

A. PROOFS

LEMMA 2.1. *Given a type-checked program g ($\vdash g : \omega$), an expression e that type-checks ($\Delta; \Pi_1 \vdash_\omega e : \tau \dashv \Delta'; \Pi'_1$) and an environment $\Delta''; \Pi_2$, then e also type-checks with a larger set of permissions ($\Delta \cup \Delta''; \Pi_1, \Pi_2 \vdash_\omega e : \tau \dashv \Delta' \cup \Delta''; \Pi'_1, \Pi_2$) in which the unused permissions are not changed.*

PROOF. As e must have been typed using one of typing rules, it suffices to show that, for each rule, if that rule applies to e using $\Delta; \Pi_1$ (and returning $\Delta'; \Pi'_1$), it also applies to e using $\Delta \cup \Delta''; \Pi_1, \Pi_2$ (and returning $\Delta' \cup \Delta''; \Pi'_1, \Pi_2$). Additionally, we inductively assume the lemma applies to subexpressions of e . We now look at each rule as a separate case:

Unit, Num, True, False, Address

As these rules neither depend upon nor alter the environment, they type under any environment and the lemma is trivially true.

Plus $e_1 + e_2$

By hypothesis, we know that

$$\Delta; \Pi_1 \vdash_\omega e_1 + e_2 : \text{int} \dashv \Delta'; \Pi'_1$$

. To have typed this using the PLUS rule, the following must be true:

$$\Delta; \Pi_1 \vdash_\omega e_1 : \text{int} \dashv \Delta_1; \Pi''_1 \vdash_\omega e_2 : \text{int} \dashv \Delta'; \Pi'_1$$

Inductively,

$$\Delta \cup \Delta''; \Pi_1, \Pi_2 \vdash_\omega e_1 : \text{int} \dashv \Delta_1 \cup \Delta''; \Pi''_1, \Pi_2$$

$$\Delta_1 \cup \Delta''; \Pi''_1, \Pi_2 \vdash_\omega e_2 : \text{int} \dashv \Delta' \cup \Delta''; \Pi'_1, \Pi_2$$

So, by applying the rule PLUS, we arrive at

$$\Delta \cup \Delta''; \Pi_1, \Pi_2 \vdash_\omega e_1 + e_2 : \text{int} \dashv \Delta' \cup \Delta''; \Pi'_1, \Pi_2$$

Equal $e_1 = e_2$

The types are different than in PLUS, but the environment is treated in the same manner. Thus the proof is essentially the same.

Read $e.f$

By hypothesis:

$$\Delta; \Pi_1 \vdash_\omega e.f : \tau \dashv \Delta'; \Pi'_1$$

As we assume we applied the READ rule, the following must be true:

$$\Delta; \Pi_1 \vdash_\omega e : \text{ptr}(l) \dashv \Delta'; \Pi'_1$$

$$\Pi_1 = l.f : \tau \setminus \{ \dots \}, \Pi$$

$$\tau \sim \tau$$

By induction,

$$\Delta \cup \Delta''; \Pi_1, \Pi_2 \vdash_\omega e : \text{ptr}(l) \dashv \Delta' \cup \Delta''; \Pi'_1, \Pi_2.$$

From the second fact and from the definition of the comma operator, we can conclude

$$\Pi_1, \Pi_2 = l.f : \tau \setminus \{ \dots \}, \Pi, \Pi_2.$$

This gives us sufficient information to apply the READ rule with the larger environment:

$$\Delta \cup \Delta''; \Pi_1, \Pi_2 \vdash_\omega e.f : \tau \dashv \Delta' \cup \Delta''; \Pi'_1, \Pi_2$$

New $\text{new}\{f_i \mid 1 \leq i \leq n\}$

The only requirement for this rule to have been applied is for r to be fresh (given context Δ . We can safely assume that r is still fresh given context $\Delta \cup \Delta''$ because we can type the expression in the original environment with such a r . Thus, we can successfully apply the NEW rule despite the altered environment.

Write $e_1.f := e_2$

The typing prerequisites follow inductively as with PLUS, one may add extra permissions to both sides of an equation here as in READ and the storage compatibility requirements are unaffected by the permissions. Thus we can prove all that is necessary to apply WRITE in the new environment.

Seq $e; e'$

Again, this is essentially the same as PLUS.

If $\text{if } e_0 \text{ then } e_1 \text{ else } e_2$

If this rule were successfully applied, the following must be true:

$$\Delta; \Pi_1 \vdash_\omega e_0 : \text{bool} \dashv \Delta_0; \Pi''_1$$

$$\Delta_0; \Pi''_1 \vdash_\omega e_1 : \text{unit} \dashv \Delta_1; \Pi'_{1a}$$

$$\Delta_0; \Pi''_1 \vdash_\omega e_2 : \text{unit} \dashv \Delta_2; \Pi'_{1b}$$

$$\Delta'; \Pi'_1 = \Delta_1; \Pi'_{1a} \vee \Delta_2; \Pi'_{1b}$$

By induction,

$$\Delta \cup \Delta''; \Pi_1, \Pi_2 \vdash_\omega e_0 : \text{bool} \dashv \Delta_0 \cup \Delta''; \Pi''_1, \Pi_2$$

$$\Delta_0 \cup \Delta''; \Pi''_1, \Pi_2 \vdash_\omega e_1 : \text{unit} \dashv \Delta_1 \cup \Delta''; \Pi'_{1a}, \Pi_2$$

$$\Delta_0 \cup \Delta''; \Pi''_1, \Pi_2 \vdash_\omega e_2 : \text{unit} \dashv \Delta_2 \cup \Delta''; \Pi'_{1b}, \Pi_2$$

Because $\Delta''; \Pi_2$ is an environment, if r appears in Π_2 then $r \in \Delta''$. From the definition of the \vee operator, we can also conclude that $\Delta_i \cup \Delta''; \Pi_{1x} = \sigma_i(\Delta_0 \cup \Delta''); \sigma_i \Pi''_1$, and therefore $\Delta_i \cup \Delta''; \Pi_{1x}, \Pi_2 = \sigma_i(\Delta_0 \cup \Delta''); (\sigma_i \Pi''_1), \Pi_2$. But because every r that appears in Π_2 must also be in $\Delta'' \subseteq (\Delta_0 \cup \Delta'') \cap (\Delta'_2 \cup \Delta'')$, $\sigma_i \Pi_2 = \Pi_2$. Therefore, $\Delta'_i \cup \Delta''; \Pi_{1x}, \Pi_2 = \sigma_i(\Delta_0 \cup \Delta_0); \sigma_i(\Pi''_1, \Pi_2)$. We further assume the fresh variables in Δ' could be chosen so as not to intersect with Δ'' . If not, simply rename to new fresh variables in the original typing proof. Therefore the variables in $\Delta' - ((\Delta_1 \cap \Delta_2) \cup \Delta'')$ are also fresh, and we may apply the definition of \vee to determine that

$$\Delta' \cup \Delta''; \Pi'_1, \Pi_2 =$$

$$\Delta_1 \cup \Delta''; \Pi'_{1a}, \Pi_2 \vee \Delta_2 \cup \Delta''; \Pi'_{1b}, \Pi_2$$

This, combined with the earlier inductive results is sufficient to apply the IF rule and type check the expression in the new environment.

IfEqual $\text{if } e = e' \text{ then } e_1 \text{ else } e_2$

By hypothesis:

$$\Delta; \Pi_1 \vdash_\omega \text{if } e = e' \text{ then } e_1 \text{ else } e_2 : \text{unit} \dashv \Delta'; \Pi'_1$$

As we are assuming the IFEQUAL rule was applied, we know the following must have been true:

$$\begin{aligned} & \Delta; \Pi_1 \vdash_{\omega} e : \text{ptr}(l) \dashv \Delta_{0_1}; \Pi_{1_1} \\ & \Delta_{0_1}; \Pi_{1_1} \vdash_{\omega} e' : \text{ptr}(l') \dashv \Delta_{0_2}; \Pi_{1_2} \\ & \Delta_{0_2}; (\Pi_{1_2}, l = l') \vdash_{\omega} e_1 : \text{unit} \dashv \Delta_1; \Pi_{1_a} \\ & \Delta_{0_2}; (\Pi_{1_2}; l \neq l') \vdash_{\omega} e_2 : \text{unit} \dashv \Delta_2; \Pi_{1_b} \\ & \Delta'; \Pi_1 = \Delta_1; \Pi_{1_a} \vee \Delta_2; \Pi_{1_b} \end{aligned}$$

By induction:

$$\begin{aligned} & \Delta \cup \Delta''; \Pi_1, \Pi_2 \vdash_{\omega} e : \text{ptr}(l) \dashv \Delta_{0_1} \cup \Delta''; \Pi_{1_1}, \Pi_2 \\ & \Delta_{0_1} \cup \Delta''; \Pi_{1_1}, \Pi_2 \vdash_{\omega} e' : \text{ptr}(l') \dashv \Delta_{0_2} \cup \Delta''; \Pi_{1_2}, \Pi_2 \\ & \Delta_{0_2} \cup \Delta''; (\Pi_{1_2}, l = l'), \Pi_2 \vdash_{\omega} e_1 : \text{unit} \dashv \\ & \quad \Delta_1 \cup \Delta''; \Pi_{1_a}, \Pi_2 \\ & \Delta_{0_2} \cup \Delta''; (\Pi_{1_2}; l \neq l'), \Pi_2 \vdash_{\omega} e_2 : \text{unit} \dashv \\ & \quad \Delta_2 \cup \Delta''; \Pi_{1_b}, \Pi_2 \end{aligned}$$

Further, we can reiterate the argument from the IF case to conclude that

$$\begin{aligned} & \Delta' \cup \Delta''; \Pi'_1, \Pi_2 = \\ & \quad \Delta_1 \cup \Delta''; \Pi'_{1_a}, \Pi_2 \vee \Delta_2 \cup \Delta''; \Pi'_{1_b}, \Pi_2 \end{aligned}$$

Therefore, we can apply the IFEQUAL rule to type the expression in the new environment.

IfTrue if true then e_1 else e_2

Trivially true by induction on e_1 .

IfFalse if false then e_1 else e_2

Trivially true by induction on e_2 .

Call call p

If we assume the CALL rule had been applied to type our expression, then its prerequisites must be true. As before, we may assume that Δ' is fresh with regard to $\Delta \cup \Delta''$ as well as Δ . Thus the only prerequisite for this rule that requires the environment (directly) is

$$\sigma_1 : \Delta_1 \rightarrow \Delta$$

However, it is clear from the definition of substitutions that one may increase the range of σ_1 without altering its definition. In particular, one may define

$$\sigma'_1 : \Delta_1 \rightarrow \Delta \cup \Delta''$$

where σ'_1 and σ_1 are identical maps. By including the Π_2 from the lemma in the Π_3 from the CALL rule, we may directly apply the CALL rule in the environment $\Delta \cup \Delta''; \Pi_1, \Pi_2$ and get back the environment $\Delta \cup \Delta' \cup \Delta''; \Pi'_1, \Pi_2$ (where Π'_1 in the lemma equals $\sigma_1^{(\cdot)} \Pi_2, \Pi_3$ in the CALL rule).

Nest nest $e.f$ in $e'.f'$

By hypothesis:

$$\Delta; \Pi_1 \vdash_{\omega} \text{nest } e.f \text{ in } e'.f' : \text{unit} \dashv \Delta'; (k' :$$

$$\tau' \setminus \{k_1 : \tau_1, \dots, k_n : \tau_n\}; k : \tau \prec k' \wedge k \neq k_n), \Pi''_1$$

Therefore, the following must have been true to apply the NEST rule:

$$\Delta; \Pi_1 \vdash_{\omega} e : \text{ptr}(l) \dashv \Delta_0; \Pi_0$$

$$\Delta_0; \Pi_0 \vdash_{\omega} e' : \text{ptr}(l') \dashv \Delta'; \Pi'_1$$

$$k = l.f$$

$$k' = l'.f'$$

$$\Pi'_1 = (k : \tau \setminus \{k_1 : \tau_1, \dots, k_n : \tau_n\}; k \neq k_1 \wedge \dots \wedge k \neq k_n), k' :$$

$$\tau' \setminus \{k_1 : \tau_1, \dots, k_n : \tau_n\}, \Pi''_1$$

By induction,

$$\Delta \cup \Delta''; \Pi_1, \Pi_2 \vdash_{\omega} e : \text{ptr}(l) \dashv \Delta_0 \cup \Delta''; \Pi_0, \Pi_2$$

$$\Delta_0 \cup \Delta''; \Pi_0, \Pi_2 \vdash_{\omega} e' : \text{ptr}(l') \dashv \Delta' \cup \Delta''; \Pi'_1, \Pi_2$$

And, from the definition of the comma operator, we can deduce that,

$$\Pi'_1, \Pi_2 = (k : \tau \setminus \{k_1 : \tau_1, \dots, k_n : \tau_n\}; k \neq k_1 \wedge \dots \wedge k \neq k_n), k' :$$

$$\tau' \setminus \{k_1 : \tau_1, \dots, k_n : \tau_n\}, \Pi''_1, \Pi_2$$

This is sufficient information to apply the NEST rule and type the expression in the new environment.

□

We also have a substitution lemma:

LEMMA 2.2. *Given a program g as an expression e that type-checks in an environment $E = (\Delta; \Pi)$ ($E \vdash_{\omega} e : \tau \dashv E'$), and given a substitution $\sigma : \Delta_1 \rightarrow \Delta_2$ where $\Delta_1 \uplus \Delta_2$ is a partition of Δ , then e also type checks in the substituted environment $\sigma E = (\Delta_2; \sigma \Pi)$ ($\sigma E \vdash_{\omega} e : \sigma \tau \dashv \sigma E'$).*

PROOF. It is worth mentioning that $\sigma \text{unit} = \text{unit}$, $\sigma \text{int} = \text{int}$, and $\sigma \text{bool} = \text{bool}$ for any substitution σ , as substitutions apply only to location variables. Further, if τ is a pointer type, then $\sigma \tau$ is also, because the range of σ is confined to location variables and literals. Thus, if $\tau_1 \sim \tau_2$, then $\sigma \tau_1 \sim \sigma \tau_2$ for any substitution σ .

Unit, Num, True, False, Address

As these rules neither depend upon nor alter the environment, they type under any environment and the lemma is trivially true.

Plus $e_1 + e_2$

Follows immediately from applying induction twice.

Equal $e_1 = e_2$

By hypothesis, this rule was applied, so

$$E \vdash_{\omega} e_1 : \tau_1 \dashv E' \vdash_{\omega} e_2 : \tau_2 \dashv E''$$

$$\tau_1 \sim \tau_2$$

Then, by induction:

$$\sigma E \vdash_{\omega} e_1 : \sigma\tau_1 \dashv \sigma E' \vdash_{\omega} e_2 : \sigma\tau_2 \dashv \sigma E''$$

As discussed above, $\sigma\tau_1 \sim \sigma\tau_2$. Therefore, we can apply the EQUAL rule to get

$$\sigma E \vdash_{\omega} e_1=e_2 : \sigma\text{bool} \dashv \sigma E''$$

Read $e.f$

We may assume the prerequisites for the READ rule are true. Then, by induction,

$$\sigma E \vdash_{\omega} e : \sigma\text{ptr}(\rho) \dashv \sigma\Delta'; \sigma\Pi'$$

Directly applying the substitution to both sides of the equation:

$$\sigma\Pi' = \sigma\rho.f : \sigma\tau \setminus \{\sigma\dots\}, \sigma\Pi_1$$

And as storage compatibility holds over substitutions,

$$\sigma\tau \sim \sigma\tau$$

Therefore we may apply the READ rule to get the desired result.

New $\text{new}\{f_i \mid 1 \leq i \leq n\}$

As the new reference variable introduced by this rule is fresh, the rule may be applied in the substituted environment. Since the fresh variable is in neither the domain nor the range of the substitution (nor are its concrete field types), the appropriate substituted environment results.

Write $e_1.f := e_2$

The typing preconditions for this rule in the substituted environment follow by double induction, as with PLUS. The equality and storage compatibility preconditions follow as in the READ rule.

Seq $e;e'$

Follows immediately from applying induction twice.

If $\text{if } e_0 \text{ then } e_1 \text{ else } e_2$

By hypothesis,

$$E \vdash_{\omega} \text{if } e_0 \text{ then } e_1 \text{ else } e_2 : \text{unit} \dashv E''$$

The following must have been true to apply this rule:

$$E \vdash_{\omega} e_0 : \text{bool} \dashv E'$$

$$E' \vdash_{\omega} e_1 : \text{unit} \dashv E_1$$

$$E' \vdash_{\omega} e_2 : \text{unit} \dashv E_2$$

$$E'' = E_1 \vee E_2$$

By induction,

$$\sigma E \vdash_{\omega} e_0 : \sigma\text{bool} \dashv \sigma E'$$

$$\sigma E' \vdash_{\omega} e_1 : \sigma\text{unit} \dashv \sigma E_1$$

$$\sigma E' \vdash_{\omega} e_2 : \sigma\text{unit} \dashv \sigma E_2$$

We can apply some of these substitutions:

$$\sigma E \vdash_{\omega} e_0 : \text{bool} \dashv \sigma E'$$

$$\sigma E' \vdash_{\omega} e_1 : \text{unit} \dashv \sigma E_1$$

$$\sigma E' \vdash_{\omega} e_2 : \text{unit} \dashv \sigma E_2$$

If we can show that $\sigma E'' = \sigma E_1 \vee \sigma E_2$, we can then apply the IF rule to prove that

$$\sigma E \vdash_{\omega} \text{if } e_0 \text{ then } e_1 \text{ else } e_2 : \sigma\text{unit} \dashv \sigma E''$$

We therefore argue that if $E'' = E_1 \vee E_2$, $\sigma E'' = \sigma E_1 \vee \sigma E_2$.

If $(\Delta'; \Pi') = (\Delta_1; \Pi_1) \vee (\Delta_2; \Pi_2)$ then by definition, there are substitutions σ_1 and σ_2 such that

$$\Delta_1; \Pi_1 = \sigma_1\Delta'; \sigma_1\Pi'$$

$$\Delta_2; \Pi_2 = \sigma_2\Delta'; \sigma_2\Pi'$$

$$\Delta = \Delta_1 \cap \Delta_2$$

$$\Delta' - \Delta \text{ fresh}$$

$$r \in \Delta \Rightarrow \sigma_i r = r$$

We assume that the domain of σ_1 and σ_2 is restricted to Δ' . If not, we can choose new σ_i which are so restricted, and which are sufficient for the definition of \vee . As such, the only variables in the domains of both σ and σ_i are those in Δ , which σ_1 and σ_2 do not alter.

As $\Delta' - \Delta$ fresh, and as σ 's domain was defined on variables existing before the \vee operation, σ can only affect the variables in Δ . It acts as the identity on the fresh variables in Δ' . Similarly, the range of σ is also confined to Δ . Thus $\sigma\Delta' = (\Delta' - \Delta) \cup \sigma\Delta$ and $\sigma\Delta' - \sigma\Delta = \Delta' - \Delta$ fresh.

As the domain and range of σ are disjoint by definition, σr will be outside the domain of σ for any r . Thus $\sigma = \sigma \circ \sigma$. Additionally, because σ_1 and σ_2 behave as the identity for location variables in Δ and because the range of σ cannot include the fresh variables which make up the (non-identity) domain of σ_1 and σ_2 , $\sigma \circ \sigma_i \circ \sigma = \sigma \circ \sigma \circ \sigma_i = \sigma \circ \sigma_i$ ($i = 1$ or $i = 2$).

We can now define two new substitutions, $\sigma'_1 = \sigma \circ \sigma_1$ and $\sigma'_2 = \sigma \circ \sigma_2$. Clearly, $\sigma\Delta_1; \sigma\Pi_1 = \sigma\sigma_1\Delta'; \sigma\sigma_1\Pi' = \sigma\sigma_1\sigma\Delta'; \sigma\sigma_1\sigma\Pi' = \sigma'_1\sigma\Delta'; \sigma'_1\sigma\Pi'$. Similarly, $\sigma\Delta_2; \sigma\Pi_2 = \sigma'_2\sigma\Delta'; \sigma'_2\sigma\Pi'$.

From $\Delta = \Delta_1 \cap \Delta_2$, we can conclude that $\sigma\Delta = \sigma(\Delta_1 \cap \Delta_2) = \sigma\Delta_1 \cap \sigma\Delta_2$. Then, for all $r \in \sigma\Delta$, there exists a $r' \in \Delta$ where $\sigma r' = r$. Also, $\forall r \in \sigma\Delta$

$$\begin{aligned} \sigma'_1 r &= \sigma\sigma_1 r \\ &= \sigma\sigma_1\sigma r' \quad (\text{definition of } r') \\ &= \sigma\sigma_1 r' \quad (\text{from above}) \\ &= \sigma r' \quad (\sigma_1 r' = r' \quad \forall r' \in \Delta) \\ &= r \end{aligned}$$

An identical argument may be applied to get $\sigma'_2 r = r \quad \forall r \in \sigma\Delta$. These facts can now be used to apply the definition of \vee to determine that if $(\Delta'; \Pi') = (\Delta_1; \Pi_1) \vee (\Delta_2; \Pi_2)$, then $(\sigma\Delta'; \sigma\Pi') = (\sigma\Delta_1; \sigma\Pi_1) \vee (\sigma\Delta_2; \sigma\Pi_2)$.

In particular, for the IF rule, $\sigma E'' = \sigma E_1 \vee \sigma E_2$. This, in turn, is sufficient (combined with the earlier inductive results) to apply the IF rule and conclude that

$$\sigma E \vdash_{\omega} \text{if } e_0 \text{ then } e_1 \text{ else } e_2 : \sigma \text{unit} \dashv \sigma E''$$

IFEQUAL if $e=e'$ then e_1 else e_2

By hypothesis,

$$E \vdash_{\omega} \text{if } e=e' \text{ then } e_1 \text{ else } e_2 : \text{unit} \dashv E''$$

As we are assuming the IFEQUAL rule was applied, the following must have been true:

$$E \vdash_{\omega} e : \text{ptr}(\rho) \dashv E' \vdash_{\omega} e' : \text{ptr}(\rho') \dashv \Delta; \Pi$$

$$\Delta; (\rho = \rho', \Pi) \vdash_{\omega} e_1 : \text{unit} \dashv E_1$$

$$\Delta; (\rho \neq \rho', \Pi) \vdash_{\omega} e_2 : \text{unit} \dashv E_2$$

$$E'' = E_1 \vee E_2$$

After applying induction (and recalling that $\sigma \text{unit} = \text{unit}$ and $\sigma \text{bool} = \text{bool}$)

$$\sigma E \vdash_{\omega} e : \sigma \text{ptr}(\rho) \dashv \sigma E' \vdash_{\omega} e' : \sigma \text{ptr}(\rho') \dashv \sigma \Delta; \sigma \Pi$$

$$\sigma \Delta; (\sigma \rho = \sigma \rho', \sigma \Pi) \vdash_{\omega} e_1 : \sigma \text{unit} \dashv \sigma E_1$$

$$\sigma \Delta; (\sigma \rho \neq \sigma \rho', \sigma \Pi) \vdash_{\omega} e_2 : \sigma \text{unit} \dashv \sigma E_2$$

As $E'' = E_1 \vee E_2$, we can argue again that $\sigma E'' = \sigma E_1 \vee \sigma E_2$. Therefore, we can apply the IFEQUAL rule again to get

$$\sigma E \vdash_{\omega} \text{if } e=e' \text{ then } e_1 \text{ else } e_2 : \sigma \text{unit} \dashv \sigma E''$$

which is what needs to be proven.

IFTRUE if true then e_1 else e_2

Follows immediately from induction on e_1 .

IFFALSE if false then e_1 else e_2

Follows immediately from induction on e_2 .

CALL call p

By hypothesis,

$$\Delta; \sigma_1 \Pi_1, \Pi_3 \vdash_{\omega} \text{call } p : \text{unit} \dashv \Delta \cup \Delta'; \sigma_1 \Pi_2, \Pi_3$$

Therefore, because we checked this using the CALL rule:

$$\omega(p) = \forall \Delta_1. \Pi_1 \rightarrow \exists \Delta_2. \sigma_2 \Pi_2$$

$$\sigma_1 : \Delta_1 \rightarrow \Delta$$

$$\Delta' \text{ fresh}$$

$$\sigma_2 : \Delta' \rightarrow \Delta_2$$

We can safely assume the variables in Δ , Δ_2 , Δ_1 , and Δ' are all disjoint as we may always find a typing using disjoint contexts. Therefore, $\sigma \sigma_1 : \Delta_1 \rightarrow \sigma \Delta$. Also, $\sigma \Delta' = \Delta'$, so $\sigma \Delta$ fresh and $\sigma_2 : \sigma \Delta' \rightarrow \Delta_2$. We thus

have:

$$\omega(p) = \forall \Delta_1. \Pi_1 \rightarrow \exists \Delta_2. \sigma_2 \Pi_2$$

$$\sigma \sigma_1 : \Delta_1 \rightarrow \sigma \Delta$$

$$\sigma \Delta' \text{ fresh}$$

$$\sigma_2 : \sigma \Delta' \rightarrow \Delta_2$$

We can now apply the CALL rule to get:

$$\sigma \Delta; \sigma \sigma_1 \Pi_1, \sigma \Pi_3 \vdash_{\omega} \text{call } p : \text{unit} \dashv$$

$$\sigma \Delta \cup \sigma \Delta'; \sigma \sigma_1 \Pi_2, \sigma \Pi_3$$

Nest nest $e.f$ in $e'.f'$

All the equations still hold true if the substitution is applied equally to both sides, so this rule naturally falls out by inducting twice, then substituting over the equations, similar to WRITE. \square

Next, we prove a narrowing rule for consistency:

LEMMA 2.3. *If we have a memory and adoption information consistent with an environment ($\mu; a \vdash \Delta; \Pi_1, \Pi_2$ consistent), then they are also consistent with an environment with fewer permissions ($\mu; a \vdash \Delta; \Pi_1$).*

PROOF. We first prove that $\mu; A; \emptyset \vdash \pi_1, \dots, \pi_n \Downarrow \hat{\Pi}$ if and only if $\mu; A; \emptyset \vdash \pi_i \Downarrow \hat{\Pi}_i$ and $\hat{\Pi} = \biguplus_i \hat{\Pi}_i$. The result is susceptible to a simple proof by induction using $n \in \{0, 1\}$ as base cases since there is only one rule for show consistency of a set of permissions, and the B sets are empty, the non-determinism of the split of permissions falls away due to the associativity of the union of disjoint sets.

Given this result, it is easy to see that the consistency of the smaller set of permissions can be established using the same substitution and assumption sets; we simply end up with a subset of the requirements on the memory that must be fulfilled. Thus the result follows. \square

An important property of this definition is that the transformed environment will be flattened to a subset of the flat permissions:

LEMMA 2.4. *If we have $\mu; a \vdash \Delta; \Pi$ consistent using $\sigma; (A_{\prec}, A_T)$ such that $\mu; A; \emptyset \vdash \sigma \Pi \Downarrow \hat{\Pi}$ and $(\Delta; \Pi) \geq (\Delta'; \Pi')$ then $\mu; A'; \emptyset \vdash \sigma' \Pi' \Downarrow \hat{\Pi}'$ and $\hat{\Pi} \supseteq \hat{\Pi}'$.*

PROOF. From the definition of the transformation relation, $\mu; a \vdash \Delta'; \Pi'$ consistent using $\sigma'; (A_{\prec}, A'_T)$. Therefore, by the definition of consistency, there must be some $\hat{\Pi}'$ such that $\mu; A; \emptyset \vdash \sigma' \Pi' \Downarrow \hat{\Pi}'$. Now suppose $l : \tau_{\text{atom}} \in \hat{\Pi}'$. Then, $\vdash \mu(l) : \tau_{\text{atom}}$.

Also, $\exists \tau'_{\text{atom}} \ l : \tau'_{\text{atom}} \in \hat{\Pi}$. If not, we can define $\mu' = \mu[l \mapsto v]$ where $\vdash v : \tau''_{\text{atom}}$ and $\tau''_{\text{atom}} \neq \tau_{\text{atom}}$. Then $\mu'; a \vdash \Delta; \Pi$ consistent using $\sigma; (A_{\prec}, A_T)$ because all the preconditions established by μ still hold. But $\mu'; a \vdash \Delta'; \Pi'$ consistent using $\sigma'; (A_{\prec}, A_T)$ is clearly not true, as we will not be able to show that $\vdash \mu'(l) : \tau_{\text{atom}}$ (because, of course, $\vdash \mu'(l) : \tau''_{\text{atom}}$).

But from $\mu; a \vdash \Delta; \Pi$ consistent using $\sigma; (A_{\prec}, A_T)$ such that $\mu; A; \emptyset \vdash \sigma \Pi \Downarrow \hat{\Pi}$, we know that $\vdash \mu(l) : \tau'_{\text{atom}}$. But then τ_{atom} must equal τ'_{atom} , so $l : \tau_{\text{atom}} \in \hat{\Pi}$. \square

LEMMA 2.5. *If we have two transformations: $\Delta; \Pi_1 \geq \Delta'_1; \Pi'_1$ and $\Delta; \Pi_2 \geq \Delta'_2; \Pi'_2$ where the fresh variables introduced are disjoint $(\Delta'_1 - \Delta) \cap (\Delta'_2 - \Delta) = \emptyset$, then the two transformations can be merged: $\Delta; \Pi_1, \Pi_2 \geq \Delta'; \Pi'_1, \Pi'_2$ where $\Delta' = \Delta'_1 \cup \Delta'_2$.*

PROOF. Suppose we have some $\mu, a, \sigma, A_{\prec}, A_T$ such that $\mu; a \vdash \Delta; \Pi_1, \Pi_2$ consistent using $\sigma; (A_{\prec}, A_T)$. Then, from Lemma 2.3 and the definition of transform (\geq), there are A'_T and σ_0 such that $\mu; a \vdash \Delta; \Pi_1$ consistent using $\sigma_0; (A_{\prec}, A'_T)$. Thus, using the definition of transform again, there exists A_{T1} and σ_1 such that $\mu; a \vdash \Delta'_1; \Pi'_1$ consistent using $\sigma_1; (A_{\prec}, A_{T1})$. Similarly, there exists A_{T2} and σ_2 such that $\mu; a \vdash \Delta'_2; \Pi'_2$ consistent using $\sigma_2; (A_{\prec}, A_{T2})$.

From the definition of memory consistency we may conclude that the following facts must be true:

a is acyclic

$\sigma_1 : \Delta'_1 \rightarrow \emptyset$

$\sigma_2 : \Delta'_2 \rightarrow \emptyset$

$\exists \tau. (l : \tau \prec l') \in A_{\prec} \Leftrightarrow (l \prec l') \in a$

$\mu; A_{\prec}, A_{T1}; \emptyset \vdash \sigma_1 \Pi'_1 \Downarrow \hat{\Pi}_1$

$\mu; A_{\prec}, A_{T2}; \emptyset \vdash \sigma_2 \Pi'_2 \Downarrow \hat{\Pi}_2$

$(l : \tau_{\text{atom}}) \in \hat{\Pi}_1 \Rightarrow \vdash \mu(l) : \tau_{\text{atom}}$

$(l : \tau_{\text{atom}}) \in \hat{\Pi}_2 \Rightarrow \vdash \mu(l) : \tau_{\text{atom}}$

$t(\nu_1, \dots, \nu_n) \in A_{T1} \Rightarrow A_{\prec}, A_{T1} \vdash [r_1 \rightarrow \nu_1, \dots, r_n \rightarrow \nu_n]T(t) = \text{true}$

$t(\nu_1, \dots, \nu_n) \in A_{T2} \Rightarrow A_{\prec}, A_{T2} \vdash [r_1 \rightarrow \nu_1, \dots, r_n \rightarrow \nu_n]T(t) = \text{true}$

Let us define $\sigma' : \Delta' \rightarrow \emptyset$ as follows:

$$\sigma'(r) = \begin{cases} \sigma(r) & \text{if } r \in \Delta \cap \Delta' \\ \sigma_1(r) & \text{if } r \in \Delta'_1 - \Delta \\ \sigma_2(r) & \text{if } r \in \Delta'_2 - \Delta \end{cases}$$

This substitution is well-defined, as $\Delta'_1 - \Delta$ and $\Delta'_2 - \Delta$ are disjoint. Because $\sigma_1 \supseteq \sigma_0 \supseteq \sigma$,⁶ $\forall r \in \Delta'_1 \ \sigma'(r) = \sigma_1(r)$ and thus $\sigma'(\Pi'_1) = \sigma_1(\Pi'_1)$. Similarly, $\forall r \in \Delta'_2 \ \sigma'(r) = \sigma_2(r)$ and thus $\sigma'(\Pi'_2) = \sigma_2(\Pi'_2)$. Also, from the definition of σ' , $r \in \Delta \Rightarrow \sigma' r = \sigma r$, so $\sigma' \supseteq \sigma$.

Suppose $A_{\prec}, A_T \vdash \Gamma = b$. Then $A_{\prec}, A_T \cup A'_T \vdash \Gamma = b$. Why? Because the only effect adding assumptions to A_T can have on the consistency of Γ is with the rule CB-AXIOM T which would serve to make something which had been undefined true. But no truth values were undefined in the original proof (or it wouldn't have been a proof). Therefore, the same proof may be reiterated with extra unused assumptions. This in turn implies that if $\mu; A_{\prec}, A_T; \emptyset \vdash \sigma \Pi \Downarrow \hat{\Pi}$, then $\mu; A_{\prec}, A_T \cup A'_T; \emptyset \vdash \sigma \Pi \Downarrow \hat{\Pi}$ because the proof trees for the boolean formulae will not change as above, and the remainder of the permission consistency rules do not refer to A_T and so will not be affected by its contents. Again, the

⁶That the \supseteq relation on substitutions is transitive follows immediately from its definition

same proof may be reused within the larger set of assumptions.

We may therefore modify the above facts:

$\mu; A_{\prec}, A_{T1} \cup A_{T2}; \emptyset \vdash \sigma \Pi' \Downarrow \hat{\Pi}_1$

$\mu; A_{\prec}, A_{T1} \cup A_{T2}; \emptyset \vdash \sigma \Pi'_2 \Downarrow \hat{\Pi}_2$

$(l : \tau_{\text{atom}}) \in \hat{\Pi}_1 \Rightarrow \vdash \mu(l) : \tau_{\text{atom}}$

$(l : \tau_{\text{atom}}) \in \hat{\Pi}_2 \Rightarrow \vdash \mu(l) : \tau_{\text{atom}}$

$t(\nu_1, \dots, \nu_n) \in A_{T1} \Rightarrow A_{\prec}, A_{T1} \cup A_{T2} \vdash [r_1 \rightarrow \nu_1, \dots, r_n \rightarrow \nu_n]T(t) = \text{true}$

$t(\nu_1, \dots, \nu_n) \in A_{T2} \Rightarrow A_{\prec}, A_{T1} \cup A_{T2} \vdash [r_1 \rightarrow \nu_1, \dots, r_n \rightarrow \nu_n]T(t) = \text{true}$

the first two may be combined using the CP-UNION rule to get:

$\mu; A_{\prec}, A_{T1} \cup A_{T2}; \emptyset \vdash \sigma_1 \Pi'_1, \sigma_2 \Pi'_2 \Downarrow \hat{\Pi}_1, \hat{\Pi}_2$

As discussed above, this is equivalent to

$\mu; A_{\prec}, A_{T1} \cup A_{T2}; \emptyset \vdash \sigma'(\Pi'_1, \Pi'_2) \Downarrow \hat{\Pi}_1, \hat{\Pi}_2$

We can further use straightforward logic to combine pairs of implications:

$(l : \tau_{\text{atom}}) \in \hat{\Pi}_1, \hat{\Pi}_2 \Rightarrow \vdash \mu(l) : \tau_{\text{atom}}$

$t(\nu_1, \dots, \nu_n) \in A_{T1} \cup A_{T2} \Rightarrow A_{\prec}, A_{T1} \cup A_{T2} \vdash [r_1 \rightarrow \nu_1, \dots, r_n \rightarrow \nu_n]T(t) = \text{true}$

$t(\nu_1, \dots, \nu_n) \in A_{T2} \Rightarrow A_{\prec}, A_{T1} \cup A_{T2} \vdash [r_1 \rightarrow \nu_1, \dots, r_n \rightarrow \nu_n]T(t) = \text{true}$

We now have enough facts to directly apply the definition of memory consistency and get:

$\mu; a \vdash \Delta'; \Pi'_1, \Pi'_2$ consistent using $\sigma; (A_{\prec}, A_{T1} \cup A_{T2})$

Thus, for any $\mu, a, \sigma, A_{\prec}, A_T$ such that $\mu; a \vdash \Delta; \Pi_1, \Pi_2$ consistent using we can produce an $A_{T1} \cup A_{T2}$ such that $\mu; a \vdash \Delta'; \Pi'_1, \Pi'_2$ consistent using $\sigma; (A_{\prec}, A_{T1} \cup A_{T2})$. Therefore, by definition, $\Delta; \Pi_1, \Pi_2 \geq \Delta'; \Pi'_1, \Pi'_2$. \square

After adding TRANSFORM, we must re-prove Lemmas 2.1 and 2.2.

LEMMA 2.6 (2.1). *Given a transformation $(\Delta; \Pi \geq \Delta'; \Pi')$ and an environment $(\Delta_1; \Pi_1)$, the corresponding larger sets also form a transformation $(\Delta \cup \Delta_1; \Pi, \Pi_1 \geq \Delta' \cup \Delta_1; \Pi', \Pi_1)$. Therefore, given a type-checked program $g \ (\vdash g : \omega)$, an expression e that type-checks $(\Delta; \Pi_1 \geq \Delta''; \Pi''_1 \vdash_{\omega} e : \tau \dashv \Delta'''; \Pi'''_1 \geq \Delta'; \Pi'_1)$ under TRANSFORM, and an environment $\Delta_2; \Pi_2$, then e also type-checks with a larger set of permissions $(\Delta \cup \Delta_2; \Pi_1, \Pi_2 \geq \Delta'' \cup \Delta_2; \Pi''_1, \Pi_2 \vdash_{\omega} e : \tau \dashv \Delta''' \cup \Delta_2; \Pi'_1, \Pi'_2 \geq \Delta' \cup \Delta_2; \Pi'_1, \Pi_2)$ in which the unused permissions are not changed. That is, Lemma 2.1 holds even with the TRANSFORM rule added.*

PROOF. It should be clear that adding extra, unused variables will not affect consistency. Thus we may conclude that $(\Delta \cup \Delta_1; \Pi \geq \Delta' \cup \Delta_1; \Pi')$. Also, trivially $\Delta_1; \Pi_1 \geq \Delta_1; \Pi_1$ as whenever $\Delta_1; \Pi_1$ is consistent, $\Delta_1; \Pi_1$ is consistent. This second transformation introduces no fresh variables. Therefore, we may apply Lemma 2.5 to get $(\Delta \cup \Delta_1; \Pi, \Pi_1 \geq \Delta' \cup \Delta_1; \Pi', \Pi_1)$.

This fact may immediately be used with Lemma 2.1 to prove the second statement in this lemma. \square

LEMMA 2.7 (2.2). *Given a transformation $(\Delta; \Pi \geq \Delta'; \Pi')$ and a substitution $(\sigma : \Delta \cup \Delta' \rightarrow \Delta'')$, the substituted environments also form a transformation $(\sigma\Delta; \sigma\Pi \geq \sigma\Delta'; \sigma\Pi')$. Therefore, given a program g as an expression e that type-checks under TRANSFORM in an environment $E = (\Delta; \Pi)$ ($E \geq E_1 \vdash_\omega e : \tau \dashv E'_1 \geq E'$), and given a substitution $\sigma : \Delta_1 \rightarrow \Delta_2$ where $\Delta_1 \uplus \Delta_2$ is a partition of Δ , then e also type checks in the substituted environment $\sigma E = (\Delta_2; \sigma\Pi)$ ($\sigma E \geq \sigma E_1 \vdash_\omega e : \sigma\tau \dashv \sigma E'_1 \geq \sigma E'$).*

PROOF. Let $E = \Delta; \Pi$ and $E' = \Delta'; \Pi'$. By definition, $\sigma\Delta; \sigma\Pi \geq \sigma\Delta'; \sigma\Pi'$ if and only if

$$\forall \mu; a; \sigma_1; A_{\prec}, A_T (\mu; a \vdash$$

$$\sigma\Delta; \sigma\Pi \text{ consistent using } \sigma_1; (A_{\prec}, A_T)) \Rightarrow$$

$$\exists A'_T (\mu; a \vdash \sigma\Delta'; \sigma\Pi' \text{ consistent using } \sigma_1; (A_{\prec}, A'_T))$$

So let us select arbitrary $\mu; a; \sigma_1; A_{\prec}, A_T$ such that $\mu; a \vdash \sigma E$ consistent using $\sigma_1; (A_{\prec}, A_T)$. Then by definition:

a is acyclic

$$\sigma_1 : \sigma\Delta \rightarrow \emptyset$$

$$\mu; A; \emptyset \vdash \sigma_1\sigma\Pi \Downarrow \hat{\Pi}$$

$$(l : \tau_{\text{atom}}) \in \hat{\Pi} \Rightarrow \vdash \mu(l) : \tau_{\text{atom}}$$

$$(l : \tau \prec l') \in A_{\prec} \Rightarrow (l \prec l') \in a$$

$$t(\nu_1, \dots, \nu_n) \in A_T \Rightarrow A \vdash [r_1 \rightarrow \nu_1, \dots, r_n \rightarrow \nu_n]T(t) =$$

true

If we now define $\sigma_2 = \sigma_1\sigma$, we can conclude that

$$\sigma_2 : \Delta \rightarrow \emptyset$$

$$\mu; A; \emptyset \vdash \sigma_2\Pi \Downarrow \hat{\Pi}$$

And therefore,

$$\mu; a \vdash \Delta; \Pi \text{ consistent using } \sigma_2; (A_{\prec}, A_T)$$

Because $\Delta; \Pi \geq \Delta'; \Pi'$, this in turn implies that

$$\mu; a \vdash \Delta'; \Pi' \text{ consistent using } \sigma_2; (A_{\prec}, A_T)$$

which is to say:

a is acyclic

$$\sigma_2 : \Delta' \rightarrow \emptyset$$

$$\mu; A; \emptyset \vdash \sigma_2\Pi' \Downarrow \hat{\Pi}'$$

$$(l : \tau_{\text{atom}}) \in \hat{\Pi}' \Rightarrow \vdash \mu(l) : \tau_{\text{atom}}$$

$$(l : \tau \prec l') \in A_{\prec} \Rightarrow (l \prec l') \in a$$

$$t(\nu_1, \dots, \nu_n) \in A_T \Rightarrow A \vdash [r_1 \rightarrow \nu_1, \dots, r_n \rightarrow \nu_n]T(t) =$$

true

Substituting again with $\sigma_2 = \sigma_1\sigma$ gives:

$$\sigma_1 : \sigma\Delta' \rightarrow \emptyset$$

$$\mu; A; \emptyset \vdash \sigma_1\sigma\Pi \Downarrow \hat{\Pi}'$$

Therefore, by definition,

$$\exists A'_T (\mu; a \vdash \sigma\Delta'; \sigma\Pi' \text{ consistent using } \sigma_1; (A_{\prec}, A'_T))$$

which is what needed to be proven.

Now let $E = (\Delta; \Pi)$. ($E \geq E_1 \vdash_\omega e : \tau \dashv E'_1 \geq E'$), and $\sigma : \Delta_1 \rightarrow \Delta_2$ where $\Delta_1 \uplus \Delta_2$ is a partition of Δ . As above, $\sigma E \geq \sigma E_1$ and $\sigma E'_1 \geq \sigma E'$. Let $E_1 = (\Delta''; \Pi'')$. Then define $\sigma' : \Delta_1 \cap \Delta'' \rightarrow (\Delta'' - (\Delta_1 \cap \Delta''))$ as $\sigma' = \sigma|_{\Delta_1 \cap \Delta''}$. By Lemma 2.1, $\sigma' E_1 \vdash_\omega e : \tau \dashv \sigma' E'_1$. But $\sigma' E_1 = \sigma E_1$ from its definition, and, as the typing can only introduce fresh variables, $\sigma' E'_1 = \sigma E'_1$. Thus, $\sigma E \geq \sigma E_1 \vdash_\omega e : \tau \dashv \sigma E'_1 \geq \sigma E'$ \square

LEMMA 2.8. *The following rules hold:*

$$E \equiv E \quad \Delta; \Pi \geq \Delta; \emptyset \quad \Delta; \emptyset \equiv \Delta; \text{true}$$

$$\Delta; \Gamma \wedge \Gamma' \equiv \Delta; \Gamma, \Gamma' \quad \Delta; \Gamma \equiv \Delta; \neg\neg\Gamma$$

$$\Delta; t(\rho_1, \dots, \rho_n) \equiv \Delta; [r_1 \rightarrow \rho_1, \dots, r_n \rightarrow \rho_n]T(t)$$

$$(\Delta; \Pi \geq \Delta; \Gamma) \Rightarrow (\Delta; \Pi \equiv \Delta; \Pi, \Gamma) \quad \Delta; \neg\Gamma \geq \Delta; \Gamma \rightarrow \Pi$$

$$\Delta; \Gamma, \Pi \equiv \Delta; \Gamma, \Gamma \rightarrow \Pi \quad \Delta; \Pi \equiv \Delta; \emptyset \rightarrow \Pi$$

$$\Delta; \emptyset \equiv \Delta; B \rightarrow B$$

REDUCE

$$\Delta; B_1 \rightarrow B_2, (B_2, B_3) \rightarrow \Pi_4 \geq \Delta; (B_1, B_3) \rightarrow \Pi_4$$

CARVE-OUT

$$\Gamma = k : \tau \prec k' \wedge k \neq k_1 \wedge \dots \wedge k \neq k_n$$

$$B = \{\{k_1 : \tau_1, \dots, k_n : \tau_n\}\}$$

$$\frac{}{\Delta; k' : \tau' \setminus \{B\}, \Gamma \geq \Delta; k' : \tau \setminus \{k : \tau, B\}, k : \tau}$$

PACK

$$(\Delta; k : \text{ptr}(\rho), [r \rightarrow \rho]\Pi) \geq (\Delta; k : \exists r. \text{ptr}(r) \text{ with } \Pi)$$

UNPACK

r' is fresh

$$\frac{}{(\Delta; k : \exists r. \text{ptr}(r) \text{ with } \Pi) \geq (\{r'\} \cup \Delta; k : \text{ptr}(r'), [r \rightarrow r']\Pi)}$$

PROOF. In the proof, we use $A'_T = A_T, \sigma' = \sigma$ except when otherwise stated.

$E \equiv E$

Trivial.

$\Delta; \Pi \geq \Delta; \emptyset$

This case follows immediately since $\mu; A; \emptyset \vdash \sigma\emptyset \Downarrow \{\}$ is an axiom.

$\Delta; \emptyset \equiv \Delta; \text{true}$

The \leq direction is already established by the previous case. The \geq direction is established by the permission consistency rule for $\Gamma = \text{true}$.

$\Delta; \Gamma \wedge \Gamma' \equiv \Delta; \Gamma, \Gamma'$

This case works because both sides require $A \vdash \Gamma = \text{true}$ and $A \vdash \Gamma' = \text{true}$.

$\Delta; \Gamma \equiv \Delta; \neg\neg\Gamma$

This case works because both sides require $A \vdash \Gamma = \text{true}$.

$\Delta; t(\rho_1, \dots, \rho_n) \equiv \Delta; [r_1 \rightarrow \rho_1, \dots, r_n \rightarrow \rho_n]T(t)$

For the \geq direction: the left side can achieve consistency only if $t(\sigma\rho_1, \dots, \sigma\rho_n) \in A_T$, which requires in turn that $A \vdash [r_1 \rightarrow \sigma\rho_1, \dots, r_n \rightarrow \sigma\rho_n]T(t) = \text{true}$ which (since we assume the correct number of parameters to t) is equivalent to $A \vdash \sigma[r_1 \rightarrow \rho_1, \dots, r_n \rightarrow \rho_n]T(t) = \text{true}$ which is precisely what is needed to prove consistency of the right-hand side.

For the \leq direction, let $A'_T = A_T \cup \{t(\sigma\rho_1, \dots, \sigma\rho_n)\}$, which is no obstacle to consistency because the right-hand side requires this fact. With this addition, proving consistency of the left-hand side is straight-forward.

$(\Delta; \Pi \geq \Delta; \Gamma) \Rightarrow (\Delta; \Pi \equiv \Delta; \Pi, \Gamma)$

The \leq direction of the equivalence to prove follows immediately since removing Γ imposes no obstacle to consistency. For the \geq direction, the antecedent shows that Γ can be found consistent, possibly with a new A_T that we shall call A'_T . Now let $A'_T = A_T \cup A_T^\Gamma$. The new permissions Π, Γ are consistent as before and the expanded A'_T will still be checkable since it is composed of parts that were checkable previously.

$\Delta; \neg\Gamma \geq \Delta; \Gamma \multimap \Pi$

This case follows immediately because the left-hand side requires $A \vdash \Gamma = \text{false}$ which enables the right-hand-side to be proved.

$\Delta; \Gamma, \Pi \equiv \Delta; \Gamma, \Gamma \multimap \Pi$

The \geq direction is similar to the last case, the result follows since the left-hand side requires $A \vdash \Gamma = \text{true}$. The \leq direction is also simple: the right side requires $A \vdash \Gamma = \text{true}$ and thus the proof of $\Gamma \multimap \Pi$ requires that we have Π , and thus the consistency of the left-hand side is easily established.

$\Delta; \Pi \equiv \Delta; \emptyset \multimap \Pi$

This case follows immediately from the consistency axiom for $B_2 \multimap \Pi$ specialized for the case $B_2 = \emptyset$:

$$\frac{\mu; A; B_1 \vdash \Pi \Downarrow \hat{\Pi}}{\mu; A; B_1 \vdash \emptyset \multimap \Pi \Downarrow \hat{\Pi}}$$

$\Delta; \emptyset \equiv \Delta; B \multimap B$

The \leq direction is a special case of the second case. The \geq direction is handled by constructing a proof for consistency with one copy of CP-IMP (with $B_1 = \emptyset, B_2 = \sigma B$) and multiple copies of CP-UNION until we have split the permissions to single keys and apply CP-IDENTITY to cancel each $\sigma\beta$ against itself.

$\Delta; B_1 \multimap B_2, (B_2, B_3) \multimap \Pi_4 \geq \Delta; (B_1, B_3) \multimap \Pi_4$

This rule is a generalization of the linear *modus ponens* rule, which is achieved when $B_1 = B_3 = \emptyset$.

When proving this rule, we may without loss of generality assume all variables have been substituted away ($\Delta = \emptyset$ and σ is the empty substitution) because if there are variables, the consistency rule will immediately substitute them away.

Now let $B_2 = \{\beta_1, \dots, \beta_n\}$, and thus from consistency on the left we have the following facts:

$$\mu; A; B_{1i} \vdash \beta_i \Downarrow \hat{\Pi}_2 \quad B_1 = \sum_{1 \leq i \leq n} B_{1i}$$

$$\mu; A; \beta_1, \dots, \beta_n, B_3 \vdash \Pi_4 \Downarrow \hat{\Pi}_4 \quad \hat{\Pi} = \hat{\Pi}_2 \uplus \hat{\Pi}_4$$

If B_2 is empty ($n = 0$), then so must be B_1 and the result is trivial. The case for $n > 1$ can be handled by repeatedly applying the $n = 1$ case. Thus without loss of generality, we may assume $n = 1$ and thus $B_2 = \{\beta\}$. So to summarize, we have the situation:

$$\mu; A; B_1 \vdash \beta \Downarrow \hat{\Pi}_2 \quad \mu; A; \beta, B_3 \vdash \Pi_4 \Downarrow \hat{\Pi}_4$$

$$\hat{\Pi} = \hat{\Pi}_2 \uplus \hat{\Pi}_4$$

Now we prove by induction over the derivation of the second rule that whenever we have these three rules, we also have

$$\mu; A; (B_1, B_3) \vdash \Pi_4 \Downarrow \hat{\Pi}$$

Proving this “mini-lemma” requires that we examine the following cases for the last step in the derivation of $\mu; A; \beta_1, B_3 \vdash \Pi_4 \Downarrow \hat{\Pi}_4$:

CP-Empty $\Pi_4 = \emptyset$ (Impossible)

CP-True $\Pi_4 = \Gamma$ (Impossible)

CP-FalseImp $\Pi_4 = \Gamma \multimap \Pi'$ where $A \vdash \Gamma = \text{false}$ (Impossible)

CP-ImpTrue $\Pi_4 = \Gamma \multimap \Pi'$ (Follows immediately by induction)

CP-Imp $\Pi_4 = B_3 \multimap \Pi'$ (By induction)

CP-Union $\Pi_4 = \Pi_1, \Pi_2$

Here β must appear on the left-hand side of one of the two facts above the line; we use induction on that one, and the other remains as before, and the desired result follows.

CP-Identity $\beta, B_3 = \Pi_4 = \{\{l : \tau\}\}$

In this case, $\hat{\Pi}_4 = \emptyset$. Also, we must have $\beta = (l : \tau), B_3 = \emptyset$, which means the required fact is already established.

CP-Field Here we have

$$B'_1 = \sum_{(l': \tau' \prec l) \in A_{\prec}} B_{l': \tau'}$$

$$\mu; A; B_{l': \tau'} \vdash l' : \tau' \Downarrow \hat{\Pi}'_{l': \tau'}$$

$$\hat{\Pi}'_1 = \biguplus_{(l': \tau' \prec l) \in A_{\prec}} \hat{\Pi}'_{l': \tau'}$$

$$\frac{\mu; A; B'_2 \vdash \mu(l) : \tau \Downarrow \hat{\Pi}'_2 : \tau_{\text{atom}}}{\mu; A; B'_1, B'_2 \vdash l : \tau \Downarrow \hat{\Pi}'_1 \uplus \hat{\Pi}'_2 \uplus \{l : \tau_{\text{atom}}\}}$$

we have two cases, either $\beta \in B'_1$ or $\beta \in B'_2$ (or both). In the former case, that means that $\beta \in$

$B_{l':\tau'}$ for some $l' : \tau' \prec l \in A_{\prec}$. Then since $\hat{\Pi}_1$ is disjoint with $\hat{\Pi}_4 = \hat{\Pi}'_1 \uplus \hat{\Pi}'_2 \uplus \{l : \tau_{\text{atom}}\}$ then it is certainly disjoint with all the $\hat{\Pi}'_{l':\tau'}$ that partition $\hat{\Pi}'_1$. Then by induction for the $l' : \tau' \prec l$ case, and the associativity of \uplus , we achieve the desired result.

If on the other hand $\beta \in B'_2$, then if τ is atomic, we are done since $\hat{\Pi}'_2 = \{\}$. Otherwise if τ is an existential, then we can use induction on the permissions packed into the existential and again achieve our result.

Thus with this mini-lemma (which is also used in the following case), we can now state

$$\mu; A; \emptyset \vdash (B_1, B_3) \multimap \Pi_4 \Downarrow \hat{\Pi} \quad .$$

which enables us to prove consistency of the right-hand side.

Carve-Out As before, we ignore variables. Now the expansion of the $\cdot : \cdot \setminus \{\dots\}$ adds adoption facts to a linear implication.

Thus $k' : \tau' \setminus \{k_1 : \tau_1, \dots, k_n : \tau_n\}, k : \tau \prec k', k \neq k_1, \dots, k \neq k_n$ where each k has no variables (is a l) in a consistent state $\mu; a$ using (A_{\prec}, A_T) requires

$$\begin{aligned} & \mu; A; l_1 : \tau_1, \dots, l_n : \tau_n \vdash k' : \tau' \Downarrow \hat{\Pi} \\ (l_i : \tau_i \prec l') \in A_{\prec} \quad & (l : \tau \prec l') \in A_{\prec} \quad l \neq l_i \end{aligned}$$

Now let $\{l'' : \tau'' \prec l \in A_{\prec}\}$ be enumerated in the following order for simplicity (where $(l_{n+1} : \tau_{n+1}) = (l : \tau)$):

$$\{l_1 : \tau_1, \dots, l_n : \tau_n, l_{n+1} : \tau_{n+1}, \dots, l_m : \tau_m\}$$

Now since adoption is acyclic $l' \neq l_i$ for any $1 \leq i \leq m$, and so the last consistency rule for permissions must be the one to prove the consistency of $k' : \tau$, which means we have the following facts:

$$\begin{aligned} & \mu; A; B_i \vdash l_i : \tau_i \Downarrow \hat{\Pi}_i \\ (l_1 : \tau_1, \dots, l_n : \tau_n) = B_0 + \sum_{1 \leq i \leq m} B_i \\ & \mu; A; B_0 \vdash \mu(l') : \tau' \Downarrow \hat{\Pi}_0 : \tau'_{\text{atom}} \\ \hat{\Pi} = \{l' : \tau'\} \uplus \hat{\Pi}_0 \uplus \biguplus_{1 \leq i \leq m} \hat{\Pi}_i \end{aligned}$$

Of interest here is B_{n+1} . If it is empty, we can create $B'_i = B_i$ except for $B'_{n+1} = \{l : \tau\}$. Now we can easily prove $\mu; A; l : \tau \vdash l : \tau \Downarrow \{\} = \hat{\Pi}'_i$, and keeping all

other $\hat{\Pi}'_i = \hat{\Pi}_i$, we can prove

$$\begin{array}{c} \mu; A; B'_i \vdash l_i : \tau_i \Downarrow \hat{\Pi}'_i \\ (l_1 : \tau_1, \dots, l_n : \tau_n, l_{n+1} : \tau_{n+1}) = B_0 + \sum_{1 \leq i \leq m} B'_i \\ \mu; A; B_0 \vdash \mu(l') : \tau' \Downarrow \hat{\Pi}_0 : \tau'_{\text{atom}} \\ \hat{\Pi}' = \{l' : \tau'\} \uplus \hat{\Pi}_0 \uplus \biguplus_{1 \leq i \leq m} \hat{\Pi}'_i \\ \hline \mu; A; l : \tau, l_1 : \tau_1, \dots, l_n : \tau_n \vdash l' : \tau' \Downarrow \hat{\Pi}' \\ \hline \mu; A \vdash (l : \tau, l_1 : \tau_1, \dots, l_n : \tau_n) \multimap l' : \tau' \Downarrow \hat{\Pi}' \\ \hline \vdots \\ \hline \mu; A; l_i : \tau_i \Downarrow \hat{\Pi}_i \\ \hline \mu; A; (l : \tau, l_1 : \tau_1, \dots, l_n : \tau_n) \multimap l' : \tau', l : \tau \Downarrow \hat{\Pi} = \hat{\Pi} \end{array}$$

which permits us to prove consistency of the right-hand side.

But suppose B_{n+1} is not empty: $B_{n+1} = l_j : \tau_j, B''$. In that case, we use the mini-lemma from the previous case to convert

$$\mu; A; B_j \vdash l_j : \tau_j \Downarrow \hat{\Pi}_j$$

$\mu; A; l_j : \tau_j, B'_{n+1} \vdash l : \tau \Downarrow \hat{\Pi}_{n+1} \quad \hat{\Pi}'_{n+1} = \hat{\Pi}_j \uplus \hat{\Pi}_{n+1}$
into

$$\mu; A; B_j, B'' \vdash l : \tau \Downarrow \hat{\Pi}'_{n+1}$$

to which we add the axiom

$$\mu; A; l_j : \tau_j \vdash l_j : \tau_j \Downarrow \{\} = \hat{\Pi}'_j$$

and produce a new proof of our original fact using $B'_{n+1} = B_j, B'', B'_j = \{\{l_j : \tau_j\}\}$, $\hat{\Pi}'_j = \{\}$ and all the other $B'_i = B_i$, $\hat{\Pi}'_i = \hat{\Pi}_i$. If this new B'_{n+1} is still not empty we can repeat this process. Eventually because of linearity (we have a partition) and finiteness, we end up with a B^*_{n+1} which is empty and can proceed.

Pack $(\Delta; k : \text{ptr}(\rho), [r \rightarrow \rho]\Pi) \geq (\Delta; k : \exists r. \text{ptr}(r)$ with $\Pi)$

Let $l = \sigma k$. Associativity of *uplus* and consistency on the left requires that we have the following facts:

$$\begin{aligned} & \mu; A; \emptyset \vdash l' : \tau' \Downarrow \hat{\Pi}_{l':\tau'} \quad \hat{\Pi}_1 = \biguplus_{l':\tau' \prec l} \hat{\Pi}_{l':\tau'} \\ & \mu; A; \emptyset \vdash \mu(l) : \text{ptr}(\sigma\rho) \Downarrow \{\} : \text{ptr}(\sigma\rho) \\ & \mu; A; \emptyset \vdash \sigma[r \rightarrow \rho]\Pi \Downarrow \hat{\Pi}_3 \\ & \hat{\Pi} = \hat{\Pi}_1 \uplus \{l : \text{ptr}(\sigma\rho)\} \uplus \hat{\Pi}_3 \end{aligned}$$

Furthermore, this element of the flattened permissions $l : \text{ptr}(\sigma\rho)$ ensures that $\mu(l) = \sigma\rho$, some object reference, we call ν .

Now since r is fresh, $\sigma[r \rightarrow \rho]\Pi = [r \rightarrow \sigma\rho]\sigma_r\Pi = [r \rightarrow \nu]\sigma_r\Pi$, where $\sigma_r r = r, r' \neq r \Rightarrow \sigma_r r' = \sigma_r'$, and thus

$$\begin{array}{c} \vdots \\ \hline \mu; A; \emptyset \vdash [r \rightarrow \nu]\sigma_r\Pi \Downarrow \hat{\Pi}_3 \\ \hline \mu; A; \emptyset \vdash \nu : \sigma(\exists r. \text{ptr}(\rho)) \text{ with } \Pi \Downarrow \hat{\Pi}_3 \end{array}$$

which permits us to prove the consistency of the right-hand side.

Unpack

$$\frac{r' \text{ is fresh}}{(\Delta; k : \exists r. \text{ptr}(r) \text{ with } \Pi) \geq (\{r'\} \cup \Delta; k : \text{ptr}(r'), [r \rightarrow r']\Pi)}$$

Let $l = \sigma k$. For consistency on the left we must have used CP=FIELD and thus have the following facts:

$$\begin{aligned} \mu; A; \emptyset \vdash l' : \tau' \Downarrow \hat{\Pi}_{l':\tau'} \\ \hat{\Pi}_1 = \bigsqcup_{l':\tau' \prec l} \hat{\Pi}_{l':\tau'} \\ \mu; A; \emptyset \vdash \mu(l) : \sigma(\exists r. \text{ptr}(r) \text{ with } \Pi) \Downarrow \hat{\Pi}_2 : \text{ptr}(\sigma\rho) \\ \hat{\Pi} = \hat{\Pi}_1 \uplus \hat{\Pi}_2 \uplus \{l : \text{ptr}(\sigma\rho)\} \end{aligned}$$

Furthermore, this element of the flattened permissions $l : \text{ptr}(\sigma\rho)$ ensures that $\mu(l) = \sigma\rho$, and both equal some object reference, ν .

Let $\sigma' = \sigma[r' \rightarrow \nu]$. Trivially $\sigma' \supseteq \sigma$. Then, $\mu; A; \emptyset \vdash \mu(\sigma'k) : \text{ptr}(\sigma'r') \Downarrow \{\} : \text{ptr}(\nu)$. We can use this with the first two facts above to get:

$$\mu; A; \emptyset \vdash \sigma'(k : \text{ptr}(r')) \Downarrow \hat{\Pi}_1 \uplus \{l : \text{ptr}(\sigma'\rho)\} \quad (*)$$

(As σ and σ' differ only on r' (fresh) $\sigma\rho = \sigma'\rho$.)

The third fact, in turn, requires that

$$\mu; A; \emptyset \vdash [r \rightarrow \nu]\sigma\Pi \Downarrow \hat{\Pi}_2$$

But this is identical to

$$\mu; A; \emptyset \vdash \sigma'([r \rightarrow r']\Pi) \Downarrow \hat{\Pi}_2 \quad (*')$$

The facts $*$ and $*'$ can be combined using CP-UNION to get:

$$\begin{aligned} \mu; A; \emptyset \vdash \sigma'(k : \text{ptr}(r'), [r \rightarrow r']\Pi) \Downarrow \\ \hat{\Pi}_1 \uplus \hat{\Pi}_2 \uplus \{l : \text{ptr}(\sigma'\rho)\} \end{aligned}$$

The union of disjoint sets is well-defined as it duplicates that used to define $\hat{\Pi}$ above. \square

LEMMA 2.9. *Given a type-checked program g ($\vdash g : \omega$), an expression e that type-checks in a variable-free environment $E = (\emptyset; \Pi)$ ($E \vdash_\omega e : \tau \dashv E'', E'' = (\Delta''; \Pi'')$, we do not require $\Delta'' = \emptyset$) and a memory and adoption information consistent with E ($\mu; a \vdash E$ consistent), then either e is a value or there exists an evaluation step $(\mu; a; e) \rightarrow_g (\mu'; a'; e')$ and for any such evaluation step, there exists a substitution (with absolute addresses) on some of the new type variables $\sigma : \Delta \rightarrow \emptyset$ where $\Delta \subseteq \Delta''$ such that the resulting expression type-checks in a new environment $E' = (\emptyset; \Pi')$ with the substituted result ($E' \vdash_\omega e' : \sigma\tau \dashv \Delta'' - \Delta; \sigma\Pi''$). In addition, the witness A' for the new consistency will include everything in the witness of the original consistency A ($A_\prec \subseteq A'_\prec, A_T \subseteq A'_T$), and furthermore the new flattened permissions will only include locations from the old permissions or newly allocated memory: $\text{Dom}(\hat{\Pi}') \cap \text{Dom}(\mu) \subseteq \text{Dom}(\hat{\Pi})$.*

PROOF. Let σ and A be the (original) witnesses of consistency. We prove the result by induction over the typing derivation. with the following case analysis:

Unit, Num, True, False, Address

For all these cases, e is a value, and thus the result is vacuously satisfied.

Plus $e_1 + e_2$

If both e_1 and e_2 are values, then they must be integer constants and evaluation to another integer constant is immediate. The memory hasn't changed and typing is assured and thus we choose the same environment $E' = E$, an empty substitution $\sigma = \square$ and use the same witnesses for consistency.

If e_1 is a value but e_2 is not, then we get an evaluation and the new environment and witnesses by induction. Since e_1 is a value, its typing is assured and we can form a typing of all of e' with the new environment.

If e_1 is not a value, then by induction, we get a new environment E' and substitution σ with witnesses for the consistency of the memory after the evaluation of e_1 . Thus we have evaluation of e and using the substitution lemma can type e_2 in the (substituted) output environment from e' .

Equal $e_1 = e_2$

Analogous to the case for PLUS. Here we use the fact that substitutions on atomic types always yield atomic types, in fact types that are storage compatible, thus enabling us to preserve the typing $\tau_1 \sim \tau_2$.

Read $e_1 . f$

If e_1 is a value, then it must be an object ν , and $\tau_1 = \text{ptr}(\nu)$. We must have therefore $\Pi = \nu.f : \tau \setminus \{B\}, \Pi_1$ where τ is an atomic type. By consistency, we know that $\mu; A; B \vdash \nu.f : \tau \Downarrow \hat{\Pi}_1$ where $\hat{\Pi}_1 \subseteq \hat{\Pi}$ used check μ for consistency, and where every $\beta \in B$, we have $(\beta_i \prec \nu.f) \in A_\prec$ and thus for every $\beta_i = l_i : \tau_i$ we have $(l_i \prec \nu.f) \in a$. By the acyclicity of a , this means $l_i \neq \nu.f$, and thus the only rule for consistency is the final one, which (since τ is already atomic) requires that $\hat{\Pi}_1 \ni \nu.f : \tau$. Thus the rule for consistency requires that $\vdash \mu(\nu.f) : \tau$. Given these facts, we can determine that evaluation to $(\mu; a; e')$ is assured where $e' = \mu(\nu.f)$. Now let $E' = E, \sigma = \square$, and so $E \vdash_\omega e' : \tau \dashv E''$ is true. Since we have $\mu' = \mu, a' = a$, consistency is checking the same values.

Otherwise if e_1 is not a value, then the result follows by induction.

New $\text{new}\{f_i \mid 1 \leq i \leq n\}$

In this case, we have $\tau = \text{ptr}(r)$ for a fresh variable r . Since we assume μ is finite while the set of addresses is infinite, evaluation is assured to a new state $(\mu'; a'; e')$ where $a' = a, e' = \nu$. Now let $\sigma = [r \rightarrow \nu]$ be a substitution that replaces the fresh variable with ν , and thus $\sigma\tau = \text{ptr}(\nu)$. Now let $E' = \sigma E''$; it is clear that $E' \vdash_\omega e' : \tau' \dashv E'$, and thus all we need to prove is the preservation of consistency. The new set of permissions Π' has additions for the newly allocated fields: $\{\nu.f_1 : \tau_{f_1}, \dots, \nu.f_n : \tau_{f_n}\}$. Now the evaluation ensures us that $\nu.f_i \notin \text{Rng}(a = a')$, and thus

$\mu'; a'; \emptyset \vdash \nu.f_i : \tau_{f_i} \Downarrow \{\nu.f_i : \tau_{f_i}\}$. Next we check that these flattened sets are disjoint with the flattened permissions from the original Π . Since these permissions were consistent with μ , and because that means μ must be defined on the locations in these flattened sets, and because the evaluation rule ensures that $\mu\nu.f$ was not defined for any f , we get the desired result. We note that this meets the requirements on the domain of the new flattened permissions. Finally, we need to check the full flattened permission set against μ' . It is only changed for the new locations, so it remains consistent with the original flattened permissions, and the changes of evaluation gives it exactly the correct values that correspond to the fields' initial types and thus we are done.

Write $e_1.f := e_2$

For this type rule, the type of the pointer e_1 is $\text{ptr}(\rho)$ and the type τ of new value is storage compatible with τ' the type recorded for the field in the permission. If e_1 and e_2 are both values, then e_1 must be an object reference ν . As with the rule for **READ**, consistency requires that $\vdash \mu(\nu.f) : \tau'$ and τ' is atomic. We also have that $\vdash e_2 : \tau$, the value is of the required (atomic) type. Since the type rule requires $\tau \sim \tau_f$, we have $e_2 \sim \nu.f$ and thus we have progress to $(\mu; a; ())$. Let $E' = E''$, and thus $E' \vdash_{\omega} () : \text{unit} \dashv E''$ follows immediately. Now, because of the union of disjoint sets computing $\hat{\Pi}$, Π_1 must not flatten to include a requirement that $\mu(o.f)$ has type τ' , and thus changing the type of this field will not cause problems, and indeed the new type now matches what is in the store, and so we have consistency. The domain of the flattened permissions remains the same as before.

If e_1 is a value, but not e_2 , the desired result follows immediately by induction. If e_1 is not a value, again we use induction, and then also the substitution lemma to achieve the required result.

Seq $e_1; e_2$

If e_1 is a value, then since it must have unit type, it can only be $()$ and thus we have one step of evaluation to $(\mu; a; e_2)$. The type rule permits us to write $E \vdash_{\omega} e_1 : \text{unit} \dashv E_1 \vdash_{\omega} e_2 : \tau \dashv E_2$ and since $e_1 = ()$, we have $E_1 = E$. Let $E' = E = E_1$ and then typing is preserved trivially. Consistency is also preserved since no part of the relation is affected by evaluation.

If e_1 is not a value, then the result follows through application of induction and the substitution lemma.

If **if** e_0 **then** e_1 **else** e_2

If e_0 is a value, then since the type is boolean, it must be **true** or **false**. In the former case, we have evaluation immediately to $(\mu; a; e_2)$. Now we know that $E \vdash_{\omega} e_0 : \text{bool} \dashv E' \vdash_{\omega} e_1 : \text{unit} \dashv E_1$ and $E' = E$, and also that $E'' = E_1 \vee E_2$, which means $E_1 = \sigma_1 E''$, and thus the substitution we use is $\sigma = \sigma_1$. Thus we have the typing result needed, and since memory and environment are unchanged, consistency is also as needed. The case for $e_0 = \text{false}$ is analogous.

Now if e_0 is not a value, we can use induction and the substitution lemma (including the part that says

that substitution carries over \vee) to achieve the desired result.

IfEqual **if** $e_0 = e_1$ **then** e_2 **else** e_3

If both e_0 and e_1 are values, then given their types, they must be ν_1 and ν_2 respectively, and their types must be $\text{ptr}(\nu_1)$ and $\text{ptr}(\nu_2)$. Now the type rule ensures that e_2 type checks in an environment in which the equality is assumed and e_3 in the environment where they are assumed not equal:

$$\Delta; \nu_1 = \nu_2, \Pi \vdash_{\omega} e_2 : \text{unit} \dashv E_2 \Delta; \nu_1 \neq \nu_2, \Pi \vdash_{\omega} e_3 :$$

$$\text{unit} \dashv E_3 E'' = E_2 \vee E_3$$

If the objects are indeed equal, we have immediate progress to **if true then** e_2 **else** e_3 . Furthermore, (it can be easily shown), the equality fact adds no information and can be dropped, and thus we have $E \vdash_{\omega} e_2 : \text{unit} \dashv E_2$. By the definition of \vee , we have $E_2 = \sigma_2 E''$ and thus $E \vdash_{\omega} e_2 : \text{unit} \dashv \sigma E''$ which is precisely what we need to use **IFTRUE** to type-check the new program. Since the store and environment are unchanged, consistency also follows. Thus we have achieved the desired result. The case for inequality is completely analogous.

If either e_0 or e_1 is not a value, then we can use the inductive hypothesis and straightforward reasoning afterwards.

IfTrue **if** **true** **then** e_1 **else** e_2

Trivial.

IfFalse **if** **false** **then** e_1 **else** e_2

Trivial.

Call **call** p

The type rule, **PROC** and **PROGRAM**, ensure we have the following facts:

$$E = \emptyset; \sigma_1 \Pi_1, \Pi_3$$

$$E'' = \Delta_+; \sigma_1 \Pi_2, \Pi_3$$

$$\omega(p) = \forall \Delta_1. \Pi_1 \rightarrow \exists \Delta_2. \sigma_2 \Pi_2$$

$$\sigma_1 : \Delta_1 \rightarrow \emptyset$$

$$\Delta_+ \text{ fresh}$$

$$\sigma_2 : \Delta_+ \rightarrow \Delta_2$$

$$\Delta_1; \Pi_1 \vdash_{\omega} g(p) : \text{unit} \dashv \Delta'_1; \sigma_3 \sigma_2 \Pi_2$$

$$\Delta'_1 \cap \Delta_2 = \emptyset$$

$$\sigma_3 : \Delta_2 \rightarrow \Delta'_1$$

In particular, we know that $g(p)$ is defined and thus we have immediate progress to $(\mu; a; g(p))$. By application of the substitution lemma (for σ_1) and the widening lemma (adding Π_3 to both sides), we achieve

$$E \vdash_{\omega} g(p) : \text{unit} \dashv \sigma_1 \Delta'_1; \sigma_1 \sigma_3 \sigma_2 \Pi_2, \Pi_3$$

Since σ_1 is idempotent and neither σ_2 nor σ_3 have any effect on variables in Δ_1 (Δ_1 must be disjoint with

both Δ_2 and with fresh variables Δ_+ , $\sigma_1\sigma_3\sigma_2\Pi_2 = \sigma_1\sigma_3\sigma_2(\sigma_1\Pi_2)$. Let $\sigma = \sigma_1\sigma_3\sigma_2$. Now $\sigma\Delta_+ = \sigma_1\sigma_3\sigma_2\Delta_+ = \sigma_1\sigma_3\Delta_2 = \sigma_1\Delta'_1$ and thus since Π_3 has no free variables, the previous typing can be written $E \vdash_\omega g(p) : \text{unit} \dashv \sigma E''$ which is the required type preservation. Consistency is preserved trivially since the input environment $E' = E$, memory and adoption are all unchanged.

Nest nest $e_0.f_0$ in $e_1.f_1$

If both e_0 and e_1 are values, then the typing rules require (using the unstated canonical forms lemma) that they both be objects ν and ν' respectively, and thus we have immediate progress to $(\mu; a'; \emptyset)$ where $a' = a \cup \{(o, f) \prec (o', f')\}$. Typing is preserved too using $E' = E''$ and UNIT: $E' \vdash_\omega () : \text{unit} \dashv E''$. Preservation of consistency is more interesting. The starting permissions are

$$\Pi = (l : \tau, l \neq l_1 \wedge \dots \wedge l \neq l_n), \\ l' : \tau' \setminus \{l_1 : \tau_1, \dots, l_n : \tau_n\}, \Pi_1$$

Consistency in the original state gives inequalities (which we do not need for this lemma) and also the following (we may need to use the associativity and commutativity of $+$ and \uplus to rearrange the consistency proof to have this shape):

$$\mu; A; \emptyset \vdash l : \tau \Downarrow \hat{\Pi}_2$$

$$\mu; A; l_1 : \tau_1, \dots, l_n : \tau_n \vdash l' : \tau' \Downarrow \hat{\Pi}_3$$

$$\mu; A; \emptyset \vdash \Pi_1 \Downarrow \hat{\Pi}_1 \quad \hat{\Pi} = \hat{\Pi}_1 \uplus \hat{\Pi}_2 \uplus \hat{\Pi}_3$$

Furthermore, we have $a \ni (l_i \prec l)$ for all $1 \leq i \leq n$. Since the adoption relation cannot be cyclic the consistency rule for the second consistency fact above cannot be the self-canceling one, and must instead use adoption. Now if $l' \prec^* l$ using our original adoption relation, then the consistency proof for the first consistency fact above would include a subrule $\mu; A; \emptyset \vdash l' : \tau'' \Downarrow \hat{\Pi}_n$ but then $\hat{\Pi}_n$ would include $l' : \tau''_{\text{atom}}$ whereas $\hat{\Pi}_3 \ni l' : \tau''_{\text{atom}}$. If these atomic types are the same, then the union of disjoint sets operator will fail at some point. If they are different, then $\mu(l)$ will be required to match two different atomic types, which is not possible. Thus the new addition to a cannot cause a cycle. Now define $A'_{\prec} = A_{\prec} \cup \{l : \tau \prec l'\}$ and thus matches a' as required for consistency. It remains only to show that we can form the new set of flattened permissions and that this set has no more requirements upon μ than did the previous set.

We consider two cases: whether or not the location was already adopted with this type: $l : \tau \prec l' \in A_{\prec}$. If it was already adopted, we have $A'_{\prec} = A_{\prec}$, $a' = a$ and thus there is no change in the consistency proof and thus we have $\mu; A'; l_1 : \tau_1, \dots, l_n : \tau_n \vdash l' : \tau' \Downarrow \hat{\Pi}_3$ and thus, the consistency proof simply produces a smaller set than it did before, and thus consistency is preserved.

Otherwise, first we note that $\mu; A; \emptyset \vdash \Pi_1 \Downarrow \hat{\Pi}_1$ cannot depend on the absence of the new adoption fact. This is due to the ways in which adoption facts are used. In particular, a fact permission $\pi = \Gamma$ can never depend

on the *absence* of an adoption fact. Similarly for the case $\pi = \Gamma \dashv \Pi$. And the only other place adoption is used is when we use the consistency rule CP-FIELD. If the consistency proof for Π_1 uses this rule and depends on the presence or absence of $l : \tau \prec l'$ then the resulting flattened permission set must include an element of the form $l' : \tau''_{\text{atom}}$ which would have caused consistency problems as described previously. Thus we conclude $\mu; A'; \emptyset \vdash \Pi_1 \Downarrow \hat{\Pi}_1$.

Next, the new application of CP-FIELD for $l' : \tau'$ is completed by adding the subgoal $\mu; A'; \emptyset \vdash l : \tau \Downarrow \hat{\Pi}_2$, which means that we have:

$$\mu; A'; l_1 : \tau_1, \dots, l_n : \tau_n \vdash l' : \tau' \Downarrow \hat{\Pi}_3 \uplus \hat{\Pi}_2$$

$$\mu; A'; \emptyset \vdash \Pi_1 \Downarrow \hat{\Pi}_1 \quad \hat{\Pi}' = \hat{\Pi}_1 \uplus \hat{\Pi}_3 \uplus \hat{\Pi}_2$$

This set is the same as before and thus consistency is preserved.

Transform $E \geq E_1 \vdash_\omega e : \tau \dashv E_2 \geq E''$

If e is a value, we are done. Otherwise, we have progress by induction $(\mu; a; e) \rightarrow_g (\mu'; a'; e')$, and also have E'_1 and σ where $E'_1 \vdash_\omega e' : \tau \dashv \sigma E_2$ and $\mu'; a' \vdash E'_1$ consistent. Now by the substitution lemma, $\sigma E_2 \geq \sigma E''$ and thus we can apply TRANSFORM again to achieve:

$$E'_1 \geq E'_1 \vdash_\omega e' : \tau \dashv \sigma E_2 \geq \sigma E''$$

which is the required result. The set of flattened permissions is always a subset of the original ones (Lemma 2.4) and thus the domain requirement is met. \square

LEMMA 2.10. *Given a type-checked program g ($\vdash g : \omega$), an expression e that type-checks in a variable-free environment $\emptyset; \Pi \vdash_\omega e : \tau \dashv E''$ and a memory and adoption consistent in an environment with more permissions ($\mu; a \vdash \emptyset; \Pi, \Pi_+$ consistent), then the evaluation of the expression (if any) will not read or write any field mentioned in the flattening of the extra permissions ($\hat{\Pi}_+$).*

PROOF. First we note that consistency requires first that $\text{Dom}(\hat{\Pi}) \cup \text{Dom}(\hat{\Pi}_+) \subseteq \text{Dom}(\mu)$, and second that the two sets of (flattened) permissions refer to different segments of memory: $\text{Dom}(\hat{\Pi}) \cap \text{Dom}(\hat{\Pi}_+) = \emptyset$. Otherwise, either they would have overlap (and have the union of disjoint sets be undefined) or they would have conflicting requirements on the store which would make consistency impossible.

Now the statement is trivial if evaluation is already complete (e is a value). Otherwise, we proceed with induction over the length of the evaluation sequence, by first showing the result for one step of evaluation and then showing that we can re-establish the conditions of the lemma.

First, however, we observe that if if we type a value v ($E \vdash_\omega v : \tau \dashv E'$) then the resulting flattened set of permissions for any consistent memory will be a (possible improper) subset of the original flattened permissions $\hat{\Pi}' \subseteq \hat{\Pi}$. This follows because the five rules specifically for values UNIT/NUM/TRUE/FALSE/ADDRESS make no change in the environment at all, and the only other one TRANSFORM (which may be applied an arbitrary number of times) can only produce a subset of the flattened permissions (by Lemma 2.4).

If evaluation proceeds for one step $Eval\mu; ae\mu'; a'e'$ let $\hat{\Pi}$ be the flattened permissions before evaluation. We make a simple proof by induction over the typing of e that any location l read or written in the evaluation will be present in the flattened permissions: $l \in \text{Dom}(\hat{\Pi})$:

Unit/Num/True/False/Address No evaluation possible.

Plus e_1+e_2

If e_1 is not a value, then the next step of evaluation will evaluate e_1 . The environment used in the typing of e_1 is same as the environment for e and thus the result follows immediately by induction. Otherwise, if it is a value, but e_2 is not, then we see that the flattened permissions for the environment used to check e_2 are a subset, and thus the result follows by induction. If both e_1 and e_2 are values, then they must be integer constants, and thus the evaluation does not access memory at all.

Equal $e_1=e_2$

Analogous to PLUS

Read $e_1.f$

If e_1 is not a value, then the result follows by induction (there is no change in environment). Otherwise, e_1 must be a value of type $\text{ptr}(\nu)$ (given that the environment is empty). Now E_1 (the environment after typing the object) must either be the same as E , or must be a transformed version of E . In either case, it must have an empty Δ , and its permissions Π_1 must include $\nu.f : \tau \setminus \{\dots\}$. Using a similar process as was used for the proof of progress, we can show that the flattened permissions $\hat{\Pi}_1$ must include $\nu.f : \tau$. Thus it must be in the flattened permissions for the original environment $\hat{\Pi}$.

New $\text{new } \{ \dots \}$

This construct does not access any existing memory locations when evaluated.

Write $e_1.f:=e_2$

Analogous to READ

Seq $e_1;e_2$

If e_1 is a value, then it must be $()$, and thus evaluation goes forward without any read or write effects. Otherwise, the result follows by induction.

If/IfEqual $\text{if } e_0 \text{ then } e_1 \text{ else } e_2$

If e_0 is a value, then it must be a boolean constant and evaluation proceeds without affecting memory. Otherwise the result holds by induction.

IfTrue/IfFalse/Call Evaluation doesn't use memory.

Nest $\text{nest } e_0.f_0 \text{ in } e_1.f_1$

If e_0 is not a value, the result follows by induction. Otherwise, it e_1 is not a value, the result follows by induction once we take into account possible application of TRANSFORM, which as we have seen allow us to achieve our result still.

The interesting case is the one in which both are values. Now, the permission required (possibly after transformation that only reduce the flattened permissions) ensures that the flattened permissions will include the two locations of the nesting expression, and thus the lemma is proved.

Transform $E \geq E_1 \vdash_\omega e : \tau \dashv E_2 \geq E''$

By induction the result is true for E_1 and then because of Lemma 2.4, must be true for E as well.

Now we use the preservation aspect of Lemma 2.9 to get $\emptyset; \Pi' \vdash_\omega e' : \tau \dashv \sigma E''$, where $\text{Dom}(\hat{\Pi}') \cap \text{Dom}(\mu) \subseteq \text{Dom}(\hat{\Pi})$. Now since $\text{Dom}(\hat{\Pi}_+) \subseteq \text{Dom}(\mu)$ and as shown above $\text{Dom}(\hat{\Pi}) \uplus \text{Dom}(\hat{\Pi}_+) \subseteq \text{Dom}(\mu)$. As a result, we get that $\hat{\Pi}' \uplus \hat{\Pi}_+$ is well defined. Finally this union of disjoint sets is consistent with μ' because the first part is already consistent (by preservation), and the second part (which was consistent with μ) only puts requirements on locations that are not allowed to be modified in evaluation (by this lemma). Furthermore, evaluation only adds things to a , never removes them, and because consistency of permissions never depends on the absence of adoption information, we have $\mu'; a' \vdash \Pi', \Pi_+$ consistent, which preserves the conditions of this lemma for another evaluation. \square

THEOREM 2.11. *Given a type-checked program g ($\vdash g : \omega$), two expressions e_1, e_2 each of which type-checks in a variable-free environment $\emptyset; \Pi_i \vdash_\omega e_i : \text{unit} \dashv E'_i$) and a memory and adoption consistent with the combined environments: $(\mu; a \vdash \emptyset; \Pi_1, \Pi_2 \text{ consistent})$, then when evaluating the expressions in sequence $e_1; e_2$ no state will be accessed by both expressions.*

PROOF. Preservation and widening ensures that when we finish evaluating e_1 , we have a substitution $\sigma_1 : \Delta'_1 \rightarrow \emptyset$ for which $\emptyset; \sigma_1 \Pi'_1, \Pi_2 \vdash_\omega e_2 : \text{unit} \dashv \Delta'_2; \sigma_1 \Pi'_1, \Pi_2$. Then we apply Lemma 2.10 to ensure that evaluating e_2 does not access anything in $\hat{\Pi}'_1$ (the flattening of $\sigma \Pi'_1$). Furthermore, this set does not include any values for locations in $\text{Dom}(\hat{\Pi}_2)$. On the other hand, if we follow preservation step by step, we see that the evaluation e_1 only accessed locations in $\text{Dom}(\hat{\Pi}'_1)$. Thus the two expressions are separate. \square