot o@s \vdash k \cdot o@S \]
\[ \vdash \Sigma; \Delta_2, k \cdot o@S; E; e_2 : E \setminus \{f\} \]

No temporary permissions for \( o.f \) in \( \Delta_2 \)

Inversion of only typing rule

Let \( \Sigma' = \Sigma, \quad D_{\Delta'} = \Delta_2, k \cdot o@S \quad u'^E = - \omega' = \emptyset \)

\[ \vdash \Delta'; E; u' \vdash e' : E \setminus \omega'| \]

Substitution

Must show \( \Sigma; \Delta^E; u'^E \vdash H; S'_p \)
\( \Sigma'; (\Delta_{E'}^*, \Delta^*) \vdash S'_p \)
\( k \) added back to \( \Delta, S_p \Sigma'; \Delta^* \vdash S'_p \)

1.) ok

No other \( \Delta \) or \( S_p \) changed.

No change to \( \text{dom}(\Sigma) \) or \( \text{dom}(H) \)
2.) ok
We have not added a wt that was not previously there.
If \( \mathcal{E} \neq \text{wt} \), not-wt(\( \Delta^e \)) by inversion of \( \Sigma; \Delta \vdash \Delta + o@\$. 

3.) ok
We have only removed a permission from \( u^e \). This \( o \) is packed and inv\( _C(o, \text{unique}) \) from above.

4.a.) ok
No change

4.b.) ok
For only modified \( o, S = s \) and invariant satisfied from assumption and 4.d being true before step.

4.c.) ok
Only one new permission added to \( \Delta \), and \( S = s \).

4.d.) ok
We have only removed fields from \( \Delta \).

4.e.) ok
True before step. Can be no other full or uniques to \( u, \) now in \( \Delta ' \).

4.f.) ok
True before step. \( u, \) now in \( \Delta \), must be consistent.

4.g.) ok
From 4.e. and 4.f before step.

\( \omega' = \emptyset \subset \omega \)
\( \forall o \in \text{dom}(H) : w(o, S_p) - w(o, \Delta) - w(o, u) \leq w(o, S'_p) - w(o, \Delta') - w(o, u') \)

Net is unchanged. Permission moved from \( u \) to \( \Delta' \).

\[ T = . \]
not-active\( (\text{pack}) \) implies not-active\( (e_2) \) Inversion of not-active.
active\( (\text{pack}) \) cannot be derived.
a = a' and not-active\( (e) \) implies not-active\( (e') \) Above

CASE E-PACK-RW
So \( e = \text{pack} o \to s \) in \( e_2, e' = e_2, a = a', H' = H[o \mapsto C(\ldots)@s], S'_p = S_p[(k + k_0) \cdot o] \) and \( T = . \)

\[ \vdash ; \Sigma; \Delta; \mathcal{E}; u \vdash e : E \setminus \omega|u'' \]
Assumption

\[ \Sigma; \Delta; \mathcal{E}; u^e \vdash H; S_p \]
Assumption

\( o \mapsto C(\ldots)@\text{unpacked}(k) \in H \) \( k_0 \cdot o \in S_p \) inv\( _C(s) \) satisfied by \( o \)’s fields

Inversion of only eval rule

\( o \in \Sigma; \Delta = (\Delta_1, \Delta_2) \)
\( \Sigma; \Delta_1 \vdash o/\text{this}|\text{inv}_C(s, k) \)
\( \Sigma; k \cdot o@S \vdash k \cdot o@\$ \)
\[ \vdash ; \Sigma; \Delta_2, k \cdot o@S; \mathcal{E}; u \vdash e_2 : E \setminus \{l.f\}\]
No temporary permissions for \( o.f \) in \( \Delta_2 \)

Inversion of only typing rule

Let \( \Sigma' = \Sigma \), \( D^e = \Delta_2, k \cdot o@\$ \) \( u^e' = = \omega' = \emptyset \)

\[ \vdash ; \Delta; \mathcal{E}; u' \vdash e' : E \setminus \omega'\]
Substitution

Must show \( \Sigma; \Delta; \mathcal{E}; u^e' \vdash H; S_p \)
\( \Sigma'; (\Delta^e, \Delta^{'}) \vdash S_p' \)
\( k \) added back to \( \Delta, S_p, \Sigma', \Delta = S_p' \)

No other \( \Delta \) or \( S_p \) changed.

1.) ok
No change to \( \text{dom}(\Sigma) \) or \( \text{dom}(H) \)

2.) ok
We have not added a wt that was not previously there.

22
If $E \not= \text{wt}$, \text{not-wt}($\Delta'^E$) by inversion of $\Sigma; \Delta' \vdash E \cdot k \cdot o@\$. 

3.) ok
We have only removed a permission from $u^E$. This $o$ is packed and inv$_C$($o$, unique) from above.
4.a.) ok
No change
4.b.) ok
For only modified $o$, $S = s$ and invariant satisfied from assumption and 4.d being true before step.
4.c.) ok
Only one new permission added to $\Delta$, and $S = s$.
4.d.) ok
We have only removed fields from $\Delta$.
4.e.) ok
True before step. Can be no other full or uniques to $u$, now in $\Delta'$.
4.f.) ok
True before step. $u$, now in $\Delta$, must be consistent.
4.g.) ok
From 4.e. and 4.f before step.

$\omega' = \emptyset \subset \omega$
$\forall o \in \text{dom}(H): w(o, S_p) - w(o, \Delta) - w(o, u) \leq w(o, S'_p) - w(o, \Delta') - w(o, u')$
Net is unchanged. Permission moved from $u$ to $\Delta'$.

$T =$ .
\text{not-active(pack)} implies \text{not-active($e_2$)} \hspace{2cm} \text{Inversion of not-active.}
\text{active(pack)} cannot be derived.
\text{$a = a'$ and not-active($e$) implies not-active($e'$)} \hspace{2cm} \text{Above}

\text{CASE E-ASSIGN}

So $e = o_1.f := k \cdot o_2$, $e' = k' \cdot o'$, $a = a'$, $H' = H[o_1 \mapsto C(\ldots, f = k \cdot o_2, \ldots) \cdot \text{unpacked}(k')]$, $S'_p = S_p[(k_2 - k) \cdot o_2], k' \cdot o'$ and $T = \cdot$.

$; \Sigma; \Delta; E; u \vdash E \cdot o$$u''$
$; \Sigma; \Delta^E; u^E \vdash H; \overline{S_p}$
$e_1 \cdot o_1 \in S_p$
$k_1 \cdot o_1 \in S_p$

\text{Assumption}

$; \Sigma; \Delta; E; u \vdash \cdot$

\text{Assumption}

\text{Assumption}

$k_2 \cdot o_2 \in S_p$

$\emptyset \subset \omega$
$k_2 \cdot o_2 \mapsto C(\ldots)@S_2 \in H$

$; \Sigma; \Delta_1 \vdash E \cdot k \cdot o \cdot \exists x : C_i, P$

\text{Inversion of only eval rule}

$; \Sigma; \Delta_1 \vdash E \cdot o$

\text{Inversion of only typing rule}

$k'$ \cdot $o'$ went into both $\Delta'$, as subst. for field permission and $S'_p$.
Field permissions inserted, which are ignored.

$\Sigma' \vdash \overline{S'_p}$

No other $\Delta$ or $S_p$ changed.
Must show $\Sigma'; \Delta'; S'_p; u' \vdash H'$

1.) ok  
No change to $\text{dom}(\Sigma)$ or $\text{dom}(H)$

2.) ok  
We have not changed the number of $\text{wt}$ elements from before.

The permissions added to $\Delta'$ were cleansed, by the inverse of transaction-aware linear judgment.

3.) ok  
$u'^e$ unchanged.

4.a.) ok  
No change

4.b.) ok  
$S = \text{unpacked}(k)$

4.c.) ok  
From 4.d. true before step.

4.d.) ok  
From 4.c. true before step.

4.e.) ok  
From 4.g. true before step.

4.f.) ok  
From 4.g. true before step.

4.g.) ok  
From 4.d,f. true before step.

$\omega' = \{o_1, f\} \forall o \in \text{dom}(H) \rightarrow w(o, S_p') - w(o, \Delta) - w(o, u) \leq w(o, S_p') - w(o, \Delta') - w(o, u')$

$k \cdot o_2$ and $k' \cdot o'$ move between field and stack.

$T = \cdot$

$a' = a$ and only not-active$(e)$ can be derived.

not-active$(e')$

not-active rules for field.

not-active rules for loc.

CASE E-CALL

So $e = k \cdot o.m(\overline{k \cdot o})$, $e' = [o/this][\overline{o/this}]\cdot e_m$, $H' = H'$, $S_p = S_p$, $a' = a$,

$\vdash \Sigma; \Delta; \mathcal{E}; u \vdash e : E \wedge \omega'u''$  
Assumption

$\Sigma; \Delta'; \mathcal{E}; u'^e \vdash H' : S'_p$  
Assumption

$\text{mbody}(C, m) = \overline{\pi}.e_m \quad \text{mtype}(C, m) = \forall x : C. P \rightarrow E$

$H; S_p[k \cdot o, k \cdot o + [o/this][\overline{o/this}]P$

Inversion only eval rule

$\vdash \Sigma; \Delta; \mathcal{E}; \vdash k \cdot o.m(\overline{k \cdot o}) : \exists x : C. P'_r \wedge \emptyset$  

$\vdash \Sigma; \Delta + e \vdash [o/this][\overline{o/this}]P \quad \text{mtype}(C, m) = \forall x : C. P \rightarrow \exists x : C. P_r$

$\text{forget}_e(P'_r) = P'_r$

Only typing rule and its inversion.

$x : C, this : C, \vdash P; \text{wt}; \vdash e_m : \exists x : C. P'_r \wedge \emptyset$  

$x : C, this : C, \vdash P; \text{ot}; \vdash e_m : \exists x : C. P''_r \wedge \emptyset$  

$\mathcal{E} = \text{wt}$ implies $P_r = P'_r$

$\mathcal{E} \neq \text{wt}$ implies $P''_r = P'_r$

Inversion of $M$ ok

Inversion of forget

$\vdash \Sigma'; \Delta'; \mathcal{E}; u' \vdash [o/this][\overline{o/this}]\cdot e_m : E \wedge \omega'u''$

$\Sigma'; \Delta'; S'_p; u' \vdash H'$  

No changes
∀o ∈ dom(H) : w(o, S_p) − w(o, ∆) − w(o, u) ≤ w(o, S'_p) − w(o, ∆') − w(o, u')

No changes

a' = a and not-active(e)

Only not-active can be derived for call.

not-active(e')

Well formed method body cannot be active.

**CASE E-SPAWN**

So e = spawn (k · o.m(k · o)), H' = H', S'_p = S_p, S_p2 with S_p1, immutable · o_d @ S_d replacing S_p.

Assumption

\[ \vdash \Sigma; \Delta; \mathcal{E}; t \vdash E \setminus \Omega | u'' \]

\[ \vdash \Sigma; \Delta; \mathcal{E}; t \vdash H; S_p \]

\[ \text{mbody}(C, m) = \forall e_m \text{ mtype}(C, m) = \forall x : C.P \rightarrow o \]

\[ H; S_p, [k \cdot o, k \cdot o] \vdash [o/this][\sigma/\tau]P \]

Inversion of only eval rule

Inversion of only typing rule

Let e' = immutable · o_d, T = (⟨[o/this][\sigma/\tau]e_m, S_p2⟩, \Sigma' = \Sigma, \Delta_o = immutable · o_d @ S_d.

Inversion of mtype.

Always true of o_d, which is implicitly in all ∆.

\[ \Delta_o = \Delta, u_o = - , u_o' = - \]

\[ \vdash \Sigma'; \Delta_o; t \vdash E_t \setminus \emptyset | - \]

\[ x : C, this : C; \vdash P; ot; t \vdash e_m : E_t \setminus \emptyset | - \]

\[ \vdash \Sigma'; \Delta_o; t \vdash immutable \cdot o_d ; \exists : C_s \cdot \text{immutable} \cdot o_d @ S_d \]

Only one permission in ∆' and we added it to S'_p.

From above

No other ∆ or S_p changed.

Must show \( \Sigma; \Delta_o; u_o' \vdash H; S_p \)

1.) ok

No change to dom(\( \Sigma \)) or dom(H)

2.) ok New ∆s are tagged with ot. ∆' on has immutable objects and ∆_t is clean, inverse of TALJ.

3.) ok

New us are both -.

4.a.) ok

No states or fields changed.

4.b.) ok

Nothing new in ∆s w.r.t. the heap.

4.c.) ok

Nothing new in ∆s w.r.t. the heap.

4.d.) ok

Nothing new in ∆s w.r.t. the heap or u.

4.e.) ok

Special default object, o_d, is always in state s_d.

4.f.) ok

No fields modified.

\[ \omega' \subseteq \omega \]

\[ \omega_t \subseteq \omega \]

\[ \omega' = \omega = \emptyset \]

\[ \omega_t = \omega = \emptyset \]

\[ \Delta' \text{ contains no permissions for fields in } \emptyset \]

\[ \Delta_t \text{ contains no permissions for fields in } \emptyset \]
\[ \forall o \in \text{dom}(H) : w(o, S_p) - w(o, \Delta) - w(o, u) \leq w(o, S'_p \cup S_{ip}) - w(o, \Delta', \Delta_i) - w(o, u', u_t) \]

Net is unchanged. \text{immutable} \cdot o_d @ s_d added to \( S'_p \) and \( \Delta' \)

\( a' = a \) and \text{not-active}(e)

\text{not-active}(e') 

\text{not-active}(e_t) 

\text{Property of well-typed method body.}

\text{Case E-Read-R}

So \( e = k \cdot o_f, e' = k \cdot o', T = \cdot, a' = a, H' = H[o \mapsto C(\ldots, f_i = (k' - k) \cdot a, \ldots) @ \text{unpacked}(s'\hat{n})], S'_p = S_p, (k \cdot o') \).

\[ \vdash \Sigma; \Delta; \epsilon; u \vdash e : E \setminus \omega | u' \]

\[ \Sigma; \Delta; u \vdash H; S_p \]

Assumption

\( k = \text{pure} | \text{immutable} \o o \mapsto C(\ldots, f_i = k' \cdot o') \text{unpacked}(s'\hat{n}) \in H \)

Inversion of only eval rule

\[ \vdash \Sigma; \Delta; \epsilon; k_u \cdot o @ S_u \vdash k \cdot o.f : \exists x : T_i.x / f_i \text{P} \setminus \emptyset | k_u \cdot o @ S_u \]

\( \text{readonly}(k_u) \implies \text{readonly}(k) \)

\[ \vdash \Sigma; \Delta \vdash P \]

\[ \text{localFields}(C) = \overline{f : \overline{C}} \]

Only typing rule and its inversion

Let \( \Sigma' = \Sigma, u^{\epsilon'} = u^\epsilon, \Delta^{\epsilon'} = [o' / o.f] \text{P}, \omega' = \omega = \emptyset \).

\[ \vdash \Sigma'; \Delta' ; \epsilon'; u' \vdash k \cdot o' : E \setminus \omega' | u' \]

\text{Rule T-LOC.}

\text{Must show} \( \Sigma; \overline{\Delta^{\epsilon'}} ; \overline{u^{\epsilon'}} \vdash H; S'_p \)

\[ \Sigma' ; (\Delta^{\epsilon'}, \Delta^\ast) \vdash S'_p \]

\( \Delta' \) only has permissions for \( o' \), this object was added to \( S_p \).

No other \( \Delta \) or \( S_p \) changed.

1.) ok

2.) ok

We have not added a wt that was not previously there.

If \( \mathcal{E} \neq \text{wt} \), \text{not-wt}(\Delta^{\epsilon'}) \text{by inversion of} \( \Sigma; \Delta' \vdash k \cdot o @ S_u \).

\( u' \) has not changed.

3.) ok

4.a.) ok

No change

4.b.) ok

By inversion of \( - \) on permissions and \( \text{inv}_C(s, \text{immutable}) \)

4.c.) ok

States are correct by \( \text{inv}_C(s, \text{immutable}) \) of \( o' \)'s fields.

4.d.) ok

We have only removed field permissions from \( \Delta \).

4.e.) ok

There can be no full or unique perm in \( P \) after downgrading.

4.f.) ok

From 4.g. true before step.

4.g.) ok

True by inversion of subtraction on permissions.

\( \omega' \subseteq \omega \)

\( \omega' = \omega = \emptyset \)

\( \Delta' \) contains no permissions for fields in \( \emptyset \)

\( \forall o \in \text{dom}(H) : w(o, S_p) - w(o, \Delta) - w(o, u) \leq w(o, S'_p \cup S_{ip}) - w(o, \Delta', \Delta_i) - w(o, u') \)

Net unchanged. \( k \cdot o' \) added to \( S_p \) and \( \Delta \).

\( T = \cdot \)

\( a' = a \) and only \text{not-active}(e) can be derived.

\text{not-active} rule for field reads.

Only \text{not-active}(e') can be derived.

\text{not-active} rule for location reads.
CASE E-READ-RW

So \( e = k \cdot o.f_i, e' = k \cdot o' \), \( T = \cdot \), \( a' = a \), \( H' = H[o \mapsto C(\ldots, f_i = (k' - k) \cdot o, \ldots)] \@\text{unpacked}(k'') \), \( S' = S_p = S_p, (k \cdot o'). \)

\[
\vdash; \Sigma; \Delta; \mathcal{E}; u \vdash e : E \setminus \omega \mid u'' \\
\Sigma; \Delta' \vdash \mathcal{E} \cdot u \vdash H; S_p \\
k \leq k' \quad o \mapsto C(\ldots, f_i = k' \cdot o') \@\text{unpacked}(k'') \in H
\]

Inversion of only eval rule.

\[
\vdash; \Sigma; \Delta \vdash \mathcal{E} : \Pi \quad \text{localFields}(\mathcal{E}) = \bar{f} : C \\
\vdash; \Sigma; \Delta; \mathcal{E}; k_u \cdot o \@ S_u \vdash k \cdot o.f_i : \exists x : T_i[x/f_i] P \setminus \emptyset \mid k_u \cdot o \@ S_u
\]

Inversion of only typing rule.

Let \( \Sigma' = \Sigma, u^{e'} = u^e, \Delta^{e'} = [o'/o.f_i]P, \omega' = \omega = \emptyset \).

\[
\vdash; \Sigma'; \Delta'; \mathcal{E}'; u \vdash k \cdot o' : E \setminus \omega' \mid u'
\]

Rule T-LOC.

Must show \( \Sigma; \Delta' \vdash \mathcal{E}' \cdot u^{e'} \vdash H; S_p \)

\( \Sigma'; (\Delta^{e'}, \Delta^*) \vdash S_p \)

\( \Sigma'; \Delta' \vdash S_p \)

Must show \( \Sigma' ; \Delta^{e'} ; S_p' ; u^{e'} \vdash H' \)

1.) \( \text{ok} \)  
No change to \( \text{dom}(\Sigma) \) or \( \text{dom}(H) \)

2.) \( \text{ok} \)  
We have not added a \( \text{wt} \) that was not previously there.

If \( \mathcal{E} \neq \text{wt} \), not-\( \text{wt}(\Delta^{e'}) \) by inversion of \( \Sigma; \Delta' \vdash k \cdot o \@ S \).

\( u' \) has not changed.

3.) \( \text{ok} \)  
No change

4.a.) \( \text{ok} \)  
4.d.) was true before step.

4.b.) \( \text{ok} \)  
4.e.) \( \text{ok} \)  
4.g.) was true before step.

4.c.) \( \text{ok} \)  
4.f.) \( \text{ok} \)  
4.g.) \( \text{ok} \)  
True by inversion of subtraction on permissions.

\( \omega' \subseteq \omega \)

\( \Delta' \) contains no permissions for fields in \( \emptyset \)

\( \forall o \in \text{dom}(H) : w(o, S_p) - w(o, \Delta) - w(o, u) \leq w(o, S'_p) - w(o, \Delta') - w(o, u') \)

Net unchanged. \( k \cdot o' \) added to \( S_p \) and \( \Delta \).

\( T = \cdot \)

\( o' = a \) and only not-active(e) can be derived.

Not-active rule for field reads.

Only not-active(e') can be derived.

Not-active rule for location reads.

CASE E-INATOMIC

So \( e = \text{inatomic}(e_1), e' = \text{inatomic}(e'_1), o' = a \), \( H' = H' \) from I.H., \( S'_p = S'_p \) from I.H., \( \omega' = \omega' \) from I.H.
\[\vdash \Sigma; \Delta; \varepsilon; u \vdash e : E \setminus \omega|u''\]  
\[\Delta u \vdash H; S_p\]  
Assumption

\[a ; H; \langle e_1, S_p \rangle \rightarrow a'; H'; \langle e'_1, S'_p \rangle ; T\]  
Inversion of only eval rule

\[\vdash \Sigma; \Delta; \varepsilon; u \vdash \text{inatomic}(e_1) : \exists x : C. P' \setminus \omega|u'\]  
\[\vdash \Sigma; \Delta; \wt; u \vdash e_1 : \exists x : C. P \setminus \omega|u' \quad \text{forget}_\varepsilon(P) = P'\]  
Inversion of only typing rule

Apply induction hypothesis.
\[\Sigma; \Delta u \vdash H; S_p\]  
I.H.

\[T \text{ ok}\]  
I.H.

\[\omega' \text{ ok}\]  
I.H.

\[a' = a \text{ and active(\text{inatomic}(e_1))}\]  
active rule for \text{inatomic}.

\[\text{active}(e')\]  
active rule for \text{inatomic}.

**Case E-Atomic-Exit**

So \(e = \text{inatomic}(k \cdot o)\), \(e' = k \cdot o\), \(S_p' = S_p\), \(H' = H\), \(a' = a\).

\[\vdash \Sigma; \Delta; \varepsilon; u \vdash e : E \setminus \omega|u''\]  
Assumption

\[\Delta u \vdash H; S_p\]  
Assumption

\[\vdash \Sigma; \Delta; \varepsilon; u \vdash \text{inatomic}(e) : \exists x : C. P' \setminus \omega|u''\]  
\[\vdash \Sigma; \Delta; \wt; u \vdash e : \exists x : C. P \setminus \omega|u'' \quad \text{forget}_\varepsilon(P) = P'\]  
Only typing rule and its inverse. Let \(\Sigma' = \Sigma, \omega' = \omega\)

Case: \(\varepsilon = \wt\)

Let \(\Delta u = \Delta, u\).
\[\vdash \Sigma' ; \Delta' ; \wt; u \vdash k \cdot o \vdash \exists X : C. P \setminus \omega|u''\]  
By substitution, and \(P' = P\) when \(\varepsilon = \wt\)

\[\langle \Delta u, u \rangle \text{ ok}\]  
Above

Case: \(\varepsilon \neq \wt\)

Let \(\Delta u = P', u \rangle = u\).
\[\vdash \Sigma' ; \Delta' ; \varepsilon; u \vdash k \cdot o \vdash \exists X : C. P' \setminus \omega|u''\]  
Rules T-LOC

\(\Delta u\) contains no share or pure perms.
\(u\) contains no share or pure permissions.

\[\langle \Delta u, u \rangle \text{ ok}\]  
Above

Heap cond 3 satisfied.

\[\Sigma; \Delta u \vdash H; S_p\]  
Rest of heap unchanged. \(T = a' = o \neq \bullet = a\), \text{active}(e)

\[\text{not-active}(e')\]  
Active rule for \text{inatomic}

Only derivable rule for \(k \cdot o\)
CASE E-ATOMIC

So $e = \text{atomic}(e_1), e' = \text{inatomic}(e_1), H' = H, S_p' = S_p, a' = \bullet, \omega' = \omega$.

$\vdash \Sigma; \Delta; \mathcal{E}; u \vdash e : E \setminus \omega | u''$

Assumption

$\Sigma; \overline{\Delta'}; u \in H; \overline{S_p}$

Assumption

$\vdash \Sigma; \Delta; \mathcal{E}; u \vdash \text{atomic}(e_1) : \exists x : C.P' \setminus \omega | u''$

$\vdash \Sigma; \Delta; \mathcal{E}; u \vdash \exists x : C.P \setminus \omega | u''$

only typing rule and its inversion. Let $\Sigma' = \Sigma, u' = u, \Delta' = \Delta, \omega' = \omega$.

By rule T-INATOMIC. Let $u'$ and $\Delta'$ be tagged with $\mathcal{W}$.

$a' = o \neq \bullet = a$. Given not-active($e$).

active($e'$)

active rule for inatomic.

CASE E-NEW

So $e = \text{new} C(k \cdot o), e' = \text{unique} \cdot o_n, H' = H, o_n \mapsto C(f = k \cdot o)@s, S_p' = (S_p - k \cdot o), \text{unique} \cdot o_n, a' = a$.

$\vdash \Sigma; \Delta; \mathcal{E}; u \vdash e : E \setminus \omega | u''$

Assumption

$\Sigma; \overline{\Delta'}; u \in H; \overline{S_p}$

Assumption

$H; S_p \vdash \overline{[\sigma/f]}P \quad \text{init}(C) = \langle \exists f : C.P, s \rangle$

Inversion of only evaluation rule

$\vdash \Sigma; \Delta; \mathcal{E}; u \vdash \text{new} C(k \cdot o) : \exists x : C.\text{unique} \cdot x@s \setminus \emptyset | u$

$\vdash \Sigma; \Delta \vdash \overline{[\sigma/f]}P \quad o : C \subseteq \Sigma \quad \text{init}(C) = \langle \exists f : C.P, s \rangle$

Only typing rule and its inversion.

Let $\Sigma' = \Sigma, o_n : C, \omega'' = u, \Delta'' = \text{unique} \cdot o_n@s$, where $\mathcal{E}$ tag is the same as before step, $\omega' = \omega = \emptyset$.

$\vdash \Sigma'; \Delta'; \mathcal{E}; u \vdash \text{unique} \cdot o_n : \exists x : C.\text{unique} \cdot x@s$

By rule T-LOC

Must show $\Sigma; \overline{\Delta'}; u \in H; \overline{S_p}$

$\Sigma'; (\overline{\Delta''}, \Delta^*) \vdash S_p'$

We removed $k \cdot o$ from both $S_p$ and $\Delta$ and added $\text{unique} \cdot o_n@s$ to both.

$\Sigma'; \overline{\Delta''} \vdash \overline{S_p'}$

No other $\Delta$ or $S_p$ changed.

Must show $\Sigma'; \overline{\Delta''}; S_p'; u \in H'$

1.) ok

2.) ok We have not changed $\mathcal{E}$ tagging. Only new permission is unique, so invariant holds, if nec.

3.) ok $o_n$ is packed. $\mathcal{I}_C$ holds b/c inverse of init and runtime proof of $P$.

29
4.a.) ok
4.b.) ok
4.c.) ok
4.d.) ok
4.e.) ok
4.f.) ok
4.g.) ok

Fields were all in \( \Delta \) before step, therefore by 4.e and 4.f property now holds for fields.
\[ \forall o \in \text{dom}(H) : w(o, S_p) - w(o, \Delta) - w(o, u) \leq w(o, S_p') - w(o, \Delta') - w(o, u') \]

\( k \cdot o \) removed from \( S_p' \) and \( \Delta' \).

\[ T = \ . \]
\[ a' = a \] and only not-active\((e)\) can be derived.

not-active\((e')\) inv on not-active rule.

not-active rule for locations.

**CASE E-LET-E**

So \( e = \text{let } x = e_1 \text{ in } e_2, e' = \text{let } x = e'_1 \text{ in } e_2. \)

\[
\cdot; \Sigma; \Delta; \epsilon; u \vdash e : E \setminus \omega|u'' \\
\Sigma; \overline{\Delta}^e; u^e \vdash H; \overline{S}_p^e \\
a; H; \langle e_1, S_p \rangle \rightarrow a'; H'; \langle e'_1, S'_p \rangle; T \\
\Delta^e = (\Delta_1, \Delta_2) \cdot; \Sigma; \Delta_1; \epsilon; u \vdash e_1 : \exists x : \Sigma. P \setminus \omega_1|u_2 \\
\Sigma; \Delta_2, P = \epsilon P' \\
x : C; \Sigma; P'; \epsilon; u_2 \vdash e_2 : E \setminus \omega_2|u''
\]

Inversion of only eval rule

Inversion of only typing rule

\[
\Sigma; \overline{\Delta}^e; u^e \vdash H; \overline{S}_p^e \text{ where } \overline{\Delta}^e \text{ has } \Delta_1 \text{ instead of } \Delta.
\]

Compositionality

Apply induction hypothesis where \( (\Delta_2, \Delta^*) \) is the additional linear context.

\[
\Sigma'; \overline{\Delta}^e; u' \vdash H'; \overline{S}_p^e \\
\overline{\Delta} \text{ is the same as } \overline{\Delta} \text{ except } \Delta \text{ is now } \Delta_1, \Delta_2, \Delta^*. 
\]

I.H.

Gives us \( \Sigma' \supseteq \Sigma. u^e' \) and \( \omega'_1 \)

I.H.

Either (a) \( u = - \) and \( u' = k \cdot o \) or (b) \( \omega'_1 \supseteq \omega_1 \).

I.H.

\[
\cdot; \Sigma'; \Delta_1; \epsilon; u' \vdash e'_1 : \exists x : \Sigma. P \setminus \omega'_1|u_2
\]

I.H.

\[ \forall o \in \text{dom}(H) : w(o, S_p) - w(o, \Delta) - w(o, u) \leq w(o, S_p') - w(o, \Delta') - w(o, u') \]

I.H. Fractions in \( \Delta_2 \) unchanged.

\[ T \text{ ok} \]

I.H.

**SUBCASE:** \( u = - \) and \( u' = k \cdot o \) and \( \omega'_1 - \omega_1 \) only contains fields of \( o \).

\( \Delta_2, \Delta^* \) do not contain permissions for fields of \( o \).

Definition of well-typed store.

\[ \omega'_1 - \omega_1 \text{ contains only fields of } o \]
\[ \vdash \Sigma'; \Delta'\cdot \mathcal{E}; u' \vdash e' : E \setminus \omega'|u'' \]

**SUBCASE:** \( \omega'_1 \supseteq \omega_1 \)

\( \Delta_2, \Delta^* \) do not contain permissions for fields in \( \omega'_1 \)

\[ \vdash \Sigma'; \Delta'\cdot \mathcal{E}; u' \vdash e' : E \setminus \omega'|u'' \]

By rule T-LET

If \( a = a' \) and \( \text{active}(e) \), then \( \text{active}(e_1), \text{not-active}(e_2) \)

active\( (e_1) \) implies active\( (e'_1) \)

active\( (e'_1) \) and not-active\( (e_2) \) imply active\( (e') \)

If \( a = a' \) and not-active\( (e) \) then not-active\( (e_1) \) and not-active\( (e_2) \)

\( a = a' \) and not-active\( (e_1) \) implies not-active\( (e'_1) \)

not-active\( (e'_1) \) and not-active\( (e_2) \) imply not-active\( (e') \)

If \( a = o \) and \( a' = e \), then not-active\( (e_1) \) and active\( (e'_1) \)

\( a = o \) implies not-active\( (e) \)

not-active\( (e) \) implies not-active\( (e_2) \)

active\( (e'_1) \) and not-active\( (e_2) \) imply active\( (e') \)

If \( a = e \) and \( a' = o \) then active\( (e_1) \) and not-active\( (e_1) \)

\( a = e \) implies either active\( (e) \) or not-active\( (e) \)

Given active\( (e_1) \), not-active\( (e) \) impossible

active\( (e) \)

active\( (e) \) implies not-active\( (e_2) \)

not-active\( (e'_1) \) and not-active\( (e_2) \) implies not-active\( (e') \)

**CASE** E-LET-V

So \( e = \mathsf{let} \ x = k \cdot \mathsf{o} \ \mathsf{in} \ e_2, e' = [o/x]e_2, H' = H, S'_p = S_p, a' = a. \)

\[ \vdash \Sigma; \Delta; \mathcal{E}; u \vdash e : E \setminus \omega|u'' \]

Assumption

\[ \Sigma; \Delta^*; u^2 \vdash H; S_p \]

Assumption

\( k' \cdot \mathsf{o} \in S_p \) \quad \( o \mapsto C(\ldots) \cap S \in H \) \quad \( k \leq k' \)

Inversion of only eval rule.

Inversion of only eval rule.

Only typing rule and its inversion.

Let \( \Sigma' = \Sigma, \Delta^\mathcal{E} = P', u^\mathcal{E} = u, \omega' = \omega. \) \( \vdash \Sigma'; \Delta'; \mathcal{E}; u' \vdash e_2 : E \setminus \omega_1|u_2 \)

Substitution

Must show \( \Sigma; \Delta^\mathcal{E}; u^\mathcal{E} \vdash H; S_p \)

No change at all except forgetting permissions in \( \Delta \).

\( \forall o \in \text{dom}(H) : w(o, S_p) - w(o, \Delta) - w(o, u) \leq w(o, S'_p) - w(o, \Delta') - w(o, u') \)

No changes

\( T = \)
\[ a' = a. \]
\[ \text{not-active}(e) \]
\[ \text{not-active}(e_2) \]

No active rule for locations, let rule.

1.6 Progress

1.6.1 Top-Level Progress

If \( \vdash a; H; T \)

Then there exists either:

- \( \tau \) such that \( T = \langle \tau, \overline{S_p} \rangle \), or
- \( a'; H'; T' \) such that \( a; H; T \rightarrow a'; H'; T' \)

Proof: By structural induction on the derivation of \( \vdash a; H; T \)

**CASE T-TOP-LEVEL**

\( \vdash a; H; T \)

**correct-atomic**(*a, T*)

Subcase: \( a = \circ \)

Every \( e_i \) in \( \langle \text{o} e, S_p \rangle \) is not-active(*e_i*).

Subcase: Every \( e \) in \( \overline{\tau} \) is a value

Proof satisfied

Subcase: \( \exists e_i \) in \( \overline{\tau} \) s.t. \( e_i \) not a value

\( e_i \) must take a step

Global thread pool steps

Subcase: \( a = \bullet \)

There is a \( e_i \) in \( \overline{\tau} \) such that active(*e_i*).

\( e_i \) must take a step

Global thread pool steps

1.6.2 Thread-Level Progress

If

- \( \Sigma; \overline{\Delta}; \overline{w} \vdash H; \overline{S_p} \)

Then the following three items must hold true:

1. If \( \cdot; \Sigma; \Delta; \mathcal{E}; u \vdash e : E \setminus \omega | u \) and active(*e*), then \( \exists e', a', H', T, S_p' \) such that \( \bullet; H; \langle e, S_p \rangle \rightarrow a'; H'; \langle e', S_p' \rangle; T \), where \( \Delta \) and \( S_p \) come from \( \overline{\Delta} \) and \( \overline{S_p} \) respectively and are associated.
2. If $\Delta; \Sigma; u \vdash e : E \setminus \omega | u$, and not-active$(e)$, then $e$ is a value, or $\exists e', a', H', T, S'_{p}$ such that $o; H; (e, S_{p}) \rightarrow a'; H'; (e', S'_{p}); T$, where $\Delta$ and $S_{p}$ come from $\overline{\Delta}$ and $\overline{S_{p}}$ respectively and are associated.

Proof: By structural induction on the derivation of $\Gamma; \Sigma; \Delta; E; u \vdash e : E \setminus \omega | u''$

CASE T-LOC $k \cdot o$ is already a value.

CASE T-CALL

So $e = k \cdot o.m(k \cdot o)$.

$\Sigma; \overline{\Delta}; \overline{E}; \overline{u} \vdash H; \overline{S}_{p}$

Assumption

$\vdash \Sigma; \Delta; \overline{E}; \overline{u} \vdash k \cdot o.m(k \cdot o) : \exists x : C.P_{p} \vdash$

Assumption

$(o : C) \in \Sigma \quad \overline{o : C} \subseteq \Sigma$

$(o : \Delta) \vdash \overline{o/this}[\overline{\sigma/\overline{f}}]P$

Heap condition 1

mtype$(C, m) = \forall x : C.P \rightarrow \exists x : C.P$

Inversion of typing rule. $o, \overline{o} \in \text{dom}(H)$

$o, \overline{o} \in \text{dom}(S_{p})$

$\{k' \cdot o, k \cdot o\} \subseteq S_{p}$

$H, S_{p}|k \cdot o, k \cdot o \vdash [o/this][\overline{\sigma/\overline{f}}]P$

$\Sigma; \Delta \vdash S_{p}$ and heap well-typed

$a; H; (e, S_{p}) \rightarrow a'; H'; (e', S'_{p}); T'$

By rule E-CALL

No rule for active Call.

CASE T-SPAWN

So $e = \text{spawn} (k \cdot o.m(k \cdot o))$.

$\Sigma; \overline{\Delta}; \overline{E}; \overline{u} \vdash H; \overline{S}_{p}$

Assumption

$\vdash \Sigma; \Delta; \overline{E}; \overline{u} \vdash \text{spawn} (k \cdot o.m(k \cdot o)) : \exists x : C_{d}.\text{immutable} \cdot o_{d} \oplus S_{d} \setminus \emptyset \vdash$

Assumption

$mtype(C, m) = \forall x : C.P \rightarrow E$

Inversion of only typing rule.

Heap condition 1

$\Sigma; \Delta \vdash S_{p}$

Above

$\Sigma; \Delta \vdash S_{p}$

$\Sigma; \Delta \vdash S_{p}$ and heap well-typed
\( a; H; (e, S_p) \rightarrow a'; H'; (e', S'_p); T' \)

**By rule E-SPAWN**

No rule for active Spawn.

**CASE T-UNPACK-WT**

So \( e = \text{unpack}_{wt} k \cdot o@s \) in \( e_2 \).

\[
\begin{align*}
\Sigma; \Delta; u^e \vdash H; S_p \\
\vdash \Sigma; (\Delta, \Delta'); wt; \vdash \text{unpack}_{wt} k \cdot o@s \in e' : E \setminus \emptyset \\
(o : C) \in \Sigma; \overline{\sigma : C} \subseteq \Sigma \\
\vdash \Sigma; \Delta \vdash_{e} [o/\text{this}] [\overline{\sigma/\overline{f}}] P \quad \text{mtype}(C, m) = \forall x : C. P \rightarrow \exists x : C.P_r \\
\text{forget}_{e}(P_r) = P'_r \\
k' \cdot o \in S_p \\
o \in \text{dom}(H) \\
k \leq k' \\
\text{SUBCASE: readonly}(k) \\
k = \text{immutable} \implies o \mapsto C(\ldots)@s \in H \text{ or } o \mapsto C(\ldots)@\text{unpacked}(s) \in H \\
\text{From heap condition 4.c and } \leq. \\
k = \text{pure} \implies o \mapsto C(\ldots)@s \in H \\
\text{From heap conditions 4.c, 2 and 3.} \\
\begin{align*}
a; H; (e, S_p) &\rightarrow a'; H'; (e', S'_p); T' \\
\text{By rule E-UNPACK-R-WT} \\
\end{align*}
\]

Only not-active\((e)\) can be derived, and we can step when \( a = \circ \).

**SUBCASE: writes\((k)\)**

\( k = \text{share} \mid \text{full} \mid \text{unique} \implies o \mapsto C(\ldots)@s \in H \)

\[
\begin{align*}
k' &\in S_p \\
k \leq k' \\
a; H; (e, S_p) &\rightarrow a'; H'; (e', S'_p); T' \\
\text{By rule E-UNPACK-RW-WT} \\
\end{align*}
\]

Only not-active\((e)\) can be derived, and we can step when \( a = \circ \).

**CASE T-UNPACK**

So \( e = \text{unpack}_{e} k \cdot o@s \) in \( e_2 \).

\[
\begin{align*}
\Sigma; \Delta; u^e \vdash H; S_p \\
\vdash \Sigma; (\Delta, \Delta'); E; \vdash \text{unpack}_{e} k \cdot o@s \in e' : E \setminus \emptyset \\
k = \text{unique} \mid \text{full} \mid \text{immutable} \quad (o : C) \in \Sigma; \Sigma; \Delta \vdash_{e} k \cdot o@s \\
\mathcal{E} = \text{emp} | \text{ot} \quad \vdash \Sigma; \Delta', [o/\text{this}] \text{inv}_{C}(s, k); E; k \cdot o@s \vdash e' : E \setminus \omega \\
\text{Inversion of only typing rule} \\
\end{align*}
\]

34
\( k' \cdot o \in S_p \)

\( o \in \text{dom}(H) \)

\( k \leq k' \)

**SUBCASE: readonly(\( k \))**

\( k = \text{immutable} \implies o \mapsto C(\ldots)@s \in H \) or \( o \mapsto C(\ldots)@\text{unpacked}(s) \in H \)

From heap condition 4.c and \( \leq \).

\( a; H; \langle e, S_p \rangle \rightarrow a'; H'; \langle e', S'_p \rangle; T' \)

By rule E-UNPACK-R

Only not-active(\( e \)) can be derived, and we can step when \( a = o \).

**SUBCASE: writes(\( k \))**

\( k = \text{full}\mid\text{unique} \implies o \mapsto C(\ldots)@s \in H \)

Heap condition 4.c.

\( \Sigma; \Delta \vdash S_p \)

By rule E-UNPACK-RW

Only not-active(\( e \)) can be derived, and we can step when \( a = o \).

**CASE T-PACK**

So \( e = \text{pack} o \to s' \) in \( e_2 \).

\( \Sigma; \Sigma'; \Delta; \Delta' \vdash H; \bar{S}_p \)

\( \vdash \Sigma'; (\Delta, \Delta') \vdash E; k \cdot o@\bar{s} \vdash \text{pack} o \to s' \) in \( e' : E \setminus \{o\} \)

Assumption

\( \text{forget}_e(k \cdot o@\bar{s}) = k \cdot o@\bar{s} \)

Assumption

\( k = \text{immutable} \implies s = s' \)

Inversion of only typing rule.

\( \text{localFields}(C) = \bar{f} : C \quad (o : C) \in \Sigma \quad ; \Sigma; \Delta \vdash [o/\text{this}]\text{inv}_C(s, k) \)

No temporary permissions for \( o.f \) in \( \Delta' \)

**SUBCASE: writes(\( k \))**

\( o \mapsto C(\ldots)@\text{unpacked}(k) \in H \)

\( o' \)'s fields satisfy \( [o/\text{this}]\text{inv}_C(s, k) \)

\( \Sigma; \Delta \vdash S_p \)

Above and heap condition 4.d.

\( \vdash \Sigma; \Delta \vdash S_p \)

By rule E-PACK-RW

Only not-active(\( e \)) can be derived.

We can step when \( a = o \).

**SUBCASE: readonly(\( k \))**

\( o \mapsto C(\ldots)@\text{unpacked}(s) \in H \)

\( o' \)'s fields satisfy \( [o/\text{this}]\text{inv}_C(s, k) \)

\( \Sigma; \Delta \vdash S_p \)

Above and heap condition 4.d.

\( \vdash \Sigma; \Delta \vdash S_p \)
Only \textbf{not-active}(e) can be derived.

We can step when \( a = \circ \).

\textbf{CASE T-ATOMIC}  
\( e = \text{atomic}(e_1) \)

\( \Sigma; \overrightarrow{\Delta}; \overrightarrow{\epsilon}; \overrightarrow{u} \vdash H; \overrightarrow{S_p} \)  
\( \vdash \Sigma; \Delta; \mathcal{E}; u \vdash \text{atomic}(e_1): \exists x: C.P' \setminus \omega | u' \)  
\( \vdash \Sigma; \Delta; \mathfrak{wt}; u \vdash e_1: \exists x: C.P \setminus \omega | u' \quad \text{forget}_{\mathcal{E}}(P) = P' \)

Inversion of only typing rule

\( a; H; \langle e, S_p \rangle \rightarrow a'; H'; \langle e', S'_p \rangle; T' \)

By rule \textbf{E-ATOMIC-BEGIN}

\textbf{CASE T-INATOMIC}  
\( e = \text{inatomic}(e_1) \)

\( \Sigma; \overrightarrow{\Delta}; \overrightarrow{\epsilon}; \overrightarrow{u} \vdash H; \overrightarrow{S_p} \)  
\( \vdash \Sigma; \Delta; \mathcal{E}; u \vdash \text{inatomic}(e): \exists x: C.P' \setminus \omega | u' \)  
\( \vdash \Sigma; \Delta; \mathfrak{wt}; u \vdash e: \exists x: C.P \setminus \omega | u' \quad \text{forget}_{\mathcal{E}}(P) = P' \)

Inversion of typing rule.

\textbf{SUBCASE:} \( e_1 \) is a value  
It is only possible to derive \textbf{active}(e).

When \( a = \bullet \), we can step.

\( a; H; \langle e, S_p \rangle \rightarrow a'; H'; \langle e', S'_p \rangle; T' \)

By rule \textbf{E-ATOMIC-EXIT}

\textbf{SUBCASE:} \( e_1 \) is not a value.
\( e_2 \) can take a step  
It is only possible to derive \textbf{active}(e).

When \( a = \bullet \), we can step.

\( a; H; \langle e, S_p \rangle \rightarrow a'; H'; \langle e', S'_p \rangle; T' \)

By rule \textbf{E-INATOMIC}

\textbf{CASE T-READ}  
So \( e = k \cdot o.f_i \).

\( \Sigma; \overrightarrow{\Delta}; \overrightarrow{\epsilon}; \overrightarrow{u} \vdash H; \overrightarrow{S_p} \)  
\( \vdash \Sigma; \Delta; \mathcal{E}; u \vdash k \cdot o : \exists x: C.[x/o]P \setminus \emptyset | u \)  
\( \text{readonly}(k_u) \implies \text{readonly}(k) \quad \vdash \Sigma; \Delta \vdash \emptyset P \quad \text{localFields}(C) = \overrightarrow{f(C)} \)  

Inversion of typing rule
\[ o \mapsto C(\ldots, f_i = k_i \cdot o_i, \ldots) @ S \]

\[ k \leq k_i \]

**Subcase:** \( \text{writes}(k_u) \)

\[ S = \text{unpacked}(k_u) \]

Only \text{not-active}(e) can be derived.

We can step when \( a = o \)

\[ a; H; (e, S_p) \rightarrow a'; H'; (e', S_p'); T' \]

By rule E-READ-RW

**Subcase:** \( \text{readonly}(k_u) \)

\[ S = \text{unpacked}(s) \]

\[ k = \text{immutable}|\text{pure} \]

Only \text{not-active}(e) can be derived.

We can step when \( a = o \)

\[ a; H; (e, S_p) \rightarrow a'; H'; (e', S_p'); T' \]

By rule E-READ-R

**Case T-Let**

So \( e = \text{let } x = e_1 \text{ in } e_2. \)

\[ \Sigma; \Delta^c; u \vdash H; S_p \]

Assumption

\[ \vdash \Sigma; (\Delta_1, \Delta_2); E; u \vdash \text{let } x = e_1 \text{ in } e_2 : E \setminus \omega_1 \cup \omega_2 | u' \]

Assumption

\[ \Sigma; \Delta_2; P \vdash E \]

\[ \vdash \Sigma; \Delta_1; E; u \vdash \exists x : T.P \setminus \omega_1 | u_2 \quad x : C; \Sigma; P'; E; u_2 \vdash e_2 : E \omega_2 | u' \]

No permissions for \( \omega_1 \) in \( \Delta_2 \)

Inversion of typing rule

**Subcase:** \( e_1 \) is a value.

\[ e_1 = k \cdot o \]

\[ k' \cdot o \in S_p \quad k \leq k' \]

\[ o \in H \]

Only \text{not-active}(e) possible when \( e_1 \) is a value.

We can step when \( a = o \).

\[ a; H; (e, S_p) \rightarrow a'; H'; (e', S_p'); T' \]

By inversion of T-LOC and \( \Sigma; \Delta \vdash S_p \)

Heap condition 1

By rule E-LET-V

**Subcase:** \( e_1 \) is not a value.

\[ e_1 \text{ is well-typed} \]

\[ e_1 \text{ must step} \]

If \( \text{active}(e) \) then \( \text{active}(e_1) \)

\[ e_1 \text{ must step when } a = \bullet \]

If \( \text{not-active}(e) \) then \( \text{not-active}(e_1) \)

\[ e_1 \text{ must step when } a = o \]

Above

Induction hypothesis

active for Let

I.H

not-active for Let

I.H
\[ a; H; (e, S_p) \rightarrow a'; H'; (e', S_p'); T' \]

By rule E-LET-E

**Case T-NEW**

So \( e = \text{new } C(k \cdot o) \).

\[ \Sigma; \Delta; u^\mathcal{E} \vdash H; S_p \]
\[ \vdash \Sigma; \Delta; \mathcal{E}; u \vdash \text{new } C(k \cdot o) : \exists x : C.\text{unique} \cdot x @ s \setminus \emptyset | u \]

Assumption

\[ \vdash \Sigma; \Delta \vdash \mathcal{E} x : C \subseteq \Sigma \quad \text{init}(C) = \langle \exists f : C.P, s \rangle \]

Inversion of typing rule

\[ H; S_p \vdash [o/ʃ] P \]

Heap condition 4.c.

\[ \Sigma; \Delta \vdash S_p \]
\[ \Sigma; \Delta \vdash S_p \]

By rule E-NEW-E

**Case T-ASSIGN**

\[ \Sigma; \Delta; u^\mathcal{E} \vdash H; S_p \]
\[ \vdash \Sigma; \Delta; \mathcal{E}; k' \cdot o @ s' \vdash k \cdot o : \exists x : C'.P' \otimes [o'.f_i/x] P \setminus \{o_i.f\}|k' \cdot o' @ s' \]

Assumption

\[ \text{localFields}(C'') = \langle f : C \rangle \quad (o' : C'') \in \Sigma \quad \text{writes}(k') \]

Assumption

\[ \vdash \Sigma; \Delta \vdash \mathcal{E} x : C_i.P \]
\[ \vdash \Sigma; \Delta' \vdash \mathcal{E} [o'.f_i/x'] P' \]

Inversion of typing rule.

\[ o \mapsto C(\ldots) @ \text{unpacked}(k') \in H \]
\[ k' \in S_p \]
\[ k_i \leq k' \]

Heap condition 3

Heap condition 4.d

Heap condition 4.d

Only not-active(e) can be derived.

We can step when \( a = o \)

\[ a; H; (e, S_p) \rightarrow a'; H'; (e', S_p'); T' \]

By rule E-ASSIGN-E

**References**
