

$$\begin{aligned}
& \llbracket \text{skip} \rrbracket_{\text{ok}} \triangleq \{(s, s) \mid s \in \text{World}\} & \llbracket \text{skip} \rrbracket_{\text{er}} \triangleq \emptyset \\
& \llbracket x := e \rrbracket_{\text{ok}} \triangleq \{((\eta, \sigma), (\eta[x \mapsto v], \sigma)) \mid \llbracket e \rrbracket \eta = v\} & \llbracket x := e \rrbracket_{\text{er}} \triangleq \emptyset \\
& \llbracket \text{assume}(B) \rrbracket_{\text{ok}} \triangleq \{(s, s) \mid s=(\eta, \sigma) \wedge \llbracket B \rrbracket \eta \neq 0\} & \llbracket \text{assume}(B) \rrbracket_{\text{er}} \triangleq \emptyset \\
& \llbracket \text{error}() \rrbracket_{\text{ok}} \triangleq \emptyset & \llbracket \text{error}() \rrbracket_{\text{er}} \triangleq \{(s, s) \mid s \in \text{World}\} \\
& \llbracket C_1; C_2 \rrbracket_{\epsilon} \triangleq \left\{ (s, s') \mid \begin{array}{l} \epsilon = \text{er} \wedge (s, s') \in \llbracket C_1 \rrbracket_{\epsilon} \\ \vee \exists s''. (s, s'') \in \llbracket C_1 \rrbracket_{\text{ok}} \wedge (s'', s') \in \llbracket C_2 \rrbracket_{\epsilon} \end{array} \right\} \\
& \llbracket C_1 + C_2 \rrbracket_{\epsilon} \triangleq \llbracket C_1 \rrbracket_{\epsilon} \cup \llbracket C_2 \rrbracket_{\epsilon} \\
& \llbracket C^* \rrbracket_{\epsilon} \triangleq \bigcup_{i \in \mathbb{N}} \llbracket C^i \rrbracket_{\epsilon} \quad \text{with } C^0 \triangleq \text{skip} \quad \text{and } C^{i+1} \triangleq C; C^i \\
& \llbracket x := \text{malloc}() \rrbracket_{\text{ok}} \triangleq \left\{ (s, (\eta[x \mapsto l], \sigma[l \mapsto v])) \mid \begin{array}{l} s=(\eta, \sigma) \wedge v \in \text{Val} \\ \wedge (l \notin \text{dom}(\sigma) \vee \sigma(l)=\perp) \end{array} \right\} \\
& \quad \cup \left\{ (s, (\eta[x \mapsto \text{nil}], \sigma)) \mid s=(\eta, \sigma) \right\} \\
& \llbracket x := \text{malloc}() \rrbracket_{\text{er}} \triangleq \emptyset \\
& \llbracket \text{free}(x) \rrbracket_{\text{ok}} \triangleq \left\{ (s, (\eta, \sigma[\eta(x) \mapsto \perp])) \mid s=(\eta, \sigma) \wedge \sigma(\eta(x)) \in \text{Val} \right\} \\
& \llbracket \text{free}(x) \rrbracket_{\text{er}} \triangleq \left\{ (s, s) \mid s=(\eta, \sigma) \wedge (\eta(x)=\text{nil} \vee \sigma(\eta(x))=\perp) \right\} \\
& \llbracket x := [y] \rrbracket_{\text{ok}} \triangleq \left\{ (s, (\eta[x \mapsto v], \sigma)) \mid s=(\eta, \sigma) \wedge \sigma(\eta(y))=v \in \text{Val} \right\} \\
& \llbracket x := [y] \rrbracket_{\text{er}} \triangleq \left\{ (s, s) \mid \sigma=(s, \text{h}()) \wedge (\eta(y)=\text{nil} \vee \sigma(\eta(y))=\perp) \right\} \\
& \llbracket [x] := y \rrbracket_{\text{ok}} \triangleq \left\{ (s, (\eta, \sigma[\eta(x) \mapsto \eta(y)])) \mid s=(\eta, \sigma) \wedge \sigma(\eta(x)) \in \text{Val} \right\} \\
& \llbracket [x] := y \rrbracket_{\text{er}} \triangleq \left\{ (s, s) \mid \sigma=(\eta, \sigma) \wedge (\eta(x)=\text{nil} \vee \sigma(\eta(x))=\perp) \right\} \\
& \llbracket \text{local } x. C \rrbracket_{\epsilon} \triangleq \left\{ ((\eta, \sigma), (\eta'[x \mapsto \eta(x)], \sigma')) \mid ((\eta[x \mapsto \text{nil}], \sigma), (\eta', \sigma')) \in \llbracket C \rrbracket_{\epsilon} \right\}
\end{aligned}$$

Fig. 8. The Pulse-X denotational semantics

A SEMANTICS

The Pulse-X denotational semantic sis given in Fig. 8, and is analogous to that of ISL in [Raad et al. 2020].

1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568

1569 **B ERROR TRACE OF NPE IN LISTING 1**

```

1570
1571 1 apps/lib/s_cb.c:959: error: Nullptr Dereference
1572 2   PISL found a potential null pointer dereference on line 959.
1573 3
1574 4 apps/lib/s_cb.c:957:23: in call to `app_malloc`
1575 5   955. static int ssl_excert_prepend(SSL_EXCERT **pexc)
1576 6   956. {
1577 7     957.   SSL_EXCERT *exc = app_malloc(sizeof(*exc), "prepend cert");
1578 8     ^
1579 9
1580 10   958.
1581 11   959.   memset(exc, 0, sizeof(*exc));
1582 12
1583 13 test/testutil/apps_mem.c:16:16: in call to `CRYPTO_malloc` (modelled)
1584 14   14. void *app_malloc(size_t sz, const char *what)
1585 15   15. {
1586 16     16.   void *vp = OPENSSL_malloc(sz);
1587 17     ^
1588 18
1589 19   17.
1590 20   18.   return vp;
1591 21
1592 22 test/testutil/apps_mem.c:16:16: is the null pointer
1593 23   14. void *app_malloc(size_t sz, const char *what)
1594 24   15. {
1595 25     16.   void *vp = OPENSSL_malloc(sz);
1596 26     ^
1597 27
1598 28   17.
1599 29   18.   return vp;
1600 30
1601 31 test/testutil/apps_mem.c:16:5: assigned
1602 32   14. void *app_malloc(size_t sz, const char *what)
1603 33   15. {
1604 34     16.   void *vp = OPENSSL_malloc(sz);
1605 35     ^
1606 36
1607 37   17.
1608 38   18.   return vp;
1609 39     ^
1610 40
1611 41   19. }
1612
1613 43 apps/lib/s_cb.c:957:23: return from call to `app_malloc`
1614 44   955. static int ssl_excert_prepend(SSL_EXCERT **pexc)
1615 45   956. {
1616 46     957.   SSL_EXCERT *exc = app_malloc(sizeof(*exc), "prepend cert");
1617 47     ^
1618 48
1619 49   958.
1620 50   959.   memset(exc, 0, sizeof(*exc));

```

```

1618 51 apps/lib/s_cb.c:957:5: assigned
1619 52     955. static int ssl_excert_prepend(SSL_EXCERT **pexc)
1620 53     956. {
1621 54         957.     SSL_EXCERT *exc = app_malloc(sizeof(*exc), "prepend cert");
1622 55             ^
1623 56
1624 57     958.     memset(exc, 0, sizeof(*exc));
1625 58
1626 59 apps/lib/s_cb.c:959:5: invalid access occurs here
1627 60     957.     SSL_EXCERT *exc = app_malloc(sizeof(*exc), "prepend cert");
1628 61     958.
1629 62         959.     memset(exc, 0, sizeof(*exc));
1630 63             ^
1631 64
1632 65     960.     exc->next = *pexc;

```

Listing 10. Error trace of the bug in Listing 1.

```

1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666

```

1667 C MANIFEST ERRORS

1668 THEOREM C.1. For all manifest errors $\models [p] C [er : q]$:

$$1670 \quad \forall s. \exists s'. (s, s') \in \llbracket C \rrbracket_{er} \wedge s' \in (q * \text{true})$$

1671 PROOF. Pick a valid ISL triple $\models [p] C [er : q]$ denoting a manifest error. Pick arbitrary s and let
 1672 $r \triangleq \{s\}$. From Def. 3.2 we then know there exists f such that $p * f \vdash r$ and $\text{sat}(q * f)$ holds. As
 1673 $[p] C [er : q]$ is a valid triple, using the Frame rule of ISL we can derive $\models [p * f] C [er : q * f]$. Subsequently, since $p * f \vdash r$, from Def. 3.1 we also have $\models [r] C [er : q * f]$. On the other hand, as
 1674 $\text{sat}(q * f)$ holds, we know there exists s' such that $s' \in (q * f)$ and thus $s' \in (q * \text{true})$. Moreover,
 1675 from $\models [r] C [er : q * f]$, Def. 3.1 and the definitions of r we know $(q * f) \subseteq \llbracket C \rrbracket_{er}(\{s\})$; i.e.,
 1676 $\forall s_q \in (q * f). (s, s_q) \in \llbracket C \rrbracket_{er}$, and thus $(s, s') \in \llbracket C \rrbracket_{er}$, as required. \square
 1677

1678 **Manifest Errors and Reverse Under-Approximate Triples.** We next demonstrate that under
 1679 certain conditions manifest errors coincide with *reverse* under-approximate triples. We write $\models \{p\}$
 1680 $C \{\epsilon : q\}$ to denote a reverse (under-approximate) triple that is valid. Intuitively, a valid reverse
 1681 triple is the dual of a valid ISL triple (Def. 3.1): $\models \{p\} C \{\epsilon : q\}$ denotes that executing C on each
 1682 state in p reaches some state in q under ϵ , while preserving the frame property. Put formally:

$$1683 \quad \models \{p\} C \{\epsilon : q\} \stackrel{\text{def}}{\Leftrightarrow} \forall r. (p * r) \subseteq \llbracket C \rrbracket_\epsilon^{-1}((q * r))$$

1684 where $\llbracket C \rrbracket_\epsilon^{-1}$ denotes the inverse of the $\llbracket C \rrbracket_\epsilon$ relation in Def. 3.1. The notion of reverse under-
 1685 approximate triples corresponds to that of *total Hoare triples* in [de Vries and Koutavas 2011].

1686 Interestingly, as we show in Theorem C.2 below, given a manifest error $T \triangleq \models [p] C [er : q]$, if
 1687 $p \equiv \text{emp} \wedge \text{true}$ (i.e., p imposes no spatial or pure constraints on the context), then T is also a valid
 1688 reverse triple, i.e., $\models \{p\} C \{\epsilon : q\}$ also holds.

1689 THEOREM C.2. Given a manifest error $\models [p] C [er : q]$, if $p \equiv \text{emp} \wedge \text{true}$, then $\models \{p\} C \{\epsilon : q\}$.

1690 PROOF. Pick a manifest error $T \triangleq \models [p] C [er : q]$ such that $p \equiv \text{emp} \wedge \text{true}$. Pick arbitrary r and
 1691 $s \in (p * r)$; it then suffices to show there exists $s' \in (q * r)$ such that $(s, s') \in \llbracket C \rrbracket_\epsilon$.

1692 Let $r' \triangleq \{s\}$; as $\text{sat}(r')$ holds and T denotes a manifest error, from Def. 3.2 we know $\text{sat}(q * r')$
 1693 holds. Moreover as $s \in (p * r)$ and $p \equiv \text{emp} \wedge \text{true}$ (and thus from the semantics of assertions
 1694 $p * r \equiv r$), we know $s \in (r)$ and thus $r' \vdash r$. Moreover, as $\text{sat}(q * r')$ holds, we know there exists s'
 1695 such that $s' \in (q * r')$ and thus $s' \in (q * r)$ since $r' \vdash r$. On the other hand, as T is a valid triple,
 1696 from Def. 3.1 we know $(q * r') \subseteq \llbracket C \rrbracket_\epsilon((p * r'))$ and thus $(q * r') \subseteq \llbracket C \rrbracket_\epsilon((r'))$ since
 1697 $p \equiv \text{emp} \wedge \text{true}$. Consequently, as $s' \in (q * r')$, we know $s' \in \llbracket C \rrbracket_\epsilon((r'))$, i.e., $(s, s') \in \llbracket C \rrbracket_\epsilon$ since
 1698 $r' \triangleq \{s\}$. That is, there exists $s' \in (q * r)$ such that $(s, s') \in \llbracket C \rrbracket_\epsilon$, as required. \square

1699 Definition C.3 (Path-manifest errors). An error triple $\models [p] C [er : q]$ denotes a *path-manifest*
 1700 error iff for all r , if $\text{sat}(p * r)$ holds then $\text{sat}(q * r)$ also holds.

1701 THEOREM C.4 (PATH-MANIFEST ERRORS). An error triple $\models [p] C [er : q]$ with $p \triangleq \exists \vec{X}_p. \kappa_p \wedge \pi_p$
 1702 and $q \triangleq \exists \vec{X}_q. \kappa_q \wedge \pi_q$ denotes a path-manifest error if:

- 1703 (1) $\text{sat}(q)$ holds;
- 1704 (2) $\text{locs}(\kappa_q) \setminus \vec{X}_q \subseteq \text{locs}(\kappa_p) \setminus \vec{X}_p$;
- 1705 (3) for all \vec{v} , $\text{sat}(\pi_q[\vec{v} / \vec{Y}] \cup \text{locs}(\kappa_q))$ holds, where $\vec{Y} = \text{flv}(q)$ and:

1706 $\text{locs}(\text{emp}) \triangleq \emptyset \quad \text{locs}(x \mapsto X) \triangleq \{x\} \quad \text{locs}(X \mapsto V) = \text{locs}(X \not\mapsto) \triangleq \{X\} \quad \text{locs}(\kappa_1 * \kappa_2) \triangleq \text{locs}(\kappa_1) \cup \text{locs}(\kappa_2)$

PROOF. Pick arbitrary $p, q, r, C, \vec{X}_p, \kappa_p, \pi_p, \vec{X}_q, \kappa_q, \pi_q$ and \vec{Y} such that $p \triangleq \exists \vec{X}_p. \kappa_p \wedge \pi_p$, $q \triangleq \exists \vec{X}_q. \kappa_q \wedge \pi_q$, $\vec{Y} = \text{flv}(q)$, $\text{sat}(q)$ holds, $\text{locs}(\kappa_q) \setminus \vec{X}_q \subseteq \text{locs}(\kappa_p) \setminus \vec{X}_p$, for all \vec{v} , $\text{sat}(\pi_q[\vec{v}/\vec{Y} \cup \text{locs}(\kappa_q)])$ holds and $\text{sat}(p * r)$ holds.

As $\text{sat}(p * r)$ holds, we know there exist $\eta, \eta_p, \sigma_p, \sigma_r, \vec{v}_p$ such that $\eta_p = \eta[\vec{X}_p \mapsto \vec{v}_p]$, $\eta_p, \sigma_p \models \kappa_p$, $\eta_p \models \pi_p, \eta, \sigma_r \models r$ and $\sigma_p \# \sigma_r$, i.e., $(\eta(\text{locs}(\kappa_p)) \setminus \vec{X}_p) \cap \text{dom}(\sigma_r) = \emptyset$.

As $\text{locs}(\kappa_q) \setminus \vec{X}_q \subseteq \text{locs}(\kappa_p) \setminus \vec{X}_p$, we know there exist \vec{Z}_1, \vec{Z}_2 such that $\text{locs}(\kappa_q) = \vec{Z}_1 \uplus \vec{Z}_2$, $\vec{Z}_1 \cap \vec{X}_q = \emptyset$ (i.e., $\vec{Z}_1 \subseteq \vec{Y}$), $\vec{Z}_1 \subseteq \text{locs}(\kappa_p) \setminus \vec{X}_p$ and $\vec{Z}_2 \subseteq \vec{X}_q$. Note that as $\vec{Z}_1 \subseteq \text{locs}(\kappa_p) \setminus \vec{X}_p$ and $\eta_p, \sigma_p \models \kappa_p$, from the definition of η we know that $\eta \models \text{disjoint}(\vec{Z}_1)$ holds.

Pick \vec{v}_2 such that $\vec{v}_2 \cap \text{dom}(\sigma_r) = \emptyset$ and $\vec{v}_2 \cap \eta(\vec{Z}_1) = \emptyset$. Let $\pi_1 = \pi[\vec{Y} \mapsto \vec{\eta}(\vec{Y})]$ and $\pi_2 = \pi_1[\vec{Z}_2 \mapsto \vec{v}_2]$. As for all \vec{v} , $\text{sat}(\pi_q[\vec{v}/\vec{Y} \cup \text{locs}(\kappa_q)])$ holds, we know that $\text{sat}(\pi_2)$ holds and thus there exists η_q, \vec{v}_3 such that $\eta_q = \eta[\vec{Z}_2 \mapsto \vec{v}_2][(\vec{X}_q \setminus \vec{Z}_2) \mapsto \vec{v}_3]$ and $\eta_q \models \pi_q$. That is, there exist \vec{v}_q such that $\eta_q = \eta[\vec{X}_q \mapsto \vec{v}_q]$, $\eta_q(\vec{Y}) = \eta(\vec{Y})$ and $\eta_q(\vec{Z}_2) = \vec{v}_2$. Moreover, since $\eta \models \text{disjoint}(\vec{Z}_1)$, $\vec{Z}_1 \subseteq \vec{Y}$, $\eta_q(\vec{Y}) = \eta(\vec{Y})$, $\vec{v}_2 \cap \eta(\vec{Z}_1) = \emptyset$ and $\eta_q(\vec{Z}_2) = \vec{v}_2$, we also have $\eta_q \models \text{disjoint}(\vec{Z}_1 \cup \vec{Z}_2)$. As such, since $\text{locs}(\kappa_q) = \vec{Z}_1 \uplus \vec{Z}_2$, from [Proposition C.9](#) we have $\eta_q, \sigma_q \models \kappa_q$, where $\sigma_q = [\kappa_q]_{\eta_q}$. Consequently, as $\eta_q = \eta[\vec{X}_q \mapsto \vec{v}_q]$, $\eta_q, \sigma_q \models \kappa_q$ and $\eta_q \models \pi_q$ we have $\eta, \sigma_q \models \exists \vec{X}_q. \kappa_q \wedge \pi_q$ and thus $\eta, \sigma_q \models q$.

Lastly, since $\text{dom}(\sigma_q) = \eta_q(\vec{Z}_1) \uplus \eta_q(\vec{Z}_2) = \eta(\vec{Z}_1) \uplus \vec{v}_2, \vec{v}_2 \cap \text{dom}(\sigma_r) = \emptyset$, $(\eta(\text{locs}(\kappa_p)) \setminus \vec{X}_p) \cap \text{dom}(\sigma_r) = \emptyset$ and thus $\eta(\vec{Z}_1) \cap \text{dom}(\sigma_r) = \emptyset$ (since $\vec{Z}_1 \subseteq \text{locs}(\kappa_p) \setminus \vec{X}_p$), we also know that $\text{dom}(\sigma_q) \cap \text{dom}(\sigma_r) = \emptyset$ and thus $\sigma_q \# \sigma_r$. Consequently, as $\eta, \sigma_q \models q, \eta, \sigma_r \models r$ and $\sigma_q \# \sigma_r$, we know $\eta, \sigma_q \uplus \sigma_r \models q * r$ and thus $\text{sat}(q * r)$, as required. \square

[Definition C.5 \(Resource-manifest errors\).](#) An error triple $\models [p] C [er: q]$ denotes a *resource-manifest error* iff:

- $\models [p] C [er: q]$ denotes a path-manifest error; and
- $\text{pheap}(p)$ holds, where

$$\begin{aligned} \text{pheap}(\exists \vec{X}. \kappa \wedge \pi) &\stackrel{\text{def}}{\iff} \text{nlocs}(\kappa) = \emptyset \wedge \pi \equiv \text{true} \\ \text{nlocs}(\text{emp}) &\triangleq \emptyset & \text{nlocs}(x \mapsto V) &\triangleq \emptyset & \text{nlocs}(X \mapsto V) &\triangleq \emptyset \\ \text{nlocs}(X \not\mapsto) &\triangleq \{X\} & \text{nlocs}(\kappa_1 * \kappa_2) &= \text{nlocs}(\kappa_1) \cup \text{nlocs}(\kappa_2) \end{aligned}$$

[THEOREM C.6 \(RESOURCE-MANIFEST ERRORS\).](#) An error triple $\models [p] C [er: q]$ with $p \triangleq \exists \vec{X}_p. \kappa_p \wedge \pi_p$ and $q \triangleq \exists \vec{X}_q. \kappa_q \wedge \pi_q$ denotes a resource-manifest error if:

- (1) $\text{sat}(q)$ holds;
- (2) $\text{locs}(\kappa_q) \setminus \vec{X}_q \subseteq \text{locs}(\kappa_p) \setminus \vec{X}_p$;
- (3) for all \vec{v} , $\text{sat}(\pi_q[\vec{v}/\vec{Y} \cup \text{locs}(\kappa_q)])$ holds, where $\vec{Y} = \text{flv}(q)$;
- (4) $\text{pheap}(p)$ holds.

PROOF. Follows from [Theorem C.4](#) and the definitions of $\text{pheap}()$ and resource-manifest errors. \square

[Definition C.7 \(Manifest errors\).](#) An error triple $\models [p] C [er: q]$ denotes a *manifest error* iff for all r , if $\text{sat}(r)$ holds, then there exists f such that $p * f \vdash r$ and $\text{sat}(q * f)$ also holds.

[THEOREM C.8 \(MANIFEST ERRORS\).](#) An error triple $\models [p] C [er: q]$ with $q \triangleq \exists \vec{X}_q. \kappa_q \wedge \pi_q$ denotes a manifest error if:

- 1765 (1) $p \equiv \text{emp} \wedge \text{true};$
 1766 (2) $\text{sat}(q)$ holds;
 1767 (3) $\text{locs}(\kappa_q) \setminus \vec{X}_q \subseteq \text{locs}(\kappa_p) \setminus \vec{X}_p;$
 1768 (4) for all \vec{v} , $\text{sat}(\pi_q[\vec{v}/\vec{Y}] \cup \text{locs}(\kappa_q))$ holds, where $\vec{Y} = \text{flv}(q).$

1770 PROOF. Pick an arbitrary error triple $\models [p] C [er: q]$ such that conditions (1)-(4) above hold. Pick
 1771 an arbitrary r such that $\text{sat}(r)$ holds. As $p \equiv \text{emp} \wedge \text{true}$, we then have $p * r \equiv r$, and thus $p * r \vdash r$.
 1772 Moreover, from conditions (2)-(4) and Theorem C.4 we know $\models [p] C [er: q]$ is a path-manifest
 1773 error. On the other hand, as $\text{sat}(r)$ holds and $p * r \equiv r$, we know $\text{sat}(p * r)$ and thus from the
 1774 definition of path-manifest errors we have $\text{sat}(q * r)$, as required. \square

1775 PROPOSITION C.9. For all $\kappa, \pi, \eta, \sigma, \sigma_1, \sigma_2, \kappa_1, \kappa_2$:

- 1776 (1) if $\eta, \sigma_1 \models \kappa$ and $\eta, \sigma_2 \models \kappa$, then $\sigma_1 = \sigma_2$;
 1777 (2) $\kappa \equiv \kappa \wedge \text{disjoint}(\text{locs}(\kappa))$, where:

$$1779 \text{disjoint}(\emptyset) = \text{disjoint}(\{X\}) \triangleq \text{true} \quad \text{disjoint}(\{X\} \uplus S) \triangleq \bigwedge_{Y \in S} X \neq Y \wedge \text{disjoint}(S)$$

- 1780
 1781 (3) if $\eta, \sigma \models \kappa$, then $\eta \models \text{disjoint}(\text{locs}(\kappa))$ (follows from the previous part);
 1782 (4) if $\eta \models \text{disjoint}(\text{locs}(\kappa))$, then $\eta, [\kappa]_\eta \models \kappa$, where:

$$1783 \begin{array}{lll} [\text{emp}]_\eta = \emptyset & [x \mapsto X]_\eta = [\eta(x) \mapsto \eta(V)] & [X \mapsto Y]_\eta = [\eta(X) \mapsto \eta(Y)] \\ 1784 & [X \not\mapsto]_\eta = [\eta(X) \mapsto \perp] & [\kappa_1 * \kappa_2]_\eta = [\kappa_1]_\eta \uplus [\kappa_2]_\eta \\ 1785 \end{array}$$

- 1786 (5) if $\eta, \sigma \models \kappa$, then $\sigma = [\kappa]_\eta$ (follows from parts 1, 3 and 4).

1787

1788

1789

1790

1791

1792

1793

1794

1795

1796

1797

1798

1799

1800

1801

1802

1803

1804

1805

1806

1807

1808

1809

1810

1811

1812

1813