Reasoning about Software in the Presence of Transient Faults -- Complete Proofs

Frances Perry

Princeton University Technical Report TR-831-08

August 2008

This file contains the complete proofs for my thesis (Princeton University Technical Report TR-830-08): Reasoning About Software in the Presence of Transient Faults. These were my working notes and contain little or no explanations and probably a few errors. For a good explanation of the concepts, I highly suggest reading the thesis instead (or at least first). The body of the thesis contains overviews of the proofs, and the appendices contain more detailed proof sketches.

These notes are written in ascii text using abbrieviations like |- for \vdash and G for \Gamma. A few of the symbols may differ from those in the thesis, but the provided syntax for each chapter should clear up any confusion. Also, the use of ascii has resulted in some symbol duplication. For example, S is used for both \Sigma and substitutions, and G is used for both green and \Gamma. The context should make it clear which is meant.

Should you choose to continue beyond this point, I wish you luck!

TAL_FT Proofs

The following notes correspond to Chapter 2 and Appendix A.

Complete Rules -- including those omitted from the paper

Lemmas -- lemmas used in the theorems

Theorem 1: Progress Part 1 and Part 2

Theorem 2: Preservation Part 1 and Part 2

Corollary 3: No False Positives

Lemma: Singlestep Fault Detection

Theorem 4: Fault Tolerance

Top Level

ETAL_FT Proofs

These proofs correspond to Chapter 3 and Appendices B - E. Often the changes from TAL_FT are specifically noted.

Complete Rules -- changes from TAL_FT are in blue, additions to TAL_FT are shown in red

Lemmas -- lemmas used in progress/preservation

Theorem : Progress Part 1 and Part 2

Theorem: Preservation Part 1 and Part 2

Corollary: No False Positives

Multistep -- definitions/lemmas about multistep

FD Lemmas -- lemmas about fault detection

Lemma: Singlestep Fault Detection

Theorem: Fault Tolerance

MiniC -- MiniC Language Definition

Translation -- translation definition

Theorem: Translation Theorem

[Top Level]

TAL_CF Proofs

This online appendix includes the full proofs for Chapter 4 and Appendices F-G.

A few of the symbols differ from those in the thist, including R for orange, A for \sigma, to for \tau opt, and alpha and Y as expression variables. The provided syntax should clear up any confusion.

The Control-Flow Machine

- Syntax
  - Machine State Syntax (includes typing syntax too).
- Dynamic Semantics
  - Non-faulty and faulty single step operational semantics.

Typing

- Typing Rules
- Lemmas used by Progress & Preservation
- Progress
- Preservation

Fault Tolerance

- Definitions - definitions needed for fault tolerance, including block evaluation, transition evaluation, and program execution
- Lemmas - useful lemmas for fault tolerance
- CF Recovery Lemma - When there has been a cf fault, control always transfers to the recovery code before leaving the block
- Block Lemmas - Outcome of evaluating a block

- Transition Lemmas - Outcome of transitioning between blocks
- Fault Tolerance Theorem

Translation

- Definitions - Defintion of while language, wellformedness, type translation, etc
- Lemmas - Lemmas used in the translation theorem
- Translation Theorem - Well typed while language programs translate into well typed assembly programs

Top Level

Fault-tolerant Typed Assembly Language
Complete Rules

$$
\begin{array}{llll}
\textit{colors} & c & ::= & G \mid B \\
\textit{colored values} & v & ::= & c\,n \\
\textit{registers} & r & ::= & r_n \\
\textit{general regs} & a & ::= & r \mid d \mid pc_c \\
\textit{register file} & R & ::= & \cdot \mid R, a \to v \\
\textit{code memory} & C & ::= & \cdot \mid C, n \to i \\
\textit{value memory} & M & ::= & \cdot \mid M, n \to n \\
\textit{store queue} & Q & ::= & \overline{(n,n)} \\
\textit{ALU ops} & op & ::= & add \mid sub \mid mul \\
\textit{instructions} & i & ::= & op\; r_d, r_s, r_t \mid op\; r_d, r_s, v \\
 & & \mid & ld_c\; r_d, r_s \mid st_c\; r_d, r_s \mid mov\;\; r_d, v \\
 & & \mid & bz_c\;\; r_z, r_d \mid jmp_c\;\; r_d \\
\textit{inst register} & ir & ::= & i \mid \cdot \\
\textit{state} & \Sigma & ::= & (R, C, M, Q, ir) \mid fault
\end{array}
$$

Figure 1. Syntax of FM states

$$
\frac{R(a) = c\,n}{(R, C, M, Q, ir) \longrightarrow_1 (R[a \mapsto c\,n'], C, M, Q, ir)}\ \textit{(reg-zap)}
$$

$$
\frac{\begin{array}{c} Q_1 = \overline{(n_1, n_1')}, (m_1, m'), \overline{(n_2, n_2')} \\ Q_2 = \overline{(n_1, n_1')}, (m_2, m'), \overline{(n_2, n_2')} \end{array}}{(R, C, M, Q_1, ir) \longrightarrow_1 (R, C, M, Q_2, ir)}\ (Q_1\textit{-zap})
$$

$$
\frac{\begin{array}{c} Q_1 = \overline{(n_1, n_1')}, (m, m_1'), \overline{(n_2, n_2')} \\ Q_2 = \overline{(n_1, n_1')}, (m, m_2'), \overline{(n_2, n_2')} \end{array}}{(R, C, M, Q_1, ir) \longrightarrow_1 (R, C, M, Q_2, ir)}\ (Q_2\textit{-zap})
$$

Figure 2. Fault Rules

Instruction Fetch:

$$
\frac{R_{val}(pc_G) = R_{val}(pc_B) \quad R_{val}(pc_G) \in\ Dom(C)}{(R, C, M, Q, \cdot) \longrightarrow_0 (R, C, M, Q, C(R_{val}(pc_G)))}\ \textit{(fetch)}
$$

$$
\frac{R_{val}(pc_G) \neq R_{val}(pc_B)}{(R, C, M, Q, \cdot) \longrightarrow_0\ fault}\ \textit{(fetch-fail)}
$$

Basic Instructions:

$$
\frac{R' = R\text{++}[r_d \mapsto R_{col}(r_t)\ (R_{val}(r_s)\ op\ R_{val}(r_t))]}{(R, C, M, Q, op\;\; r_d, r_s, r_t) \longrightarrow_0 (R', C, M, Q, \cdot)}\ \textit{(op2r)}
$$

$$
\frac{R' = R\text{++}[r_d \mapsto c(R_{val}(r_s)\ op\ n)]}{(R, C, M, Q, op\;\; r_d, r_s, c\,n) \longrightarrow_0 (R', C, M, Q, \cdot)}\ \textit{(op1r)}
$$

$$
\frac{R' = R\text{++}[r_d \mapsto v]}{(R, C, M, Q, mov\;\; r_d, v) \longrightarrow_0 (R', C, M, Q, \cdot)}\ \textit{(mov)}
$$

Figure 3. Operational rules for basic instructions

$\boxed{\Sigma \longrightarrow_k^s \Sigma'}$

$$\frac{Q' = ((R_{val}(r_d), R_{val}(r_s)), Q)}{(R, C, M, Q, st_G\ \ r_d, r_s) \longrightarrow_0 (R\texttt{++}, C, M, Q', \cdot)}\ (st_G\text{-}queue)$$

$$\frac{R_{val}(rd) = n_1 \qquad R_{val}(r_s) = n_1'}{\begin{array}{l}(R, C, M, (\overline{(n, n')}, (n_l, n_l')), st_B\ \ r_d, r_s) \\ \quad \longrightarrow_0^{(n_l, n_l')} (R\texttt{++}, C, M[n_l \mapsto n_l'], \overline{(n, n')}, \cdot)\end{array}}\ (st_B\text{-}mem)$$

$$\frac{\begin{array}{c}Q = (\overline{(n, n')}, (n_l, n_l')) \\ R_{val}(r_d) \neq\ n_l \textit{ or } R_{val}(r_s) \neq\ n_l'\end{array}}{(R, C, M, Q, st_B\ \ r_d, r_s) \longrightarrow_0\ \ fault}\ (st_B\text{-}mem\text{-}fail)$$

$$\frac{\begin{array}{c}find(Q, R_{val}(r_s)) = (R_{val}(r_s), n) \\ R' = R\texttt{++}[r_d \mapsto G\ n]\end{array}}{(R, C, M, Q, ld_G\ \ r_d, r_s) \longrightarrow_0 (R', C, M, Q, \cdot)}\ (ld_G\text{-}queue)$$

$$\frac{\begin{array}{c}find(Q, R_{val}(r_s)) = () \\ R_{val}(r_s) \in\ Dom(M) \\ R' = R\texttt{++}[r_d \mapsto G\ M(R_{val}(r_s))]\end{array}}{(R, C, M, Q, ld_G\ \ r_d, r_s) \longrightarrow_0 (R', C, M, Q, \cdot)}\ (ld_G\text{-}mem)$$

$$\frac{\begin{array}{c}R_{val}(r_s) \in\ Dom(M) \\ R' = R\texttt{++}[r_d \mapsto B\ M(R_{val}(r_s))]\end{array}}{(R, C, M, Q, ld_B\ \ r_d, r_s) \longrightarrow_0 (R', C, M, Q, \cdot)}\ (ld_B\text{-}mem)$$

$$\frac{\begin{array}{c}find(Q, R_{val}(r_s)) = () \\ R_{val}(r_s) \notin\ Dom(M)\end{array}}{(R, C, M, Q, ld_G\ \ r_d, r_s) \longrightarrow_0\ \ fault}\ (ld_G\text{-}fail)$$

$$\frac{R_{val}(r_s) \notin\ Dom(M)}{(R, C, M, Q, ld_B\ \ r_d, r_s) \longrightarrow_0\ \ fault}\ (ld_B\text{-}fail)$$

$$\frac{\begin{array}{c}find(Q, R_{val}(r_s)) = () \\ R_{val}(r_s) \notin\ Dom(M) \\ R' = R\texttt{++}[r_d \mapsto G\ n]\end{array}}{(R, C, M, Q, ld_G\ \ r_d, r_s) \longrightarrow_0 (R', C, M, Q, \cdot)}\ (ld_G\text{-}rand)$$

$$\frac{\begin{array}{c}R_{val}(r_s) \notin\ Dom(M) \\ R' = R\texttt{++}[r_d \mapsto B\ n]\end{array}}{(R, C, M, Q, ld_B\ \ r_d, r_s) \longrightarrow_0 (R', C, M, Q, \cdot)}\ (ld_B\text{-}rand)$$

Figure 4. Operational rules for memory instructions.

$$\boxed{\Sigma \longrightarrow_k^s \Sigma'}$$

$$\frac{R_{val}(d) = 0 \qquad R' = R\text{++}[d \mapsto R(r_d)]}{(R, C, M, Q, jmp_G \;\; r_d) \longrightarrow_0 (R', C, M, Q, \cdot)}\ (\textit{jmp}_G)$$

$$\frac{R_{val}(d) \neq 0}{(R, C, M, Q, jmp_G \;\; r_d) \longrightarrow_0 \; fault}\ (jmp_G\textit{-fail})$$

$$\frac{\begin{array}{c} R_{val}(d) \neq 0 \qquad R_{val}(r_d) = R_{val}(d) \\ R' = R[pc_G \mapsto R(d)][pc_B \mapsto R(r_d)][d \mapsto G\;0] \end{array}}{(R, C, M, Q, jmp_B \;\; r_d) \longrightarrow_0 (R', C, M, Q, \cdot)}\ (\textit{jmp}_B)$$

$$\frac{R_{val}(r_d) \neq R_{val}(d) \;\; or \; R_{val}(d) = 0}{(R, C, M, Q, jmp_B \;\; r_d) \longrightarrow_0 \; fault}\ (jmp_B\textit{-fail})$$

$$\frac{R_{val}(d) = 0 \qquad R_{val}(r_z) \neq 0}{(R, C, M, Q, bz_c \;\; r_z, r_d) \longrightarrow_0 (R\text{++}, C, M, Q, \cdot)}\ (\textit{bz-untaken})$$

$$\frac{\begin{array}{c} R_{val}(r_z) \neq 0 \\ R_{val}(d) \neq 0 \end{array}}{(R, C, M, Q, bz_c \;\; r_z, r_d) \longrightarrow_0 \; fault}\ (\textit{bz-untaken-fail})$$

$$\frac{\begin{array}{c} R_{val}(d) = 0 \qquad R_{val}(r_z) = 0 \\ R' = R\text{++}[d \mapsto R(r_d)] \end{array}}{(R, C, M, Q, bz_G \;\; r_z, r_d) \longrightarrow_0 (R', C, M, Q, \cdot)}\ (\textit{bz}_G\textit{-taken})$$

$$\frac{\begin{array}{c} R_{val}(r_z) = 0 \\ R_{val}(d) \neq 0 \end{array}}{(R, C, M, Q, bz_G \;\; r_z, r_d) \longrightarrow_0 \; fault}\ (\textit{bz}_G\textit{-taken-fail})$$

$$\frac{\begin{array}{c} R_{val}(d) \neq 0 \qquad R_{val}(r_z) = 0 \\ R_{val}(r_d) = R_{val}(d) \\ R' = R[pc_G \mapsto R(d)][pc_B \mapsto R(r_d)][d \mapsto G\;0] \end{array}}{(R, C, M, Q, bz_B \;\; r_z, r_d) \longrightarrow_0 (R', C, M, Q, \cdot)}\ (\textit{bz}_B\textit{-taken})$$

$$\frac{\begin{array}{c} R_{val}(r_z) = 0 \\ R_{val}(r_d) \neq R_{val}(d) \;\; or \; R_{val}(d) = 0 \end{array}}{(R, C, M, Q, bz_B \;\; r_z, r_d) \longrightarrow_0 \; fault}\ (\textit{bz}_B\textit{-taken-fail})$$

Figure 5. Operational rules for control flow instructions.

Static Expressions

$exp\ kinds$	κ	$::=$	$\kappa_{int} \mid \kappa_{mem}$
$exp\ contexts$	Δ	$::=$	$\cdot \mid \Delta, x : \kappa$
$exps$	E	$::=$	$x \mid n \mid E\ op\ E \mid sel\ E_m\ E_n$
			$\mid emp \mid upd\ E_m\ E_{n_1}\ E_{n_2}$
$substitutions$	S	$::=$	$\cdot \mid S, E/x$

Types

$zap\ tags$	Z	$::=$	$\cdot \mid c$
$base\ types$	b	$::=$	$int \mid \Theta \rightarrow void \mid b\ ref$
$reg\ types$	t	$::=$	$\langle c, b, E \rangle \mid E' = 0 \Rightarrow \langle c, b, E \rangle$
$reg\ file\ types$	Γ	$::=$	$\cdot \mid \Gamma, a \rightarrow t$
$result\ types$	RT	$::=$	$\Theta \mid void$

Contexts

$heap\ typing$	Ψ	$::=$	$\cdot \mid \Psi, n : b$
$static\ context$	Θ	$::=$	$\Delta; \Gamma; \overline{(E_d, E_s)}; E_m$

Figure 6. TAL_{FT} type syntax.

$\boxed{\Psi \vdash\ n : b}$

$$\frac{}{\Psi \vdash\ n : int}\ (\textit{int-t}) \qquad \frac{}{\Psi \vdash\ n : \Psi(n)}\ (\textit{base-t})$$

$\boxed{\Psi; \Delta \vdash^{Z}\ v : t}$

$$\frac{\Psi \vdash\ n : b \qquad \Delta \vdash\ E = n}{\Psi; \Delta \vdash^{Z}\ c\, n : \langle c, b, E \rangle}\ (\textit{val-t})$$

$$\frac{n \neq 0 \qquad \Psi; \Delta \vdash^{Z}\ c\, n : \langle c, b, E \rangle \qquad \Delta \vdash\ E' = 0}{\Psi; \Delta \vdash^{Z}\ c\, n : E' = 0 \Rightarrow \langle c, b, E \rangle}\ (\textit{cond-t})$$

$$\frac{\Delta \vdash\ E' \neq 0}{\Psi; \Delta \vdash^{Z}\ c\, 0 : E' = 0 \Rightarrow \langle c, b, E \rangle}\ (\textit{cond-t-n0})$$

$$\frac{\Delta \vdash\ E : \kappa_{int}}{\Psi; \Delta \vdash^{c}\ c\, n : \langle c, b, E \rangle}\ (\textit{val-zap-t})$$

$$\frac{\Delta \vdash\ E' : \kappa_{int} \qquad \Delta \vdash\ E : \kappa_{int}}{\Psi; \Delta \vdash^{c}\ c\, n : E' = 0 \Rightarrow \langle c, b, E \rangle}\ (\textit{val-zap-cond})$$

Figure 7. Value Typing

$\boxed{\Delta \vdash\ E : \kappa}$

$$\frac{x \in Dom(\Delta)}{\Delta \vdash\ x : \Delta(x)}\ (\textit{E-var-t})$$

$$\frac{}{\Delta \vdash\ n : \kappa_{int}}\ (\textit{E-int-t})$$

$$\frac{\begin{array}{c}\Delta \vdash\ E_1 : \kappa_{int}\\ \Delta \vdash\ E_2 : \kappa_{int}\end{array}}{\Delta \vdash\ E_1\ op\ E_2 : \kappa_{int}}\ (\textit{E-op-t})$$

$$\frac{\begin{array}{c}\Delta \vdash\ E_m : \kappa_{mem}\\ \Delta \vdash\ E_n : \kappa_{int}\end{array}}{\Delta \vdash\ sel\ E_m\ E_n : \kappa_{int}}\ (\textit{E-sel-t})$$

$$\frac{\begin{array}{c}\Delta \vdash\ E_m : \kappa_{mem}\\ \Delta \vdash\ E_{n_1} : \kappa_{int}\\ \Delta \vdash\ E_{n_2} : \kappa_{int}\end{array}}{\Delta \vdash\ upd\ E_m\ E_{n_1}\ E_{n_2} : \kappa_{mem}}\ (\textit{E-upd-t})$$

$$\frac{}{\Delta \vdash\ emp : \kappa_{mem}}\ (\textit{E-emp-t})$$

$\boxed{\Delta \vdash\ S : \Delta'}$

$$\frac{}{\Delta \vdash\ \cdot : \cdot}\ (\textit{sub-emp-t})$$

$$\frac{\begin{array}{c}\Delta \vdash\ S : \Delta'\\ \Delta \vdash\ E : \kappa\\ x \notin Dom(\Delta) \cup Dom(\Delta')\end{array}}{\Delta \vdash\ S, E/x : \Delta', x : \kappa}\ (\textit{sub-t})$$

$\boxed{[\![E]\!]}$

$$\begin{array}{lcl}
[\![n]\!] & = & n\\
[\![emp]\!] & = & \cdot\\
[\![E_1\ op\ E_2]\!] & = & [\![E_1]\!]\ op\ [\![E_2]\!]\\
[\![sel\ E_m\ E_n]\!] & = & [\![E_m]\!]([\![E_n]\!])\\
[\![upd\ E_m\ E_1\ E_2]\!] & = & [\![E_m]\!][\ [\![E_1]\!] \mapsto [\![E_2]\!]\ ]
\end{array}$$

$\boxed{\Delta \vdash\ E_1 = E_2}$

$$\frac{\begin{array}{c}\Delta \vdash E_1 : \kappa_{int} \qquad \Delta \vdash E_2 : \kappa_{int}\\ \forall S.\ \cdot \vdash S : \Delta \implies [\![S(E_1)]\!] = [\![S(E_2)]\!]\end{array}}{\Delta \vdash\ E_1 = E_2}\ (\textit{E-eq})$$

$$\frac{\begin{array}{c}\Delta \vdash E_1 : \kappa_{int} \qquad \Delta \vdash E_2 : \kappa_{int}\\ \forall S.\ \cdot \vdash S : \Delta \implies [\![S(E_1)]\!] \neq [\![S(E_2)]\!]\end{array}}{\Delta \vdash\ E_1 \neq E_2}\ (\textit{E-neq})$$

$$\frac{\begin{array}{c}\Delta \vdash E_1 : \kappa_{mem} \qquad \Delta \vdash E_2 : \kappa_{mem}\\ \forall \ell \in\ Dom([\![S(E_1)]\!]) \cup Dom([\![S(E_2)]\!]).\ [\![S(E_1)]\!](\ell) = [\![S(E_2)]\!](\ell)\end{array}}{\Delta \vdash\ E_1 = E_2}\ (\textit{E-mem-eq})$$

Figure 8. Properties of Static Expressions

$\boxed{\Delta \vdash \ t \leq t'}$

$$\frac{\Delta \vdash \ E_1 = E_2}{\Delta \vdash \ \langle c, b, E_1 \rangle \leq \langle c, b, E_2 \rangle} \ (\textit{subtp-triple})$$

$$\frac{\Delta \vdash \ E_1 = E_2}{\Delta \vdash \ \langle c, b, E_1 \rangle \leq \langle c, int, E_2 \rangle} \ (\textit{subtp-int})$$

$$\frac{\Delta \vdash \ t \leq t' \qquad \Delta \vdash \ E = E'}{\Delta \vdash \ (E = 0 \Rightarrow t) \leq (E' = 0 \Rightarrow t')} \ (\textit{subtp-cond})$$

$\boxed{\Delta \vdash \ \Gamma_1 \leq \Gamma_2}$

$$\frac{\forall r \in Dom(\Gamma_2).\, \Gamma_1(r) \leq \Gamma_2(r)}{\Delta \vdash \ \Gamma_1 \leq \Gamma_2} \ (\textit{reg-file-comp})$$

Figure 9. Subtyping

$$\boxed{\Psi; \Theta \vdash\ ir\ \Rightarrow RT}$$

$$\frac{}{\Psi; (\Delta; \Gamma; \overline{(E_d, E_s)}; E_m) \vdash\ \cdot\ \Rightarrow (\Delta; \Gamma; \overline{(E_d, E_s)}; E_m)}\ (\cdot\text{-}t)$$

$$\frac{\Gamma(r_s) = \langle c, int, E'_s\rangle \qquad \Gamma(r_t) = \langle c, int, E'_t\rangle}{\Psi; (\Delta; \Gamma; \overline{(E_d, E_s)}; E_m) \vdash\ op\ \ r_d, r_s, r_t\ \Rightarrow (\Delta; \Gamma{+}{+}[r_d \mapsto\ \langle c, int, E'_s\ op\ E'_t\rangle]; \overline{(E_d, E_s)}; E_m)}\ (op2r\text{-}t)$$

$$\frac{\Gamma(r_s) = \langle c, int, E'_s\rangle}{\Psi; (\Delta; \Gamma; \overline{(E_d, E_s)}; E_m) \vdash\ op\ \ r_d, r_s, c\ n\ \Rightarrow (\Delta; \Gamma{+}{+}[r_d \mapsto\ \langle c, int, E'_s\ op\ n\rangle]; \overline{(E_d, E_s)}; E_m)}\ (op1r\text{-}t)$$

$$\frac{\Psi; \Delta \vdash\ v : t}{\Psi; (\Delta; \Gamma; \overline{(E_d, E_s)}; E_m) \vdash\ mov\ \ r_d, v\ \Rightarrow (\Delta; \Gamma{+}{+}[r_d \mapsto\ t]; \overline{(E_d, E_s)}; E_m)}\ (mov\text{-}t)$$

$$\frac{\Gamma(r_s) = \langle G, b\ ref, E'_s\rangle \qquad E\ =\ sel\ (\overline{upd}\ E_m\ \overline{(E_d, E_s)})\ E'_s}{\Psi; (\Delta; \Gamma; \overline{(E_d, E_s)}; E_m) \vdash\ ld_G\ r_d\ r_s\ \Rightarrow (\Delta; \Gamma{+}{+}[r_d \mapsto\ \langle G, b, E\rangle]; \overline{(E_d, E_s)}; E_m)}\ (ld_G\text{-}t)$$

$$\frac{\Gamma(r_s) = \langle B, b\ ref, E'_s\rangle \qquad E\ =\ sel\ E_m\ E'_s}{\Psi; (\Delta; \Gamma; \overline{(E_d, E_s)}; E_m) \vdash\ ld_B\ r_d\ r_s\ \Rightarrow (\Delta; \Gamma{+}{+}[r_d \mapsto\ \langle B, b, E\rangle]; \overline{(E_d, E_s)}; E_m)}\ (ld_B\text{-}t)$$

$$\frac{\Gamma(r_d) = \langle G, b\ ref, E'_d\rangle \qquad \Gamma(r_s) = \langle G, b, E'_s\rangle}{\Psi; (\Delta; \Gamma; \overline{(E_d, E_s)}; E_m) \vdash\ st_G\ r_d\ r_s\ \Rightarrow (\Delta; \Gamma{+}{+}; (E'_d, E'_s), \overline{(E_d, E_s)}; E_m)}\ (st_G\text{-}t)$$

$$\frac{\begin{array}{c}\Gamma(r_d) = \langle B, b\ ref, E''_d\rangle \quad \Gamma(r_s) = \langle B, b, E''_s\rangle \\ \Delta \vdash\ E'_s = E''_s \quad \Delta \vdash\ E'_d = E''_d\end{array}}{\Psi; (\Delta; \Gamma; \overline{(E_d, E_s)}, (E'_d, E'_s); E_m) \vdash\ st_B\ r_d\ r_s\ \Rightarrow (\Delta; \Gamma{+}{+}; \overline{(E_d, E_s)}; upd\ E_m\ E'_d\ E'_s)}\ (st_B\text{-}t)$$

$$\frac{\begin{array}{c}\Gamma(d) = \langle G, int, 0\rangle \qquad \Gamma(r_z) = \langle G, int, E_z\rangle \\ \Gamma(r_d) = \langle G, \Theta \to void, E'_d\rangle \qquad \Theta = (\Delta'; \Gamma'; \overline{(E'_d, E'_s)}; E'_m) \qquad \Gamma'(d) = \langle G, int, 0\rangle\end{array}}{\Psi; (\Delta; \Gamma; \overline{(E_d, E_s)}; E_m) \vdash\ bz_G\ r_z\ r_d\ \Rightarrow (\Delta; \Gamma{+}{+}[d \mapsto\ E_z = 0 \Rightarrow \langle G, \Theta \to void, E'_d\rangle]; \overline{(E_d, E_s)}; E_m)}\ (bz_G\text{-}t)$$

$$\frac{\begin{array}{c}\Gamma(r_d) = \langle G, \Theta \to void, E_{rd'}\rangle \qquad \Theta = (\Delta'; \Gamma'; \overline{(E'_d, E'_s)}; E'_m) \\ \Gamma(d) = \langle G, int, 0\rangle \qquad \Gamma'(d) = \langle G, int, 0\rangle\end{array}}{\Psi; (\Delta; \Gamma; \overline{(E_d, E_s)}; E_m) \vdash\ jmp_G\ r_d\ \Rightarrow (\Delta; \Gamma{+}{+}[d \mapsto\ \langle G, \Theta \to void, E_{rd'}\rangle]; \overline{(E_d, E_s)}; E_m)}\ (jmp_G\text{-}t)$$

$$\frac{\begin{array}{c}\Gamma(r_z) = \langle B, int, E_z\rangle \\ \Gamma(r_d) = \langle B, (\Delta'; \Gamma'; \overline{(E'_d, E'_s)}; E'_m) \to void, E_r\rangle \\ \Gamma(d) = E'_z = 0 \Rightarrow \langle G, (\Delta'; \Gamma'; \overline{(E'_d, E'_s)}; E'_m) \to void, E'_r\rangle \\ \Delta \vdash\ E_z = E'_z \\ \Delta \vdash\ E_r = E'_r \\ \exists S. \Delta \vdash\ S : \Delta' \\ S(\Gamma')(d) = \langle G, int, 0\rangle \\ S(\Gamma')(pc_G) = \langle G, int, E'_r\rangle \\ S(\Gamma')(pc_B) = \langle B, int, E_r\rangle \\ \Delta \vdash \Gamma \le\ S(\Gamma') \\ \Delta \vdash\ \overline{(E_d, E_s)} = S(\overline{(E'_d, E'_s)}) \\ \Delta \vdash\ E_m = S(E'_m)\end{array}}{\begin{array}{c}\Psi; (\Delta; \Gamma; \overline{(E_d, E_s)}; E_m) \vdash\ bz_B\ r_z\ r_d\ \Rightarrow \\ (\Delta; \Gamma{+}{+}; \overline{(E_d, E_s)}; E_m)\end{array}}\ (bz_B\text{-}t)$$

$$\frac{\begin{array}{c}\Gamma(d) = \langle G, (\Delta'; \Gamma'; \overline{(E'_d, E'_s)}; E'_m) \to void, E'_r\rangle \\ \Gamma(r_d) = \langle B, (\Delta'; \Gamma'; \overline{(E'_d, E'_s)}; E'_m) \to void, E_r\rangle \\ \Delta \vdash\ E_r = E'_r \\ \exists S. \Delta \vdash\ S : \Delta' \\ S(\Gamma')(d) = \langle G, int, 0\rangle \\ S(\Gamma')(pc_G) = \langle G, int, E'_r\rangle \\ S(\Gamma')(pc_B) = \langle B, int, E_r\rangle \\ \Delta \vdash \Gamma \le\ S(\Gamma') \\ \Delta \vdash\ \overline{(E_d, E_s)} = S(\overline{(E'_d, E'_s)}) \\ \Delta \vdash\ E_m = S(E'_m)\end{array}}{\Psi; (\Delta; \Gamma; \overline{(E_d, E_s)}; E_m) \vdash\ jmp_B\ r_d\ \Rightarrow void}\ (jmp_B\text{-}t)$$

Figure 10. Instruction Typing Rules

$\boxed{\Psi \vdash^Z R : \Gamma}$

$$\frac{\begin{array}{c}\forall a.\ \Psi; \cdot \vdash^Z R(a) : \Gamma(a) \\ \cdot \vdash \Gamma(pc_G) \leq \langle G, int, E_G \rangle \\ \cdot \vdash \Gamma(pc_B) \leq \langle B, int, E_B \rangle \\ \cdot \vdash E_G = E_B\end{array}}{\Psi \vdash^Z R : \Gamma}\ (\textit{reg-file-t})$$

$\boxed{\Psi \vdash C}$

$$\frac{\begin{array}{l}0 \notin Dom(C) \\ \forall n \in Dom(C). \quad \Psi(n) = \Theta \rightarrow void\ \wedge\ \Psi; \Theta \vdash C(n) \Rightarrow RT\ \wedge \\ \qquad\qquad\qquad\quad (RT = \Theta'\ implies\ \Psi(n+1) = \Theta' \rightarrow void)\end{array}}{\Psi \vdash C}\ (\textit{C-t})$$

$\boxed{\Psi \vdash M : E_m}$

$$\frac{\begin{array}{c}\cdot \vdash E_m : \kappa_{mem} \qquad [\![E_m]\!] = M \\ \forall \ell \in Dom(M).\ \Psi \vdash \ell : b\ ref\ \wedge\ \Psi \vdash M(\ell) : b\end{array}}{\Psi \vdash M : E_m}\ (\textit{M-t})$$

$\boxed{\Psi \vdash^Z Q : \overline{(E_d, E_s)}}$

$$\frac{}{\Psi \vdash^Z () : ()}\ (\textit{Q-emp-t})$$

$$\frac{\begin{array}{c}Z \neq G \\ \Psi \vdash^Z \overline{(n_1', n_2')} : \overline{(E_d', E_s')} \\ \cdot \vdash E_d = n_1 \\ \cdot \vdash E_s = n_2 \\ \Psi \vdash n_2 : b \\ \Psi \vdash n_1 : b\ ref\end{array}}{\Psi \vdash^Z (n_1, n_2), \overline{(n_1', n_2')} : (E_d, E_s), \overline{(E_d', E_s')}}\ (\textit{Q-t})$$

$$\frac{\begin{array}{c}\Psi \vdash^G \overline{(n_1', n_2')} : \overline{(E_d', E_s')} \\ \cdot \vdash E_d : \kappa_{int} \\ \cdot \vdash E_s : \kappa_{int}\end{array}}{\Psi \vdash^G (n_1, n_2), \overline{(n_1', n_2')} : (E_d, E_s), \overline{(E_d', E_s')}}\ (\textit{Q-zap-t})$$

$\boxed{\vdash^Z (R, C, M, Q, ir)}$

$$\frac{\begin{array}{c}Dom(\Psi) = Dom(C) \cup Dom(M) \\ Z \neq G \implies Dom(Q) \subseteq Dom(M) \\ \Psi \vdash C \\ \forall c \neq Z.\ ir \neq \cdot \implies C(R_{val}(pc_c)) = ir \\ \forall c \neq Z.\ \Psi(R_{val}(pc_c)) = (\Delta; \Gamma; \overline{(E_d, E_s)}; E_m) \rightarrow void \\ \exists S.\ \cdot \vdash S : \Delta \\ \Psi \vdash M : S(E_m) \\ \Psi \vdash^Z Q : S\overline{(E_d, E_s)} \\ \Psi \vdash^Z R : S(\Gamma)\end{array}}{\vdash^Z (R, C, M, Q, ir)}\ (\textit{heap-t})$$

Figure 11. Machine State Typing

$\boxed{v_1\ sim^Z\ v_2}$

$$\frac{}{C\ n\ sim^Z\ C\ n}\ (\textit{sim-val}) \qquad \frac{}{C\ n\ sim^C\ C\ n'}\ (\textit{sim-val-zap})$$

$\boxed{R\ sim^Z\ R'}$

$$\frac{\forall a.\ R(a)\ sim^Z\ R'(a)}{R\ sim^Z\ R'}\ (\textit{sim-R})$$

$\boxed{Q\ sim^Z\ Q'}$

$$\frac{}{\cdot\ sim^Z\ \cdot}\ (\textit{sim-Q-empty})$$

$$\frac{G\ n_1\ sim^Z\ G\ n_1' \qquad G\ n_2\ sim^Z\ G\ n_2' \qquad Q\ sim^Z\ Q'}{((n_1, n_2), Q)\ \ sim^Z\ ((n_1', n_2'), Q')}\ (\textit{sim-Q})$$

$\boxed{\Sigma_1\ sim^Z\ \Sigma_2}$

$$\frac{R\ sim^Z\ R' \qquad Q\ sim^Z\ Q'}{(R, C, M, Q, ir)\ sim^Z\ (R', C, M, Q', ir)}\ (\textit{sim-}\Sigma)$$

Figure 12. Similarity of Machine States

Lemmas for Progress, Preservation, and Simulation

```
Note: Since the sel-upd expression algebra we use is quite standard,
we have not formalized its properties ourselves.  We assume basic
properties of it such as reflexivity, transitivity, substitution of
equals for equals, the relation between sel and update, etc. wherever
necessary in the proof.




Int Kinding Lemma
------------------

If  P;D |-Z v : <c,b,E>  then D |- E : kint.

Proof: By case analysis on the value typing judgment.




Queue Lemma:
------------
1. If P |-z Q : seq(Ed,Es) then length(Q) = length(seq(Ed,Es)).
2. If P |-z seq(n1,n2) : seq(Ed,Es) and z not= G then
    for k:1..length(seq(n1,n2)),
      .|- Edk = n1k and .|- Eds = n2k and for some b, P |- n1k : b ref and P |- n2k : b.

Proof:  Both parts by induction on the queue typing judgement.




Find Lemma
-----------

1. If find(Q,n1) = () and P |-Z Q : seq(E_d,E_s) then for k:1...length(Q). . |- Edk =/= n1

Proof: ???





Irrelevant Update Lemma
-------------------------

If E = sel (upd E_m E_s E_d) E_s' and . |- E_s =/= E_s' then E = sel E_m Es'

Proof: ????



Exp Evaluation Lemma
--------------------

1. If . |- E : kint then Exists n. [[E]] = n
2. If . |- E : kmem then Exists M. [[E]] = M

Proof by induction on D |- E : k




Canonical Forms Lemma
---------------------

If Dom(P) = Dom(C) union Dom(M), and
 P |- M : E_m, and
```

```
 P |- C, and
 P; . |-z c n : t
then
1. If t = <c,b,E> or t = (E' = 0) => <c,b,E>, and c = z
   then no particular properties of n are known.
2. If t = <c,int,E> and c not= z then . |- E = n.
3. If t = <c,T-->void,E> and c not= z then
    P(n) = T --> void and n in Dom(C) and . |- E = n and n =/= 0.
4. If t = <c,b ref,E> and c not= z then
    P(n) = b ref and n in Dom(M) and . |- E = n.
5. If t = (E' = 0) => t, and c not= z and . |- E' = 0 then n is not 0.
6. If t = (E' = 0) => t, and c not= z and . |- E' not= 0 then n is 0.

Proof:  By induction on the derivation P; . |-z c n : t



Exp Eq Transitivity
-------------------

If D |- E1 = E2 and D |- E2 = E3  then D |- E1 = E3

Proof: Inversion and reconstruction on (E-eq)


Substituting Closed Expressions
-------------------------------

If . |- E : k then Forall S. . |- S(E) : k

Proof: by induction on D |- E : k


Subtyping Lemma:
----------------
If D |- t <= t' and P;D |-z v:t then P;D |-z v:t'

Proof:
By induction on the derivation of P;D |-z v:t.  Each case uses inversion
of the subtyping rules and transitivity of D |- E1 = E2.  Case cond-t-n0
also requires the property that if D |- E1 = E2 and D |- E2 not= E3 then
D |- E1 not= E3.


Substitution Lemma:
-------------------
1. If D,x:k |- E':k' and D |- E:k then D |- E'[E/x]:k'.
2. If D,x:k |- E1 = E2 and D |- E:k then D |- E1[E/x] = E2[E/x].
3. If D,x:k |- E1 not= E2 and D |- E:k then D |- E1[E/x] not= E2[E/x].
4. If P;D,x:k |-z v:t and D |- E:k then P;D |- v:t[E/x]
5. If P;D,x:k;G;seq(Ed,Es);Em |-z ir => RT and D |- E:k then
         P;D;G[E/x];seq(Ed,Es)[E/x];Em[E/x] |-z ir => RT[E/x]
6. If . |- S : D and P;D |-z v:t then P; . |-z v : S(t).
7. If . |- S : D and P;D;G;seq(Ed,Es);Em |-z ir => (D;G';seq(Ed',Es');Em') then
         P;.;S(G);S(seq(Ed,Es));S(Em) |-z ir => (.;S(G');S(seq(Ed',Es'));S(Em'))

Proof:
By induction on the respective typing derivation for parts 1, 4, 5.
Parts 6, 7 by induction on the size of D, using parts 4 and 5 respectively.
Parts 2 and 3 are assumed true of the expression algebra.  Note that
part 3 is slightly unusual.  It may be trivially implemented simply
by requiring that E1 not= E2 holds only when E1 and E2 are closed.  This
judgement is only needed to type states during the proof of preservation
after a conditional branch has been executed, when, indeed, the expressions
E1 and E2 will be closed.




Memory Lemma:
-------------
1.  If  P |- M : Em then Em = emp or Em = (upd ( ... (upd Em' En1k En2k) ... ) En11 En21.
2.  Moreover, if . |- En1k = n then n in Dom(M).

Proof:  By induction on the memory typing derivation P |- M : Em.


Memory Update Corollary:
------------------------
If  P |- M : Em and Em = (upd ( ... (upd Em' En1k En2k) ... ) En11 En21 and . |- sel Em n1 = n2
then n1 in Dom(M).
```

```
Proof:  By Memory Lemma and properties of the expression algebra.




Well-Typed Domain Lemma
------------------------

If |- (R1,C,M,Q1, ld_G rd, rs) then R1_val(r_d) in Dom(M)

Proof: By inversion of the ld_G-t type rule, inversion of the
register file typing rule and the Canonical Forms Lemma.



Color Weakening Lemma
----------------------

If P;.|- v:t
then P;.|-c v:t

Proof: By induction on the value typing judgement.



Color Weakening Q Lemma
------------------------

If P |- Q : seq(E_d,E_s)
then P |-c Q : seq(E_d,E_s)

Proof: By induction on the queue typing judgement.


Color Weakening R Lemma
------------------------

If P |- R : Gamma
then P |-c R : Gamma

Proof:  By inversion of the reg-file-t rule and the Color Weakening Lemma.
```

Progress Part 1

```
1. If |- (R,C,M,Q,ir) then (R,C,M,Q,ir) -->_0^s  (R',C',M',Q',ir')


Proof by case analysis on ir.



Case .:

1. |- (R,C,M,Q,.)                                                    | Given
2. P |- R : S(G)                                                     | Inversion of (heap-t), 1
3. . |- S(G)(pc_G) =< <G,int,E_G>                                    | Inversion of (reg-file-t), 2
   . |- S(G)(pc_B) =< <B,int,E_B>
4. P;.|- R(pc_G) : <G,int,E_G>                                       | 3, Inversion of (reg-file-t), 2, Subtyping Lemma
   P;.|- R(pc_B) : <B,int,E_B>
5. .|- E_G = R_val(pc_G)                                             | Inversion of (val-t), 4
   .|- E_B = R_val(pc_B)
6. .|- E_G = E_B                                                     | Inversion of (reg-file-t), 2
7. R_val(pc_G) = R_val(pc_B)                                         | Transitivity 5, 6
8. Forall c : P(R_val(pc_c)) = (D;G;seq(E_d,E_s),E_m) --> void       | Inversion of (heap-t), 1
9. R_val(pc_G) in Dom(C)                                             | 8, Inversion of (heap-t)
10. (R,C,M,Q,.) -->_0 (R,C,M,Q,C(R_val(pc_G)))                       | fetch 7, 9
*



Case op2r:

1. |- (R,C,M,Q, op r_d,r_s,r_t)                                      | Given
2. P;(.;S(G);S(seq(E_d,E_s)),S(E_m)) |- op  r_d,r_s,r_t => RT        | Inversion of (heap-t, C-t), substitution, 1
3. S(G)(r_s) = <c,int,E_s'>                                          | Inversion of (op2r-t), 2
   S(G)(r_t) = <c,int,E_t'>
4. P |- R : S(G)                                                     | Inversion of (heap-t), 1
5. P;. |- R(r_s) : < c,int,E_s'>                                     | Inversion of (reg-file-t), 4, 3
   P;. |- R(r_t) : < c,int,E_t'>
6. r_s in Dom(R)                                                     | 5
   r_t in Dom(R)
7. pc_G, pc_B in Dom(R)                                              | Inversion of (heap-t)
8. (R,C,M,Q,op r_d,r_s,r_t)                                          | op2r 6
    -->_0 (R++[r_d--> R_col(r_t) ( R_val(r_s) op R_val(r_t) )],C,M,Q,.)
*



Case op1r:

Similar to op2r.
*



Case mov:

1.  pc_G, pc_B in Dom(R)                                            | Inversion of (heap-t)
2. (R,C,M,Q, mov r_d, v) -->_0 (R++[r_d--> v],C,M,Q,.)              | mov
*



Case ld_G:

1. |- (R,C,M,Q, ld_G r_d r_s)                                        | Given
2. P;(.;S(G);S(seq(E_d,E_s));S(E_m)) |- ld_G r_d r_s => RT           | Inversion of (heap-t, C-t), substitution, 1
3. S(G)(r_s) = <G,b ref,E_s'>                                        | Inversion of (ld_G-t), 2
4. P|- R : S(G)                                                      | Inversion of (heap-t), 1
5. P;.|- R(r_s) : < G,b ref,E_s'>                                    | Inversion of (reg-file-t), 4, 3
6. .|- R_val(r_s) = E_s'                                             | Inversion of (val-t), 5
```

```
7. E = sel (seq(upd) S(E_m) S(seq(E_d,E_s))) E_s'                    | Inversion of (ld_G-t), 2
8. P |- M : S(E_m)                                                   | Inversion of (heap-t), 1
9. pc_G, pc_B in Dom(R)                                              | Inversion of (heap-t)

subcase a.  find(Q,R_val(r_s)) = ()
10a. R_val(r_s) in Dom(M)                                            | Canonical Forms, 8, Inversion of (heap-t), 5
11a. (R,C,M,Q, ld_G r_d,r_s)                                         | ld_G-mem, assumption, 10a
      -->_0 (R++[r_d--> G M(R_val(r_s))],C,M,Q,.)

subcase b.  find(Q,R_val(r_s)) = (R_val(r_s),n)
12. Exists n'. find(Q,R_val(rs)) = ((R_val(rs),n') or ())            | def of find
13. (R,C,M,Q, ld_G r_d,r_s) -->_0 (R++[r_d--> G n'],C,M,Q,.)         | ld_G-queue, assumption
*




Case ld_B:

Similar to ld_G.
*




Case st_G:

Similar to st_B.
*



Case st_B:

a1. |- (R,C,M,Q, st_B r_d r_s)                                        | Given

1. P |- C                                                            | Inversion of (Sigma-t), a1
2. Forall c=/=Z. C(R_val(pc_c)) = st_b rd rs                         | Inversion of (Sigma-t), a1
3. P;(D;G;seq(E_d,E_s)(E_d',E_s');E_m) |- st_b rd rs                 | Inversion of (C-t), 1, 2, inspection of (st_B-t)
   ==> (D;G++;seq(E_d,E_s);upd E_m E_d' E_s')
4. Exists S. . |- S : D                                              | Inversion of (Sigma-t), a1
5. P;(.;S(G); S(seq(E_d,E_s),(E_d',E_s'));S(E_m)) |- st_B r_d r_s   | substitution lemma, 4, 3
    => (.;S(G)++;S(seq{(E_d,E_s)});S(upd E_m E_d' E_s'))

6. P |- R : S(G)                                                     | Inversion of (Sigma-t), a1
7. S(G)(r_d) = <B,b ref,E_d''>                                       | Inversion of (st_B-t), 5
   S(G)(r_s) = <B,b,E_s''>
8.  P;.|- R(r_s) : < B,b,E_s''>                                      | Inversion of (R-t), 6, 7
    P;.|- R(r_d) : < B,b ref,E_d''>
9. .|- R_val(r_s) = E_s''                                            | Inversion of (val-t), 8
   .|- R_val(r_d) = E_d''

10. .|- S(E_s') = E_s''                                              | Inversion of (st_B-t), 5
    .|- S(E_d') = E_d''

11. P |- Q : S(seq(E_d,E_s),(E_d',E_s'))                             | Inversion of (Sigma-t), a1
12. Q = (seq(n,n'),(n_l,n_l')) where .|- S(E_d')=n_l  and .|- S(E_s')=n_l' | Inversion of (Q-t), 11

13. R_val(r_s) = n_l'  and R_val(r_d) = n_l                          | Exp Eq Transitivity, 9, 10, 11
14. (R,C,M,(seq(n,n'),(n_l,n_l')),st_B  r_d,r_s)                     | st_B-mem, 14
     -->_0^(n_l,n_l') (R++,C,M[n_l --> n_l'],seq{(n,n')},.)



Case bz_G:

1. |- (R,C,M,Q, bz_G r_z,r_d)                                        | Given
2. P;(.;S(G); S(seq(E_d,E_s));S(E_m)) |- bz_G r_z r_d => RT          | Inversion of (heap-t, C-t), substitution, 1
3. S(G)(d) = <G,int,0>                                               | Inversion of (bz_G-t), 2
   S(G)(r_z) = <G,int,E_z>
   S(G)(r_d) = <G,T->void,E_d'>
4. P |- R : S(G)                                                     | Inversion of (heap-t), 1
5. P;.|- R(d) : <G,int,0>                                            | Inversion of (reg-file-t), 4, 3
   P;.|- R(r_z) : < G,int,E_z>
   P;.|- R(r_d) : <G,T->void,E_d'>
6. R_val(d) = 0                                                      | Inversion of (val-t), 5
   r_z in Dom(R), r_d in Dom(R)
7. (R,C,M,Q, bz_G r_z,r_d) -->_0 (R++[d--> R(r_d)],C,M,Q,.) or       | bz_G-taken or bz-untaken, 6
   (R,C,M,Q, bz_G r_z,r_d) -->_0 (R++,C,M,Q,.)
*



Case bz_B:
```

```
1. |- (R,C,M,Q, bz_G r_z,r_d)                                          | Given
2. P;(.;S(G); S(seq(E_d,E_s));S(E_m)) |- bz_B r_z r_d => RT            | Inversion of (heap-t,C-t), substitution, 1
3. S(G)(r_z) = <B,int,E_z>                                             | Inversion of (bz_B-t), 2
   S(G)(r_d) = <B,(D';G';seq(E_d',E_s');E_m')--> void,E_r>
   S(G)(d) = ( E_z'=0 => < G,T'--> void,E_r'> )
   T' = (D';G';seq(E_d',E_s');E_m')
   .|- E_z = E_z'
   .|- E_r = E_r'
4. P |- R : S(G)                                                       | Inversion of (heap-t), 1
5. P;.|- R(d) : E_z'=0 => < G,T'--> void,E_r'>                         | Inversion of (reg-file-t), 4, 3
   P;.|- R(r_z) : < B,int,E_z>
   P;.|- R(r_d) : < B,(D';G';seq{(E_d',E_s')};E_m')--> void,E_r>
6. .|- R_val(r_z) = E_z                                                | Inversion of (val-t), 5
   .|- R_val(r_d) = E_r
7. R_val(d)=0 and .|- E_z'=/=0                                         | Inversion of (cond-t), (cond-t-n0), 5
   or .|- R_val(d): < G,T'--> void,E_r'> and  .|- E_z'=0 and . |- E_r'=/=0
8. R_val(d)=0 and .|- E_z'=/=0                                         | Inversion of (val-t), 7
   or .|- R_val(d) = E_r' and  .|- E_z'=0 and . |- E_r'=/=0

Case a: R_val(d) = 0 and .|- E_z'=/=0
9a. . |- R_val(r_z) = E_z                                              | Inversion of (val-t), 5
10a. . |- R_val(r_z) = E_z'                                            | Exp Eq Transitivity, 9a, 3
11a. R_val(r_z) =/= 0                                                  | 10a, assumption
10a. (R,C,M,Q,bz_B r_z,r_d) -->_0 (R++,C,M,Q,.)                        | bz-untaken, assumption, 11a

Case b: .|-R_val(d) = E_r' and  .|- E_z'=0 and . |- E_r'=/=0
9b. . |- R_val(R_z) = E_z                                              | Inversion of (val-t), 5
10b. . |- R_val(r_z) = E_z'                                            | Exp Eq Transitivity, 9b, 3
11b. . |- R_val(r_z) = 0                                               | Exp Eq Transitivity, 10b, assumption
12b. . |- R_val(r_d) = R_val(d)                                        | Exp Eq Transitivity, 3, 6, assumption
13b. R_val(r_z) = 0                                                    | 11b, Inversion of (E-eq), def of [[n]]
14b. R_val(r_d) = R_val(d)                                             | 12b, Inversion of (E-eq), def of [[n]]
15b. (R,C,M,Q, bz_B r_z,r_d)                                           | bz_B-taken 13b, 14b
     -->_0 (R[pc_G--> R(d)][pc_B--> R(r_d)][d--> G .],C,M,Q,.)
*



Case jmp_G:

Similar to bz_G.
*



Case jmp_B:

Similar to bz_B.
*
```

Progress Part 2

```
2. If |-c (R,C,M,Q,ir) then (R,C,M,Q,ir) -->_0^s  S


Proof by case analysis on ir.


Case .:

1. |-c (R,C,M,Q,.)                                                    | Given
2. P |-c R : S(G)                                                     | Inversion of (heap-t), 1
3. S(G)(pc_G) =< <G,int,E_G>                                          | Inversion of (reg-file-t), 2
   S(G)(pc_B) =< <B,int,E_B>
4. P;.|-c R(pc_G) : <G,int,E_G>                                       | 3, Inversion of (reg-file-t), 2, Subtyping Lemma
   P;.|-c R(pc_B) : <B,int,E_B>

By 4, and inversion of typing, one of the following cases holds:
case a:
5a. P;.|-c R(pc_G) : <G,int,E_G>                                      | by (val-t)
    P;.|-c R(pc_B) : <B,int,E_B>                                      | by (val-t)
6a. Proof is the same as case "." in Progress 1.
    Rule fetch applies.
case b:
7b. P;.|-c R(pc_G) : <G,int,E_G>
    P;.|-c R(pc_B) : <B,int,E_B>                                      | one by (val-zap-t) not by (val-t), the other by val-t
8b. either .|- R(pc_G) not= E_G                                       | by 7b, inversion of typing
         or .|- R(pc_B) not= E_B
9b. Proof proceeds similarly as case "." in Progress 1, except we conclude R(pc_G) not= R(pc_B).
    Hence, rule fetch-fail applies.
*



Case op2r:

Similar to case op2r in Progress 1.
Rule op2r applies.
*



Case op1r:

Similar to case op1r in Progress 1.
Rule op1r applies.
*



Case mov:

Similar to case mov in Progress 1.
Rule mov applies.
*



Case ld_G:

1. |-c (R,C,M,Q, ld_G r_d r_s)                                         | Given
2. P;(.;S(G);S(seq(E_d,E_s));S(E_m)) |-c ld_G r_d r_s => RT            | Inversion of (heap-t, C-t), substitution, 1
3. S(G)(r_s) = <G,b ref,E_s'>                                          | Inversion of (ld_G-t), 2
4. P|-c R : S(G)                                                       | Inversion of (heap-t), 1
5. P;.|-c R(r_s) : < G,b ref,E_s'>                                     | Inversion of (reg-file-t), 4, 3

By 5, and inversion of typing, one of the following cases holds:
case a:
6a. P;.|-c R(r_s) : < G,b ref,E_s'>                                    | by (val-t)
7a. Proof is the same as case ld_G in Progress 1.
    Rule ld_G-mem or ld_G-queue applies.
case b:
```

```
8b. P;.|-c R(r_s) : < G,b ref,E_s'>                                | not by (val-t) but by (val-zap-t)
9b. Proof is the same as case ld_G in Progress 1 except
    on line 10a of that proof we cannot conclude R_val(r_s) in Dom(M).
    If R_val(r_s) notin Dom(M), either ld_G-rand or ld_G-fail may execute.
    Otherwise ld_G-mem or ld_G-queue applies as before.
*




Case ld_B:

Similar to case ld_G.
Rules ld_B-mem or ld_B-fail or ld_B-rand apply.
*




Case st_G:

Similar to case st_G (or op2r) in Progress 1.
Rule st-G-queue always applies.
*




Case st_B: - DAVES VERSION. UNNECESSARILY COMPLICATED? SEE FOLLOWING VERSION

1. |-c (R,C,M,Q, st_B r_d r_s)                                    | Given
2. P;(.;S(G); S(seq(E_d,E_s),(E_d',E_s'));S(E_m)) |- st_B r_d r_s   | Inversion of (heap-t,C-t), substitution, 1
    => (.;S(G)++;S(seq{(E_d,E_s)});S(upd E_m E_d' E_s'))
3. S(G)(r_d) = <B,b ref,E_d''>                                    | Inversion of (st_B-t), 2
   S(G)(r_s) = <B,b,E_s''>
4. P|-c R : S(G)                                                  | Inversion of (heap-t), 1
5. P;.|-c R(r_s) : < B,b,E_s''>                                   | Inversion of (reg-file-t), 4, 3
   P;.|-c R(r_d) : < B,b ref,E_d''>

Subcase A:  Assume c = B

  By 5, and inversion of typing, one of the following cases holds:
  case a:
  6a. P;.|-c R(r_s) : < B,b,E_s''>                                 | both by (val-t)
      P;.|-c R(r_d) : < B,b ref,E_d''>
  7a. Proof is the same as case st_B in Progress 1.
      Rule st-B-mem applies.
  case b:
  8b. P;.|-c R(r_s) : < B,b,E_s''>                                 | at least one not by (val-t) but by (val-zap-t)
      P;.|-c R(r_d) : < B,b ref,E_d''>
  9b. either .|- R_val(r_s) not= E_s''                             | by 8b.
          or .|- R_val(r_d) not= E_d''
  10b. .|- S(E_s') = E_s''                                         | Inversion of (st_B-t), 2
       .|- S(E_d') = E_d''
  11b. P |-c Q : S(seq(E_d,E_s),(E_d',E_s'))                       | Inversion of (heap-t), 1
  12b. Q = (seq(n,n'),(n_l,n_l')) where .|- S(E_d')=n_l  and .|- S(E_s')=n_l' | By Queue Lemma, 11b
  13b. either R_val(r_s) not= n_l'  or R_val(r_d) not= n_l         | 9b and Exp Eq Transitivity of 10b, 12b
  14b. Rule st_B-mem-fail applies.                                 | by 13b.
  *

Subcase B:  Assume c = G

  15. .|- R_val(r_s) = E_s''                                       | Inversion of (val-t), 5
      .|- R_val(r_d) = E_d''
  16. .|- S(E_s') = E_s''                                          | Inversion of (st_B-t), 2
      .|- S(E_d') = E_d''
  17. P |-c Q : S(seq(E_d,E_s),(E_d',E_s'))                        | Inversion of (heap-t), 1
  18. Q = (seq(n,n'),(n_l,n_l'))                                   | By Queue Lemma, 17

  By 17, and (inductive) inversion, one of the following cases holds:
  case c:
  18c. .|- S(E_d')=n_l  and .|- S(E_s')=n_l'                        | Rule (Q-t) used to type (n_1,n_l') in 17
  19c. Proof is the same as case st_B in Progress 1.
       Rule st-B-mem applies.
  case d:
  20d.  .|- S(E_d') not= n_l  or .|- S(E_s') not= n_l'             | Rule (Q-zap-t) not (Q-t) used to type (n_1,n_l') in 17
  21d. Rule st_B-mem-fail applies.                                 | By 20d and transitivity of 15, 16
  *
End.
```

```
Case st_B: - FJP
a1. |- (R,C,M,Q, st_B r_d r_s)                                    | Given

1. P |- Q : S(seq(E_d,E_s),(E_d',E_s'))                           | Inversion of (Sigma-t), a1
2. Q = (seq(n,n'),(n_l,n_l'))                                     | Queue Lemma, 1

Subcase 1: end of queue matches r_d / r_s
a2. Rval(r_d) = n_l and Rval(r_s) = n_l'

3. (R,C,M,(seq(n,n'),(n_l,n_l')),st_B  r_d,r_s)                   | st_B-mem, a2
     -->_0^(n_l,n_l') (R++,C,M[n_l --> n_l'],seq{(n,n')},.)
subcase complete.

Subcase 2: end of queue does not match r_d / r_s
a2. Rval(r_d) =/= n_l or Rval(r_s) =/= n_l'
3. (R,C,M,Q,st_B  r_d,r_s) --> fault                              | st_B-mem-fail, 2, a2
subcase complete.
```

```
Case bz_G:

1. |-c (R,C,M,Q, bz_G r_z,r_d)                                     | Given
2. P;(.;S(G); S(seq(E_d,E_s));S(E_m)) |- bz_G r_z r_d => RT        | Inversion of (heap-t, C-t), substitution, 1
3. S(G)(d) = <G,int,0>                                             | Inversion of (bz_G-t), 2
   S(G)(r_z) = <G,int,E_z>
   S(G)(r_d) = <G,T->void,E_d'>
4. P |-c R : S(G)                                                  | Inversion of (heap-t), 1
5. P;.|-c R(d) : <G,int,0>                                         | Inversion of (reg-file-t), 4, 3
   P;.|-c R(r_z) : < G,int,E_z>
   P;.|-c R(r_d) : <G,T->void,E_d'>
6. d, r_z, r_d in Dom(R)                                           | By 5

By 6, and inversion, one of the following cases holds

case a:  R_val(d) not= 0
7a. R_val(d) not= 0                                                | assumed in this case
8a. either bz-untaken-fail or bz_G-taken-fail applies              | By 7a, 6
*a

case b:  R_val(d) = 0
9b. R_val(d) = 0                                                   | assumed in this case
10b. either bz-untaken or bz_G-taken applies                       | by 9b, 6
*b

End.
```

```
Case bz_B:

1. |- (R,C,M,Q, bz_G r_z,r_d)                                     | Given
2. P;(.;S(G); S(seq(E_d,E_s));S(E_m)) |- bz_B r_z r_d => RT       | Inversion of (heap-t,C-t), substitution, 1
3. S(G)(r_z) = <B,int,E_z>                                        | Inversion of (bz_B-t), 2
   S(G)(r_d) = <B,(D';G';seq(E_d',E_s');E_m')--> void,E_r>
   S(G)(d) = ( E_z'=0 => < G,T'--> void,E_r'> )
   T' = (D';G';seq(E_d',E_s');E_m')
   .|- E_z = E_z'
   .|- E_r = E_r'
4. P |- R : S(G)                                                  | Inversion of (heap-t), 1
5. P;.|- R(d) : E_z'=0 => < G,T'--> void,E_r'>                    | Inversion of (reg-file-t), 4, 3
   P;.|- R(r_z) : < G,int,E_z>
   P;.|- R(r_d) : < B,(D';G';seq{(E_d',E_s')};E_m')--> void,E_r>
6. d, r_z, r_d in Dom(R)                                          | By 5
```

```
By 6, one of the following cases holds

case a: Rval(r_z) not= 0
7a. Rval(r_z) not= 0                                                | assumed in this case
8a. Rule bz-untaken or bz-untaken-fail applies                      | by 6, 7a
*a

case b: Rval(r_z) = 0
9b. Rval(r_z) = 0                                                   | assumed in this case
    One of the following subcases holds
      Subcase ba: Rval(d) not= 0  and Rval(rd) = Rval(d)
        Rule bz_B-taken applies                                     | By Subcase ba condition, 6, 9b
      Subcase bb: Rval(d) = 0  or Rval(rd) not= Rval(d)
        Rule bz_B-taken-fail applies                                | By Subcase bb condition, 6, 9b
*b

End.



Case jmp_G:

Similar but simpler than case bz_G.
Rule jmp_G or jmp_G-fail applies.
*


Case jmp_B:

Similar but simpler than case bz_B.
Rule jmp_B or jmp_B-fail applies.
*
```

Preservation Part 1

```
1. If |-Z (R,C,M,Q,ir)
   and (R,C,M,Q,ir)  -->_0^s  (R',C',M',Q',ir')
   then |-Z (R',C',M',Q',ir')



Proof by induction on the structure of the derivation of (R,C,M,Q,ir)  -->_0^s  (R',C',M',Q',ir').


CASE fetch:

(p1) R_val(pc_G) = R_val(pc_B)
(p2) R_val(pc_G) in Dom(C)
 ---------------------------------------- (fetch)
(R,C,M,Q,.) -->_0 (R,C,M,Q,C(R_val(pc_G)))


0. |-Z (R,C,M,Q,.)                                                     | Given
1. Dom(P) = Dom(C) union Dom(M)                                        | Inversion of (heap-t), 0
2. Z=/=G ==> Dom(Q) subseteq Dom(M)                                    | Inversion of (heap-t), 0
3. P |- C                                                              | Inversion of (heap-t), 0
4. <deleted>
5. Forall c=/=Z. P(R_val(pc_c)) = (D;G;seq(E_d,E_s);E_m)-->void        | Inversion of (heap-t), 0
6. Exists S. .|- S : D                                                 | Inversion of (heap-t), 0
7. P |- M : S(E_m)                                                     | Inversion of (heap-t), 0
8. P |-Z Q : S(seq(E_d,E_m))                                           | Inversion of (heap-t), 0
9. P |-Z R : S(G)                                                      | Inversion of (heap-t), 0

4'. Forall c=/=Z. C(R_val(pc_c)) = C(R_val(pc_G))                      | (p1),(p2)
10.  |-Z (R,C,M,Q,C(R_val(pc_G)))                                      | (heap-t), 1,2,3,4',5,6,8,9
*




CASE fetch-fail:

Rval(pc_G) =/= Rval(pc_B)
--------------------------(fetch-fail)
(R,C,M,Q,.)  -->_0  fault

does not apply (fails second assumption)
*




CASE op2r:

R2 = R++[ r_d -> R_col(r_t) (R_val(r_s) op R_val(r_t)) ]
----------------------------------------------------------(op2r)
(R,C,M,Q, op r_d, r_s, r_t )  -->_0  (R2,C,M,Q,.)


0. |-Z (R,C,M,Q,op r_d, r_s, r_t)                                          | Given
1. Dom(P) = Dom(C) union Dom(M)                                            | Inversion of (heap-t), 0
2. Z=/=G ==> Dom(Q) subseteq Dom(M)                                        | Inversion of (heap-t), 0
3. P |- C                                                                  | Inversion of (heap-t), 0
4. Forall c=/=Z. C(R_val(pc_c)) = op r_d, r_s, r_t                         | Inversion of (heap-t), 0
5. Forall c=/=Z. P(R_val(pc_c)) = (D;G;seq(E_d,E_s);E_m)-->void            | Inversion of (heap-t), 0
6. Exists S. . |- S : D                                                    | Inversion of (heap-t), 0
7. P |- M : S(E_m)                                                         | Inversion of (heap-t), 0
8. P |-Z Q : S(seq(E_d,E_m))                                               | Inversion of (heap-t), 0
9. P |-Z R : S(G)                                                          | Inversion of (heap-t), 0

Let c' = R_col(r_t)
Let R2 = R++[r_d --> c' (R_val(r_s) op R_val(r_t))]
Let G2 = G++[r_d --> <c',int, E_s' op E_t'> ]

10. P;(D;G;seq(E_d,E_s);E_m) |- op r_s,r_t,r_d ==> RT2                     | Inversion of (C-t), 3, 4
```

```
11. RT2 = (D;G2;seq(E_d,E_s);E_m)                                       | Inspection of (op2r-t), def of G2
12. Forall c=/=Z. P(R_val(pc_c)+1) = RT2 --> void                       | Inversion of (C-t), 3, 4, 11
5'. Forall c=/=Z. P(R2_val(pc_c)) = (D;G2;seq(E_d,E_s);E_m) --> void    | def of ++, def of R2, 11, 12

13. P;(D;G;seq(E_d,E_s);E_m)|- op r_s,r_t,r_d ==>(D;G2;seq(E_d,E_s);E_m)| 10, 11
14. P;(.;S(G);S(seq(E_d,E_s));S(E_m))|- op r_s,r_t,r_d                  | substitution, 6, 13
      ==> (.;S(G2);S(seq(E_d,E_s));S(E_m))
15. S(G)(r_s) = <c',int,E_s'>                                           | Inversion of (op2r-t), 13
16. S(G)(r_t) = <c',int,E_t'>                                           | Inversion of (op2r-t), 13

17. Forall a. P;. |-Z R(a) : S(G)(a)                                    | Inversion of (reg-file-t), 9
18. S(G)(pc_G) =< <G,int,E_G>    and  S(G)(pc_B)   =< <B,int,E_B>       | Inversion of (reg-file-t), 9
19. S(G)++(pc_G)=< <G,int,E_G+1> and  S(G)++(pc_B) =< <B,int,E_B+1>     | 18, def G++
19a. . |- E_G = E_B                                                     | Inversion of (reg-file-t), 9
19b. [[E_G]] = [[E_B]]                                                  | Inversion of 19a, def of [[]]
19c. [[E_G]] + [[1]] = [[E_B]] + [[1]]                                  | 19b, def of [[]]
19d. [[E_G + 1]] = [[E_B + 1]]                                          | 19c, def of [[]]
19e. . |- E_G + 1 = E_B + 1                                             | 19d, (E-eq)
20. P |-Z R++ : S(G++)                                                  | (reg-file-t), def of R++, 19, 19e
21. P;. |-Z R(r_s) : <c',int,E_s'>  and P;. |-Z R(r_t) : <c',int,E_t'>  | Inversion of (reg-file-t), 9, 15, 16

SUBCASE a: Z =/= c'
21a. . |- E_s' = R(r_s)  and . |- E_t' = R(r_t)                         | Cannonical Forms 2, 7, 3, 20a, assumption
22a. [[E_s']] = [[R(r_s)]]  and [[E_t']] = [[R(r_t)]]                   | Inversion on (E-eq), 21a
23a. [[R_val(r_s)]] op [[R_val(r_t)]]  =  [[E_s']] op [[E_t']]          | Subst of Eq for Eq, 22a
24a. [[R_val(r_s) op R_val(r_t)]] = [[E_s' op E_t']]                    | def of [[]], 23a
25a. . |- E_s' : kint and . |- E_t' : kint                              | Inversion on (E-eq), 21a
26a. . |- (E_s' op E_t') : kint                                         | (E-op-t), 25a
27a. . |- (R_val(r_s) op R_val(r_t)) : kint                             | (E-int-t)
28a. . |- (R_val(r_s) op R_val(r_t)) = (E_s' op E_t')                   | (E-eq), 27a, 28a, 24a
29a. . |- (R_val(r_s) op R_val(r_t)) : int                              | (int-t)
30a. P;.|-Z c' (R_val(r_s) op R_val(r_t)) : <c',int, E_s' op E_t'>      | (val-t), 29a, 28a

SUBCASE b: Z = c'
20b. . |- E_s' : kint and . |- E_t' : kint                              | Int Kinding Lemma, 21
21b. . |- E_s' op E_t' : kint                                           | (E-op-t), 20b
22b. P;.|-Z c' (R_val(r_s) op R_val(r_t)) : <c',int, E_s' op E_t'>      | (val-zap-t), assumption, 21b

MERGE:
31. P;.|-Z R2(r_d) : S(G2(r_d))                                         | 30a/22b, def of R2, def of G2,
9'. P |-Z R2 : S(G2)                                                    | def of R2, def of G2, 20, 31

25. |-Z (R2,C,M,Q,.)                                                    | (heap-t), 1,2,3,ir=.,5',6,7,8,9'
*




CASE op1r:

 R2 = R++[ rd -> c' (R_val(rs) op n) ]
----------------------------------------------------------(op1r)
(R,C,M,Q, op rd, rs, c' n )  -->_0  (R2,C,M,Q,.)

Similar to op2r.
*




CASE mov:

-----------------------------------------------------------(mov)
(R,C,M,Q, mv rd, v )  -->_0  (R++[rd -> v] ,C,M,Q,.)

Similar to op2r.
*




CASE ld_G-queue:

(p1) find(Q,R_val(r_s)) = (R_val(r_s),n)
-----------------------------------------------------------------(ld_G-queue)
(R,C,M,Q, ld_G r_d, r_s )  -->_0  (R++[r_d -> G n] ,C,M,Q,.)


0. |-Z (R,C,M,Q,ld_G r_d, r_s)                                          | Given
1. Dom(P) = Dom(C) union Dom(M)                                         | Inversion of (heap-t), 0
2. Z=/=G ==> Dom(Q) subseteq Dom(M)                                     | Inversion of (heap-t), 0
3. P |- C                                                               | Inversion of (heap-t), 0
4. Forall c=/=Z. C(R_val(pc_c)) = ld_G r_d, r_s                         | Inversion of (heap-t), 0
5. Forall c=/=Z. P(R_val(pc_c)) = (D;G;seq(E_d,E_s);E_m)-->void         | Inversion of (heap-t), 0
```

```
6. Exists S. . |- S : D                                            | Inversion of (heap-t), 0
7. P |- M : S(E_m)                                                 | Inversion of (heap-t), 0
8. P |-Z Q : S(seq(E_d,E_m))                                       | Inversion of (heap-t), 0
9. P |-Z R : S(G)                                                  | Inversion of (heap-t), 0

Let n = [[E]]
Let R2 = R++[r_d --> G n]
Let G2 = G++[r_d --> <G,b,E> ]

10. P;(D;G;seq(E_d,E_s);E_m) |- ld_G r_d, r_s ==> RT2              | Inversion of (C-t), 3, 4
11. RT2 = (D;G2;seq(E_d,E_s);E_m)                                  | Inspection of (ld_G-t), def of G2
12. Forall c=/=Z. P(R_val(pc_c)+1) = RT2 -> void                   | Inversion of (C-t), 3, 4, 11
5'. Forall c=/=Z. P(R2_val(pc_c)) = (D;G2;seq(E_d,E_s);E_m) --> void    | def of ++, def of R2, 12

13. P;(.;S(G);S(seq(E_d,E_s));S(E_m))|- ld_G r_d, r_s              | 10, 11, substitution, 6
      ==>(.;S(G2);S(seq(E_d,E_s));S(E_m))
14. S(G)(r_s) = <G,b ref,E_s'>                                     | Inversion of (ld_G-t), 13
15. E = sel (sequpd S(E_m) S(seq(E_d,E_s))) E_s'                   | Inversion of (ld_G-t), 13


16. . |- S(E_m) : kmem                                             | Exp Evaluation Lemma, Inversion on (M-t), 7,
17. . |- S(seq(E_d,E_s)) : seq(kint,kint)                          | Inversion on (Q-t) and (Q-zap-t)
18. . | E_s' : kint                                                | Inversion of (reg-file-t), 9, Int Kinding Lemma
19. . |- E : kint                                                  | By applying sequences of (E-upd-t) and (E-sel-t), 16, 17, 18


SUBCASE a: Z = G

20a. P;.|-Z G n : <G,b,E>                                          | (val-zap-t), assumption, 19

SUBCASE b: Z =/= G
20b. P;. |-Z R(r_s) : S(G)(r_s)                                    | Inversion of (reg-file-t), 7
21b. P;. |-Z R(r_s) : <G,b ref, E_s'>                              | 15b, 11
22b. P |-  R_val(r_s) : b ref                                      | Inversion on (val-t), assumption, 21b
23b. . |-  E_s' = R_val(r_s)                                       | Inversion on (val-t), assumption, 21b
24b. Exists n. [[E]] = n                                           | Exp Evaluation Lemma, 19
25b. P |- n : b                                | (int-t)
26b. P;. |-Z G n : <G,b,E>                                         | (val-t), 24b, 25b

MERGE:
22. P;.|-Z R2(r_d) : S(G2)(r_d)                                    | 20a/26b, def of R2, def of G2
9'. P |-Z R2 : S(G2)                                               | (reg-file-t), 9, def of R2, def of G2, def of ++, 22

23. |-Z (R2,C,M,Q,.)                                               | (heap-t), 1, 2, 3, ir=., 5', 6, 7, 8, 9'
*




CASE ld_G-mem:

(p1) find(Q,R_val(r_s)) = ()
(p2) R_val(r_s) in Dom(M)
(s1) R2 = R++[r_d -> G M(R_val(r_s)) ]
-----------------------------------------------(ld_G-mem)
(R,C,M,Q, ld_G rd, rs )  -->_0  (R2,C,M,Q,.)

0. |-Z (R,C,M,Q,ld_G r_d, r_s)                                     | Given
1. Dom(P) = Dom(C) union Dom(M)                                    | Inversion of (heap-t), 0
2. Z=/=G ==> Dom(Q) subseteq Dom(M)                                | Inversion of (heap-t), 0
3. P |- C                                                          | Inversion of (heap-t), 0
4. Forall c=/=Z. C(R_val(pc_c)) = ld_G r_d, r_s                    | Inversion of (heap-t), 0
5. Forall c=/=Z. P(R_val(pc_c)) = (D;G;seq(E_d,E_s);E_m)-->void    | Inversion of (heap-t), 0
6. Exists S. . |- S : D                                            | Inversion of (heap-t), 0
7. P |- M : S(E_m)                                                 | Inversion of (heap-t), 0
8. P |-Z Q : S(seq(E_d,E_m))                                       | Inversion of (heap-t), 0
9. P |-Z R : S(G)                                                  | Inversion of (heap-t), 0

Let R2 = R++[r_d --> G M(R_val(r_s)) ]
Let G2 = G++[r_d --> <G,b,E> ]

10. P;(D;G;seq(E_d,E_s);E_m) |- ld_G r_d, r_s ==> RT2              | Inversion of (C-t), 3, 4
11. RT2 = (D;G2;seq(E_d,E_s);E_m)                                  | Inspection of (ld_G-t), def of G2, 10
12. Forall c=/=Z. P(R_val(pc_c)+1) = RT2 -> void                   | Inversion of (C-t), 3, 4, 11
5'. Forall c=/=Z. P(R2_val(pc_c)) = (D;G2;seq(E_d,E_s);E_m) --> void    | def of ++, def of R2, 12

13. P;(.;S(G);S(seq(E_d,E_s));S(E_m))|- ld_G r_d, r_s              | 10, 11, substitution, 6
      ==>(.;S(G2);S(seq(E_d,E_s));S(E_m))
14. S(G)(r_s) = <G,b ref,E_s'>                                     | Inversion of (ld_G-t), 13
15. E = sel (sequpd S(E_m) S(seq(E_d,E_s))) E_s'                   | Inversion of (ld_G-t), 13

16. . |- S(E_m) : kmem                                             | Inversion on (M-t), 7
```

```
17. . |- S(seq(E_d,E_s)) : seq(kint,kint)                               | Inversion on (Q-t) and (Q-zap-t)
18. . | E_s' : kint                                                     | Inversion of (reg-file-t), 9, Int Kinding Lemma
19. . |- E : kint                                                       | By applying sequences of (E-upd-t) and (E-sel-t), 16, 17, 18


SUBCASE a: Z = G
20a. P;.|-Z G M(R_val(r_s)) : <G,b,E>                                   | (val-zap-t), assumption. 19


SUBCASE b: Z =/= G
20b. P;. |- R(r_s) : <G,b ref,E_s'>                                     | Inversion of (reg-file-t), 9, 14
21b. P |- R_val(r_s) : b ref                                            | Inversion of (val-t), assumption, 20b
23b. P |- M(R_val(r_s)) : b                                             | Inversion of (M-t), 7, (p2), 21b

24b. . |- E_s' = R_val(r_s)                                             | Inversion of (val-t), assumption, 20b
25b. for k:1...length(Q). . |- Edk =/= R_rval(r_s)                      | Find Lemma, (p1), 8
26b. E = sel S(E_m) E_s'                                                | recursive apps of Irrelevant Update Lemma, 15, 25b
27b. M = [[E_m]]                                                        | Inversion of (M-t)
28b. [[sel S(E_m) E_s']] = M([[E_s']])                                  | def of [[]], 27b
29b. [[E_s']] = R_val(r_s)                                              | Inversion of (E-eq), 24b
30b. [[E]] = M(R_val(rs))                                               | 26b, 28b, 29b
31b. . |- E = M(R_val(rs))                                              | (E-eq), 19, 30b

31b. P |-Z G M(R_val(r)s)) : <G, b, E>                                  | (val-t), 23b, 31b


MERGE:
32. P;.|-Z R2(r_d) : S(G2)(r_d)                                         | 20a/31b, def of R2, def of G2
9'. P |-Z R2 : S(G2)                                                    | (reg-file-t), 9, def of R2, def of G2, def of ++, 32

23. |-Z (R2,C,M,Q,.)                                                    | (heap-t), 1, 2, 3, ir=., 5', 6, 7, 8, 9'
*





CASE ld_G-rand:

find(Q,R_val(r_s)) = ()
R_val(r_s) not in Dom(M)
R2 = R++[r_d -> G n ]
-------------------------------------------------(ld_G-rand)
(R,C,M,Q, ld_G r_d, r_s )  -->_0  (R2,C,M,Q,.)

0. |-Z (R,C,M,Q,ld_G r_d, r_s)                                          | Given
1. Dom(P) = Dom(C) union Dom(M)                                         | Inversion of (heap-t), 0
2. Z=/=G ==> Dom(Q) subseteq Dom(M)                                     | Inversion of (heap-t), 0
3. P |- C                                                               | Inversion of (heap-t), 0
4. Forall c=/=Z. C(R_val(pc_c)) = ld_G r_d, r_s                         | Inversion of (heap-t), 0
5. Forall c=/=Z. P(R_val(pc_c)) = (D;G;seq(E_d,E_s);E_m)-->void         | Inversion of (heap-t), 0
6. Exists S. . |- S : D                                                 | Inversion of (heap-t), 0
7. P |- M : S(E_m)                                                      | Inversion of (heap-t), 0
8. P |-Z Q : S(seq(E_d,E_m))                                            | Inversion of (heap-t), 0
9. P |-Z R : S(G)                                                       | Inversion of (heap-t), 0

Let R2 = R++[r_d --> G n]
Let G2 = G++[r_d --> < G,int,E_n>]                                      | where E_n is just n

10. P;(D;G;seq(E_d,E_s);E_m) |- ld_G r_d, r_s ==> RT2                   | Inversion of (C-t), 3, 4
11. RT2 = (D;G2;seq(E_d,E_s);E_m)                                       | Inspection of (ld_G-t), def of G2. 10
12. Forall c=/=Z. P(R_val(pc_c)+1) = RT2 --> void                       | Inversion of (C-t), 3, 4, 11
5'. Forall c=/=Z. P(R2_val(pc_c)) = (D;G2;seq(E_d,E_s);E_m) --> void    | def of ++, def of R2, 11, 12

13. P |- n : int                                                        | (int-t)
14. . |- E_n = n                                                        | def of G2
15. P;. |-Z G n : <G,int,E_n>                                           | (val-t), 13, 14
16. S(E_n) = E_n                                                        | def of G2
9'. P |-Z R2 : S(G2)                                                    | (reg-file-t), 7, def of R2, def of G2, def of ++, 15, 16

17. |-Z (R2,C,M,Q,.)                                                    | (heap-t), 1, 2, 3, ir=., 5', 6, 7, 8, 9'
*





CASE ld_G-fail:

find(Q,R_val(r_s)) = ()
R_val(r_s) not in Dom(M)
-------------------------------------(ld_G-fail)
(R,C,M,Q, ld_G r_d, r_s)  -->_0  fault
```

```
does not apply (fails second assumption)
*







CASE ld_B-mem:

R_val(r_s) in Dom(M)
R' = R1++[ rd -> B M(R_val(r_s)) ]
-------------------------------------------------(ld_B-mem)
(R,C,M,Q, ld_B rd, rs )  -->_0  (R',C,M,Q,.)


Similar to ld_G-mem.
*







CASE ld_B-rand:

R_val(r_s) not in Dom(M)
R' = R++[ r_d -> B n ]
---------------------------------------------------(ld_B-rand)
(R,C,M,Q, ld_B r_d, r_s )  -->_0  (R',C,M,Q,.)

Similar to ld_G-rand.
*







CASE ld_B-fail:

Rval(r_s) not in Dom(M)
-------------------------------------------(ld_B-fail)
(R,C,M,Q, ld_B r_d, r_s )  -->_0  fault

does not apply (fails second assumption)
*







CASE st_G-queue:

 Q2 = ( (R_val(r_d), R_val(r_s)), Q )
---------------------------------------------------(st_G-queue)
(R,C,M,Q, st_G r_d, r_s )  -->_0  (R++,C,M,Q2,.)



0. |-Z (R,C,M,Q,st_G r_d, r_s)                                        | Given
1. Dom(P) = Dom(C) union Dom(M)                                       | Inversion of (heap-t), 0
2. Z=/=G ==> Dom(Q) subseteq Dom(M)                                   | Inversion of (heap-t), 0
3. P |- C                                                             | Inversion of (heap-t), 0
4. Forall c=/=Z. C(R_val(pc_c)) = st_G rd, rs                         | Inversion of (heap-t), 0
5. Forall c=/=Z. P(R_val(pc_c)) = (D;G;seq(E_d,E_s);E_m)-->void       | Inversion of (heap-t), 0
6. Exists S. . |- S : D                                               | Inversion of (heap-t), 0
7. P |- M : S(E_m)                                                    | Inversion of (heap-t), 0
8. P |-Z Q : S(seq(E_d,E_m))                                          | Inversion of (heap-t), 0
9. P |-Z R : S(G)                                                     | Inversion of (heap-t), 0

10. P;(D;G;seq(E_d,E_s);E_m) |- st_G r_d, r_s ==> RT2                 | Inversion of (C-t), 3, 4
11. RT2 = (D;G++;((E_d',E_s'),(seq(E_d,E_s)));E_m)                    | Inspection of (st_G-t), def of G2, 11
12. Forall c=/=Z. P(R_val(pc_c)+1) = RT2 --> void                     | Inversion of (C-t), 3, 4, 11
5'. Forall c=/=Z. P(R2_val(pc_c))                                     | def of ++, def of R2, 11, 12
    = (D;G++;((E_d',E_s'),(seq(E_d,E_s)));E_m) --> void

9'. P |- R++ : S(G++)                                                 | 9, def of ++

13. P;(.;S(G);S(seq(E_d,E_s));S(E_m))|- st_G r_d, r_s                 | 10, 11, substitution, 6
        ==>(.;S(G++);((E_d',E_s'),S(seq(E_d,E_s)));S(E_m))
14. S(G)(r_d)= <G,b ref,E_d'>                                         | Inversion of (st_G-t), 13
15. S(G)(r_s)= <G,b,E_s'>                                             | Inversion of (st_G-t), 13

Let Q2 = ( (R_val(r_d), R_val(r_s)), Q )
```

```
SUBCASE a: Z = G
16a. . |- E_d':kint  and . |- E_s':kint                                 | Int Kinding Lemma, 14, 15, Inversion of (reg-file-t), 9
17a. P |- Q2 : ((E_d',E_s'),S(seq(E_d,E_s)))                            | (Q-zap-t), assumption, 8, 16a

SUBCASE b: Z =/= G
16b. . |- E_d' = R_val(r_d) and P |- R_val(r_d) : b ref                 | Inversion of (val-t), assumption, 14
17b. . |- E_s' = R_val(r_s) and P |- R_val(r_s) : b                     | Inversion of (val-t), assumption, 15
18b. P |- Q2 : ((E_d',E_s'),S(seq(E_d,E_s)))                            | (Q-t), assumption, 8, 16b, 17b
19b. P |- R_val(r_d) : <G,b ref, E_d'>                                  | Inversion of (reg-file-t), 9, 14
20b. E_d' in Dom(M)                                                     | Cannonical Forms, 1, 7, 3,19b
21b. Dom(Q2) subseteq Dom(M)                                            | def of Q2, 20b, 1

MERGE:
8'. P |- Q2 : ((E_d',E_s'),S(seq(E_d,E_s)))                             | 17a/18b
2'. Z =/=G ==> Dom(Q2) subseteq Dom(M)                                  | 21b

22. |-Z (R++,C,M,Q2,.)                                                  | (heap-t), 1,2',3,ir=.,5',6,7,8',9'
*




CASE st_B-mem:

(p1) R_val(r_d) = n1
(p2) R_val(r_s) = n1'
------------------------------------------------------------------(st_B-mem)
(R, C, M, ((seq(ns1,ns1'),(n1,n1')), st_B r_d, r_s)
      -->_0^(n1,n1')  (R++, C, M[n1->n1'], seq(ns1,ns1') , .)


0. |-Z (R,C,M,((seq(ns1,ns1'),(n1,n1')),st_G r_d, r_s)                  | Given
1. Dom(P) = Dom(C) union Dom(M)                                         | Inversion of (heap-t), 0
2. Z=/=G ==> Dom(((seq(ns1,ns1'),(n1,n1'))) subseteq Dom(M)             | Inversion of (heap-t), 0
3. P |- C                                                               | Inversion of (heap-t), 0
4. Forall c=/=Z. C(R_val(pc_c)) = st_G rd, rs                           | Inversion of (heap-t), 0
5. Forall c=/=Z.P(R_val(pc_c))=(D;G;(seq(E_d,E_s),(E_d',E_s'));E_m)-->void | Inversion of (heap-t), 0
6. Exists S. . |- S : D                                                 | Inversion of (heap-t), 0
7. P |- M : S(E_m)                                                      | Inversion of (heap-t), 0
8. P |-Z ((seq(ns1,ns1'),(n1,n1')) : S(seq(E_d,E_m),(E_d',E_s'))        | Inversion of (heap-t), 0
9. P |-Z R : S(G)                                                       | Inversion of (heap-t), 0

10. P;(D;G;(seq(E_d,E_s),(E_d',E_s'));E_m) |- st_B r_d, r_s ==> T2      | Inversion of (C-t), 3, 4
11. T2 = (D;G++;seq(E_d,E_s); upd E_m E_d' E_s' )                       | Inspection of (st_B-t), def of G2, 10
12. Forall c=/=Z. P(R_val(pc_c)+1) = T2                                 | Inversion of (C-t), 3, 4,11
5'. Forall c=/=Z. P(R2_val(pc_c))                                       | def of ++, def of R2, 11, 12
    = (D;G++;seq(E_d,E_s); upd E_m E_d' E_s' ) --> void

9'. P |- R++ : S(G++)                                                   | 9, def of ++

13. P;(.;S(G);S(seq(E_d,E_s),(E_d',E_s'));S(E_m))|- st_B r_d, r_s       | 10, 11, substitution, 6
        ==> (.;S(G++);S(seq(E_d,E_s); upd S(E_m) S(E_d') S(E_s') )
14. S(G)(r_d)= <B,b ref,E_d''>                                          | Inversion of (st_B-t), 13
15. S(G)(r_s)= <B,b,E_s''>                                              | Inversion of (st_B-t), 13
16. . |- S(E_d') = S(E_d'')                                             | Inversion of (st_B-t), 13
17. . |- S(E_s') = S(E_s'')                                             | Inversion of (st_B-t), 13


SUBCASE a. Z = G
18a. P |-Z seq(ns1,ns1') : S(seq(E_d,E_s))                              | repeated Inversion of (Q-zap-t), assumption, 8, (Q-zap-t)

SUBCASE b. Z =/= G
18b. P |-Z seq(ns1,ns1') : S(seq(E_d,E_s))                              | repeated Inversion of (Q-t), assumption, 8, Queue Lemma, (Q-t)

MERGE:
8'. P |-Z seq(ns1,ns1') : S(seq(E_d,E_s))                               | 18a/18b
2'. Z=/=G. Dom(seq(ns1,ns1')) subseteq Dom(M[n1->n1'])                  | 2


SUBCASE a. Z = B (queue is correct)
19a. . |- S(E_d') = n1                                                  | Inversion of (Q-t), assumption, 8
20a. P |- n1 : b ref                                                    | Inversion of (Q-t), assumption, 8
21a. P;. |- B n1 : <B,b ref,E_n1>                                       | (val-t), 20a
22a. n1 in Dom(M)                                                       | Cannonical Forms 4, 7, 3, 21a

23a. . |- S(E_s') = n1'                                                 | Inversion of (Q-t), assumption, 8
24a. P |- n1' : b                                                       | Inversion of (Q-t), assumption, 8

25a. [[S(E_d')]] = n1 and [[S(E_s')]] = n1'                             | Inversion of (E-eq), 19a, 23a

SUBCASE b. Z = G (r_s/r_d are correct)
19b. P;. |- R_val(r_d) : <B,b ref, E_d''>                               | Inversion on (reg-file-t), 9, 14
20b. P;. |- n1 : <B,b ref, E_d'>                                        | 19b, (p1), Exp Eq Transitivity, 16
```

```
21b. n1 in Dom(M)                                                     | Cannonical Forms 4, 7, 3, 20b
22b. P |- n1 : b ref                                                  | Inversion on (val-t), assumption, 20b

23b. P;. |- R_val(r_s) : <B,b,E_s''>                                  | Inversion on (reg-file-t), 9, 15
24b. P;. |- n1' : <B,b,E_s'>                                          | 24b, (p2), Exp Eq Transitivity, 17
25b. P |- n1' : b                                                     | Inversion of (val-t), assumption, 24b

26b. . |- S(E_d') = n1 and . |- S(E_s') = n1'                         | Inversion of (val-t), assumption, 20b, 24b
27b. [[S(E_d')]] = n1 and [[S(E_s')]] = n1'                           | Inversion of (E-eq), 26b

MERGE:
30. Forall l in Dom(M). P |- l : b ref and P |- M(l) : b              | Inversion of (M-t), 7
31. Forall l in Dom(M[n1->n1']). P |- l : b ref and P |- M(l) : b     | 30, 20a/22b, 24a/25b

32. [[E_m]] = M                                                       | Inversion of (M-t), 7
33. [[upd E_m E_d' E_s']] = [[E_m]] [ [[E_d']] -> [[E_s']] ]          | def of [[]]
34. [[upd E_m E_d' E_s']] = M [ n1 -> n1' ]                           | 33, 32, 25a/27b

35. . |- E_m : kmem                                                   | Inversion of (M-t), 7
36. . |- E_d' : kint and . |- E_s' : kint                             | Int Kinding Lemma, Inversion of (reg-file-t), 9, 14, 15
37. . |- upd E_m E_d' E_s' : kint                                     | (E-upd-t), 35, 36

7'. P |- M[n1->n1'] : upd E_m E_d' E_s'                               | (M-t), 31, 34, 37
1'. Dom(P) = Dom(C) union Dom(M[n1->n1'])                             | 1, 22a/21b

24. |-Z (R++, C, M[n1->n1'], seq(ns1,ns1') , .)                       | 1', 2', 3, ir=., 5', 6, 7', 8', 9'
*




CASE st_B-mem-fail:

Q = (seq(n,n'),(n1,n1'))
R_val(r_d) =/= n1  or R_val(rs) =/= n1'
------------------------------------------(st_B-mem-fail)
(R,C,M,Q, ld_B r_d, r_s )  -->_0  fault

does not apply (fails second assumption)
*




CASE bz_G-untaken:

(p1) R_val(d) = 0      (p2) R_val(r_z) =/= 0
-----------------------------------------------------(bz-untaken)
(R,C,M,Q, bz_G r_z, r_d )  -->_0  (R++,C,M,Q,.)

0. |-Z (R,C,M,Q,bz_G r_z, r_d)                                        | Given
1. Dom(P) = Dom(C) union Dom(M)                                       | Inversion of (heap-t), 0
2. Z=/=G ==> Dom(Q) subseteq Dom(M)                                   | Inversion of (heap-t), 0
3. P |- C                                                             | Inversion of (heap-t), 0
4. Forall c=/=Z. C(R_val(pc_c)) = bz_G r_z, r_d                       | Inversion of (heap-t), 0
5. Forall c=/=Z. P(R_val(pc_c)) = (D;G;seq(E_d,E_s);E_m)-->void       | Inversion of (heap-t), 0
6. Exists S. . |- S : D                                               | Inversion of (heap-t), 0
7. P |- M : S(E_m)                                                    | Inversion of (heap-t), 0
8. P |-Z Q : S(seq(E_d,E_m))                                          | Inversion of (heap-t), 0
9. P |-Z R : S(G)                                                     | Inversion of (heap-t), 0

Let R2 = R++
Let T' = (D',G',seq(E_d',E_s'),E_m')
Let G2 = G++[ d -> (E_z = 0 ==> <G,T'->void,E_d'>) ]

10. P;(D;G;seq(E_d,E_s);E_m) |- bz_G r_z, r_d ==> RT2                 | Inversion of (C-t), 3, 4
11. RT2 = (D;G2;(seq(E_d,E_s));E_m)                                   | Inspection of (bz_G-t), def of G2, 10
12. Forall c=/=Z. P(R_val(pc_c)+1) = RT2                              | Inversion of (C-t), 3, 4, 11
5'. Forall c=/=Z. P(R2_val(pc_c)) = (D;G2;(seq(E_d,E_s));E_m) --> void  | def of ++, def of R2, 11, 12

13. P;(.;S(G);S(seq(E_d,E_s));S(E_m))|- bz_G r_z, r_d                 | 10, 11, substitution, 6
        ==>(.;S(G2);S(seq(E_d,E_s));S(E_m))
14. S(G)(d)= <G,int,0>                                                | Inversion of (bz_G-t), 13
15. S(G)(r_z)= <G,int,E_z>                                            | Inversion of (bz_G-t), 13
16. S(G)(r_d)= <G,T'->void,E_d'>                                      | Inversion of (bz_G-t), 13
17. G'(r_d)= <G,T'->void,E_d'>                                        | Inversion of (bz_G-t), 13

SUBCASE a: Z = G
18a. . |- E_z : kint                                                  | Inversion of (reg-file-t), 9, 15, Int Kinding Lemma
19a. . |- E_d' : kint                                                 | Inversion of (reg-file-t), 9, 16, Int Kinding Lemma
```

```
20a. P;. |-Z R(d) : (E_z = 0 ==> <G,T'->void,E_d'>)                        | (val-zap-cond), 18a, 19a, assumption, 14

SUBCASE b: Z =/= G
18b. P |- R_val(r_z) : <G, int, E_z>                                        | Inversion of (reg-file-t), 9
19b. . |- E_z = R_val(r_z)                                                  | Inversion of (val-t), assumption, 18b
20b. . |- E_z =/= 0                                                         | 19b, (p2), transitivity
21b. P;. |- R(d) : (E_z = 0 ==> <G,T'->void,E_d'>)                          | (cond-t-n0), (p1), 20b

MERGE:
22. P;. |- R(d) : (E_z = 0 ==> <G,T'->void,E_d'>)                           | 20a/21b
23. P |-Z R++ : S(G++)                                                      | 9, def of ++
9'. P |-Z R2 : S(G2)                                                        | (reg-file-t), 23, 22, def of R2, def of G2

24. |-Z (R++,C,M,Q,.)                                                       | (heap-t), 1, 2, 3, ir=., 5', 6, 7, 8, 9'
*
```

```
CASE bz_B-untaken:

(p1) R_val(d) = 0      (p2) R_val(r_z) = 0
-----------------------------------------------------(bz-untaken)
(R,C,M,Q, bz_B r_z, r_d )  -->_0  (R++,C,M,Q,.)

0. |-Z (R,C,M,Q,bz_B r_z, r_d)                                              | Given
1. Dom(P) = Dom(C) union Dom(M)                                             | Inversion of (heap-t), 0
2. Z=/=G ==> Dom(Q) subseteq Dom(M)                                         | Inversion of (heap-t), 0
3. P |- C                                                                   | Inversion of (heap-t), 0
4. Forall c=/=Z. C(R_val(pc_c)) = bz_B r_z, r_d                             | Inversion of (heap-t), 0
5. Forall c=/=Z. P(R_val(pc_c)) = (D;G;seq(E_d,E_s);E_m)-->void             | Inversion of (heap-t), 0
6. Exists S. . |- S : D                                                     | Inversion of (heap-t), 0
7. P |- M : S(E_m)                                                          | Inversion of (heap-t), 0
8. P |-Z Q : S(seq(E_d,E_m))                                                | Inversion of (heap-t), 0
9. P |-Z R : S(G)                                                           | Inversion of (heap-t), 0

10. P;(D;G;seq(E_d,E_s);E_m) |- bz_G r_z, r_d ==> RT2                       | Inversion of (C-t), 3, 4
11. RT2 = (D;G++;(seq(E_d,E_s));E_m)                                        | Inspection of (bz_B-t), def of G2, 10
12. Forall c=/=Z. P(R_val(pc_c)+1) = RT2                                    | Inversion of (C-t), 3, 4, 11
5'. Forall c=/=Z. P(R2_val(pc_c)) = (D;G++;(seq(E_d,E_s));E_m) --> void | def of ++, def of R2, 11, 12

9'. P |-Z R++ : S(G++)                                                      | (reg-file-t), 9, def of ++

13. |-Z (R++,C,M,Q,.)                                                       | (heap-t), 1, 2, 3, ir=., 5', 6, 7, 8, 9'
*
```

```
CASE bz-untaken-fail

R_val(rz) =/= 0    R_val(d) =/= 0
-------------------------------------------(bz-untaken-fail)
(R,C,M,Q, bz_c rz, rd )  -->_0  fault

does not apply (fails second assumption)
*
```

```
CASE bz_G-taken:

R_val(d) = 0    R_val(r_z) = 0    R2 = R++[d -> R(r_d)]
-------------------------------------------------------------(bz_G-taken)
(R,C,M,Q, bz_G r_z, r_d )  -->_0  (R2,C,M,Q,.)


0. |-Z (R,C,M,Q,bz_G r_z, r_d)                                              | Given
1. Dom(P) = Dom(C) union Dom(M)                                             | Inversion of (heap-t), 0
2. Z=/=G ==> Dom(Q) subseteq Dom(M)                                         | Inversion of (heap-t), 0
3. P |- C                                                                   | Inversion of (heap-t), 0
4. Forall c=/=Z. C(R_val(pc_c)) = bz_G r_z, r_d                             | Inversion of (heap-t), 0
5. Forall c=/=Z. P(R_val(pc_c)) = (D;G;seq(E_d,E_s);E_m)-->void             | Inversion of (heap-t), 0
6. Exists S. . |- S : D                                                     | Inversion of (heap-t), 0
7. P |- M : S(E_m)                                                          | Inversion of (heap-t), 0
8. P |-Z Q : S(seq(E_d,E_m))                                                | Inversion of (heap-t), 0
9. P |-Z R : S(G)                                                           | Inversion of (heap-t), 0
```

```
Let R2 = R++[d -> R(r_d)]
Let T' = (D',G',seq(E_d',E_s'),E_m')
Let G2 = G++[ d -> (E_z = 0 ==> <G,T'->void,E_d'>) ]

10. P;(D;G;seq(E_d,E_s);E_m) |- bz_G r_z, r_d ==> RT2                   | Inversion of (C-t), 3, 4
11. RT2 = (D;G2;(seq(E_d,E_s));E_m)                                     | Inspection of (bz_G-t), def of G2, 10
12. Forall c=/=Z. P(R_val(pc_c)+1) = RT2                                | Inversion of (C-t), 3, 4
5'. Forall c=/=Z. P(R2_val(pc_c)) = (D;G2;(seq(E_d,E_s));E_m) --> void  | def of ++, def of R2, 11, 12

13. P;(.;S(G);S(seq(E_d,E_s));S(E_m))|- bz_G r_z, r_d                   | 10, 11, substitution, 6
        ==>(.;S(G2);S(seq(E_d,E_s));S(E_m))
14. S(G)(d)= <G,int,0>                                                  | Inversion of (bz_G-t), 13
15. S(G)(r_z)= <G,int,E_z>                                              | Inversion of (bz_G-t), 13
16. S(G)(r_d)= <G,T'->void,E_d'>                                        | Inversion of (bz_G-t), 13
17. G'(r_d)= <G,T'->void,E_d'>                                          | Inversion of (bz_G-t), 13

SUBCASE a: Z = G
18a. . |- E_z : kint                                                    | Inversion of (reg-file-t), 9, 15, Int Kinding Lemma
19a. . |- E_d' : kint                                                   | Inversion of (reg-file-t), 9, 16, Int Kinding Lemma
20a. P;. |-Z R(r_d) : (E_z = 0 ==> <G,T'->void,E_d'>)                   | (val-zap-cond), 18a, 19a, assumption, 14

SUBCASE b: Z =/= G
18b. P |- R_val(r_z) : <G, int, E_z>                                    | Inversion of (reg-file-t), 9, 15
19b. . |- E_z = R_val(r_z)                                              | Inversion of (val-t), assumption, 18b
20b. . |- E_z = 0                                                       | 19b, (p2), transitivity

21b. P;. |-Z R(r_d) : <G,T'->void,E_d'>                                 | Inversion of (reg-file-t), 9, 16
22b. R_val(r_d) =/= 0                                                   | Cannonical Forms 3, 7, 3, 21b

23b. P;. |-Z R(r_d) : (E_z = 0 ==> <G,T'->void,E_d'>)                   | (cond-t), 22b, 21b, 20b

MERGE:
22. P;. |- R(r_d) : (E_z = 0 ==> <G,T'->void,E_d'>)                     | 20a/23b
23. P |-Z R++ : S(G++)                                                  | 9, def of ++
9'. P |-Z R2 : S(G2)                                                    | (reg-file-t), 23, 22, def of R2, def of G2

24. |-Z (R2,C,M,Q,.)                                                    | (heap-t), 1, 2, 3, ir=., 5', 6, 7, 8, 9'
*





CASE bz_G-taken-fail:

Rval(r_z) = 0    Rval(d) =/= 0
--------------------------------------------(bz_G-taken-fail)
(R,C,M,Q, bz_G r_z, r_d )  -->_0  fault

does not apply (fails second assumption)
*





CASE bz_B-taken:

(p1) R_val(d) =/= 0
(p2) R_val(r_z) = 0
(p3) R_val(r_d) = R_val(d)
R2 = R[pc_G -> R(d)][pc_B -> R(r_d)][d -> G 0]
--------------------------------------------------(bz_B-taken)
(R,C,M,Q, bz_B r_z, r_d )  -->_0  (R2,C,M,Q,.)

0. |-Z (R,C,M,Q,z_B r_z, r_d)                                           | Given
1. Dom(P) = Dom(C) union Dom(M)                                         | Inversion of (heap-t), 0
2. Z=/=G ==> Dom(Q) subseteq Dom(M)                                     | Inversion of (heap-t), 0
3. P |- C                                                               | Inversion of (heap-t), 0
4. Forall c=/=Z. C(R_val(pc_c)) = bz_B r_z, r_d                         | Inversion of (heap-t), 0
5. Forall c=/=Z. P(R_val(pc_c)) = (D;G;seq(E_d,E_s);E_m)-->void         | Inversion of (heap-t), 0
6. Exists S. . |- S : D                                                 | Inversion of (heap-t), 0
7. P |- M : S(E_m)                                                      | Inversion of (heap-t), 0
8. P |-Z Q : S(seq(E_d,E_m))                                            | Inversion of (heap-t), 0
9. P |-Z R : S(G)                                                       | Inversion of (heap-t), 0

Let R2 = R[pc_G -> R(d)][pc_B -> R(r_d)][d -> G 0]
Let T' = (D',G',seq(E_d',E_s'),E_m')

10. P;(D;G;seq(E_d,E_s);E_m) |- bz_B r_z, r_d ==> RT2                   | Inversion of (C-t), 3, 4
11. RT2 = (D;G++;(seq(E_d,E_s));E_m)                                    | Inspection of (bz_B-t), def of G2, 10
12. Forall c=/=Z. P(R_val(pc_c)+1) = RT2                                | Inversion of (C-t), 3, 4,11
```

```
13. P;(.;S(G);S(seq(E_d,E_s));S(E_m))|- bz_B r_z, r_d                  | 10, 11, substitution, 6
        ==>(.;S(G++);S(seq(E_d,E_s));S(E_m))
14. S(G)(d)= (E_z'=0 ==> <G,T'->void,E_r'>                             | Inversion of (bz_B-t), 13
15. S(G)(r_z)= <B,int,E_z>                                             | Inversion of (bz_B-t), 13
16. S(G)(r_d)= <B,T'->void,E_r>                                        | Inversion of (bz_B-t), 13
17. . |- E_z = E_z'                                                    | Inversion of (bz_B-t), 13
18. . |- E_r = E_r'                                                    | Inversion of (bz_B-t), 13
19. Exists S'. . |- S': D'                                             | Inversion of (bz_B-t), 13
20. . |- S(G) <= S'(G')                                                | Inversion of (bz_G-t), 13
21. S'(G')(d) = <G,int,0>                                              | Inversion of (bz_G-t), 13
22. S'(G')(pc_G) = <G,int,E_r'>                                        | Inversion of (bz_G-t), 13
21. S'(G')(pc_B) = <B,int,E_r>                                         | Inversion of (bz_G-t), 13
22. . |- S(seq(E_d,E_s)) = S'(seq(E_d',E_s'))                          | Inversion of (bz_G-t), 13
23. . |- S(E_m) = S'(E_m)                                              | Inversion of (bz_G-t), 13

24. R2_val(pc_G) = R2_val(pc_B) = R_val(r_d) = R_val(d)                | def of R2, (p3)

SUBCASE a: Z = G
25a. P;. |-Z R(d) : <B,T'->void,E_r'>                                  | Inversion of (reg-file-t), 7, 16
26a. P |- R_val(d) : T'->void                                          | Inversion of (val-t), assumption, 25a
27a. P |- R2_val(pc_B) : T'->void                                      | 26a, 24

SUBCASE b: Z =/= G
25b. P;. |-Z R(d): (E_z'=0 ==> <G,T'->void,E_r'>                       | Inversion of (reg-file-t), 7, 16
26b. P |- R_val(d) : T'->void                                          | Inversion of (cond-t), 25b, assumption, (p1)
27b. Forall c=/=Z. P(R2_val(pc_c)) = T' -> void                        | 26b, 24

MERGE:
5'. Forall c=/=Z. P(R2_val(pc_c)) = (D',G',seq(E_d',E_s'),E_m') -> void | 27a/27b

6'. Exists S'. . |- S' : D'                                            | 19
7'. P |- M : S'(E_m)                                                   | 7, 23, repeated applications of Substituting Closed
                                                                         Expressions Lemma
8'. P |-Z Q : S'(seq(E_d,E_m))                                         | 8, 22, repeated applications of Substituting Closed
                                                                         Expressions Lemma

28. P |-Z R : S'(G')                                                   | Subtyping Lemma, 9, 20, repeated applications of def of <=
                                                                         on regfiles
29. P;. |-Z G 0 : <G,int,0>                                            | (val-t)
30. P;. |-Z G 0 : S'(G)(d)                                             | 21, 29

SUBCASE a: Z = G
30a.  . |- E_r' : kint                                                 | Inversion of (E-eq), 18
31a. P;. |-Z R(d) : <G,int,E_r'>                                       | (val-zap-t), assumption, 30a
32a. P;. |-Z R(d) : S'(G')(pc_G)                                       | 31a, 22
33a. P;. |-Z R(r_d) : <B,T'->void,E_r>                                 | Inversion of (reg-file-t), 9, 16
34a. P;. |-Z R(r_d) : <B,int,E_r>                                      | Subtyping Lemma, 33a, (subtp-int)
35a. P;. |-Z R(r_d) : S'(G')(pc_B)                                     | 34a, 21

SUBCASE b: Z = B
30b. . |- E_r : kint                                                   | Inversion of (E-eq), 18
31b. P;. |-Z R(r_d) : <B,int,E_r>                                      | (val-zap-t), assumption, 30b
32b. P;. |-Z R(r_d) : S'(G')(pc_B)                                     | 31b, 21
33b. P;. |-Z R(d) : (E_z'=0 ==> <G,T'->void,E_r'>)                     | Inversion of (reg-file-t), 9, 14
34b. <deleted>
35b. P;. |-Z R(d) : <G,T'->void,E_r'>                                  | Inversion of (cond-t), assumption, 33b
36b. P;. |-Z R(d) : <G,int, E_r'>                                      | Subtyping Lemma, 35b, (subtp-int)
37b. P;. |-Z R(d) : S'(G')(pc_G)                                       | 36b, 22

MERGE:
38. P;. |-Z R(d)   : S'(G')(pc_G)                                      | 32a / 37b
39. P;. |-Z R(r_d) : S'(G')(pc_B)                                      | 35a / 32b
9'. P |-Z R2 : S'(G')                                                  | 28, def of R2,, 30, 38, 39

41. |-Z (R2,C,M,Q,.)                                                   | (heap-t), 1,2,3,ir=.,5',6',7',8',9'
```

```
CASE bz_b-taken-fail:

R_val(r_z) = 0
R_val(r_d) =/= R_val(d)  or  Rval(d) = 0
---------------------------------------------(bz_B-taken-fail)
(R,C,M,Q, bz_B r_z, r_d )  -->_0  fault

does not apply (fails second assumption)
*
```

```
CASE jmp_G:

(p1) R_val(d) = 0     R2 = R++[d -> R(r_d)]
------------------------------------------------(jmp_G)
(R,C,M,Q, jmp_G rd )  -->_0  (R2,C,M,Q,.)

Similar to bz_G-taken.
*




CASE jmp_G-fail:

Rval(d) =/= 0
-------------------------------------------(jmp_G-fail)
(R,C,M,Q, jmp_G rd )  -->_0  fault

does not apply (fails second assumption)
*




CASE jmp_B:

(p1) R_val(d) =/= 0
(p2) R_val(r_d) = R_val(d)
R2 = R[pc_G -> R(d)][pc_B -> R(r_d)][d -> G 0]
---------------------------------------------------(jmp_B)
(R1,C,M,Q1, jmp_B rd )  -->_0  (R2,C,M,Q1,.)

Similar to bz_B-taken.
*



CASE jmp_B-fail:

R_val(r_d) =/= R_val(d)  or  R_val(d) = 0
--------------------------------------------(jmp_B-fail)
(R,C,M,Q, jmp_B rd )  -->_0  fault

does not apply (fails second assumption)
*
```

Preservation Part 2

```
2. If |- (R,C,M,Q,ir)
   and (R,C,M,Q,ir)  -->_1^s  (R',C',M',Q',ir')
   then Exists Z : |-Z (R',C',M',Q',ir')

Proof by induction on the structure of the derivation of (R,C,M,Q,ir)  -->_1^s  (R',C',M',Q',ir').



CASE reg-zap: FJP -- S(G)(a) may have conditional shape, see version in TR

R(a) = c n
---------------------------------------------- (reg-zap)
(R,C,M,Q,ir) -->_1 ( R[ a-> c n' ], C,M,Q,ir)

0. |- (R,C,M,Q,ir)                                                       | Given
1. Dom(P) = Dom(C) union Dom(M)                                          | Inversion of (heap-t), 0
2.  Dom(Q) subseteq Dom(M)                                               | Inversion of (heap-t), 0
3. P |- C                                                                | Inversion of (heap-t), 0
4. Forall c. C(R_val(pc_c)) = ir                                         | Inversion of (heap-t), 0
5. Forall c. P(R_val(pc_c))=(D;G;seq(E_d,E_s);E_m) --> void              | Inversion of (heap-t), 0
6. Exists S. . |- S : D                                                  | Inversion of (heap-t), 0
7. P |- M : S(E_m)                                                       | Inversion of (heap-t), 0
8. P |- Q : S(seq(E_d,E_m))                                              | Inversion of (heap-t), 0
9. P |- R : S(G)                                                         | Inversion of (heap-t), 0

Let c = R_col(a)
Let <c,b,E> = S(G)(a)

10. P;. |-c c n' : <c,b,E>                                               | (val-zap-t)
11. P;. |-c R[a -> c n'](a) : S(G)(a)                                    | 10, def of <c,b,E>
9'. P |-c R[a -> c n'] : S(G)                                            | (reg-file-t), 9, 11

12. |-c (R[a -> c n'],C,M,Q,ir)                                          | (heap-t), 1, 2, 3, 4, 5, 6, 7, 8, 9'
*



CASE Q1-zap:

Q1 = (seq(n1,n1'),(m1,m'),seq(n2,n2'))
Q2 = (seq(n1,n1'),(m2,m'),seq(n2,n2'))
---------------------------------------------- (Q1-zap)
(R,C,M,Q1,ir) -->_1 (R,C,M,Q2,ir)


0. |- (R,C,M,Q1,ir)                                                      | Given
1. Dom(P) = Dom(C) union Dom(M)                                          | Inversion of (heap-t), 0
2. Dom(Q1) subseteq Dom(M)                                               | Inversion of (heap-t), 0
3. P |- C                                                                | Inversion of (heap-t), 0
4. Forall c. C(R_val(pc_c)) = ir                                         | Inversion of (heap-t), 0
5. Forall c. P(R_val(pc_c))=(D;G;seq(E_d,E_s);E_m) --> void              | Inversion of (heap-t), 0
6. Exists S. . |- S : D                                                  | Inversion of (heap-t), 0
7. P |- M : S(E_m)                                                       | Inversion of (heap-t), 0
8. P |- Q1 : S(seq(E_d,E_m))                                             | Inversion of (heap-t), 0
9. P |- R : S(G)                                                         | Inversion of (heap-t), 0

let Z = G

2'. Z=/=G ==> Dom(Q) subseteq Dom(M)                                     | def of Z
4'. Forall c=/=Z. C(R_val(pc_c)) = ir                                    | 4
5'. Forall c=/=Z. P(R_val(pc_c))=(D;G;seq(E_d,E_s);E_m) --> void         | 5


10. P |-Z Q1 : S(seq(E_d,E_s))                                           | Color Weakening Q Lemma, 8
8'. P |-Z Q2 : seq(E_d,E_s)                                              | (Q-zap-t), def of Z

9'. P |-Z R : S(G)                                                       | Color Weakening R Lemma, 9
```

```
11. |-Z (R,C,M,Q2,ir)                                                        | (heap-t), 1, 2', 3, 4', 5', 6, 7, 8', 9'
*



CASE Q2-zap:

Q1 = (seq(n1,n1'),(m,m1),seq(n2,n2'))
Q2 = (seq(n1,n1'),(m,m2),seq(n2,n2'))
---------------------------------------------- (Q1-zap)
(R,C,M,Q1,ir) -->_1 (R,C,M,Q2,ir)

Same as CASE Q1-zap.
*
```

No False Positives Lemma

```
If |- (R,C,M,Q,ir) then Forall n. (R,C,M,Q,ir) -n->_0 (R',C',M',Q',ir') and |- (R',C',M',Q',ir')

Proof: By induction over the multistep definition and use of part (1) of Progress and part (1) of Preservation (with z = .)
```

Single Step Fault Detection

If `|- S1` and `S1 sim_c S2` and `S1 -->_0^s1 S1'` then `S2 -->_0^s2 S2'` and `s2` is a prefix of `s1` and either

1. `S1' sim_c S2'`
2. `S2' = fault`

FJP -- need to modify so that singlestep is (simulates & output equal) or (reaches fault & output empty). See ETAL_FT for an example.
"If a fault has occurred, then either the faulty computation can take a step indistinguishable from the non-faulty version, or the faulty computation reaches a fault state."

Proof:

By induction on the structure of `S1 -->_0^s1 S1'`

```
(a1) S1 sim_c S2

(a2) S1 -->_0^s1 S1'

(a3) |- S1


(1) S2 = (R2,C,M,Q2,ir)                                    [ (a1), (sim-S) ]

(2) R1 sim_c R2                                            [ (a1), (1), (sim-S) ]

(3) Q1 sim_c Q2                                            [ (a1), (1), (sim-S) ]
```

CASE FETCH 1: fetch

```
(p1) R1_val(pc1_G) = R1_val(pc1_B)

(p2) R1_val(pc1_G) in Dom(C)

--------------------------------------------------(fetch)

(R1,C,M,Q1,.)  -->_0  (R1,C,M,Q1,C(R1_val(pc1_G)))


Case on R2_val(pc_G) =? R2_val(pc_B)
```

SUB-CASE FETCH 1.1: One of the pc's was zapped

```
(a5) R2_val(pc_G) =/= R2_val(pc_B)


(4) (R2,C,M,Q2,.) -->_0 fault                                   [ (fetch-fail), (a5) ]

(5) S2 -->_0 fault and () = ()                                  [ (4) ]
 *
```

SUB-CASE FETCH 1.2: Neither pc was zapped

```
(a5) R2_val(pc_G) = R2_val(pc_B)


(4) R1(pc_G) sim_c R2(pc_G)                                          [ (2), (sim-R) ]
(5) R1_col(pc_G) = G                                                 [ (a3), inversion on (reg-file-t) ]
(6) R2_col(pc_G) = G                                                 [ (sim-val), (4), (5) ]
(7) c = B ==> R1(pc_G) = R2(pc_G)                                 [ (sim-val), (4), (5), (6) ]
(8) c = B ==> R2_val(pc_G) in Dom(C)                              [ (p2), (5) ]


(9)  R1(pc_B) sim_c R2(pc_B)                                         [ (2), (sim-R) ]
(10) R1_col(pc_B) = B                                                [ (a3), inversion on (reg-file-t) ]
(11) R2_col(pc_B) = B                                                [ (sim-val), (9), (10) ]
(12) c = G ==> R1(pc_B) = R2(pc_B)                                [ (sim-val), (9), (10), (11) ]
(13) R1_val(pc_B) in Dom(C)                                          [ (p1), (p2) ]
(14) c = G ==> R2_val(pc_B) in Dom(C)                             [ (12), (13) ]


(15) R2_val(pc_G) in Dom(C)                                          [ (8), (14), (a5) ]
(16) (R2,C,M,Q2,.) -->_0 (R2,C,M,Q2,C(R2_val(pc_G))               [ (fetch), (a5), (15) ]


(17) R1_val(pc_G) = R2_val(pc_G)                                     [ (p1), (a5), (7), (12) ]
(18) (R1,C,M,Q1,C(R1_val(pc_G)))  sim_c  (R2,C,M,Q2,C(R2_val(pc_G))    [ (sim-S), (2), (3), (17) ]
(19) S2 -->_0 S2' and ()=() and S2' sim_c S1'                             [ (16), (18) ]
 *
```

CASE FETCH 2: fetch-fail

```
Rval(pc_G) =/= Rval(pc_B)
--------------------------(fetch-fail)
(R,C,M,Q,.)  -->_0  fault



(4) S1 -/->_0 fault                                               [ (Progress 1), (a3) ]
(5) case does not apply
 *
```

CASE BASIC 1: op2r

```
(s1) R1' = R1++[ rd -> R1_col(rt) (R1_val(rs) op R1_val(rt)) ]
-----------------------------------------------------------------(op2r)
(R1,C,M,Q1, op rd, rs, rt )  -->_0  (R1',C,M,Q1,.)



(4) let R2' = R2++[ rd -> R2_col(rt) (R2_val(rs) op R2_val(rt)) ]
(5) (R2,C,M,Q2,.) -->_0 (R2',C,M,Q2,.)                                  [ (op2r), (4) ]

(6) R1(rt) sim_c R2(rt)                                             [ (2), (sim-R) ]
(7) R1(rs) sim_c R2(rs)                                             [ (2), (sim-R) ]
(8) R1++ sim_c R2++                                                 [ (2), def of ++, (sim-R), handwave ]

(9)  R1_col(rt) = R1_col(rs)                                        [ (a3), inversion of (op1r-t), handwave ]
(10) R1_col(rt) = R2_col(rt) = R2_col(rs)                           [ (2), (9), (sim-val), handwave ]

Case on R1_col(rt) =? c
```

SUB-CASE BASIC 1.1: R1_col(rt) =/= c

```
(a5) R1_col(rt) =/= c

(11) R1_val(rt) = R2_val(rt)                                        [ (6), (10), (sim-val), (a5) ]
(12) R1_val(rs) = R2_val(rs)                                        [ (7), (9), (10), (sim-val), (a5) ]
(13) R1_val(rs) op R1_val(rt) = R2_val(rs) op R2_val(rt)            [ (11), (12) ]
(14) R1_col(rt) (R1_val(rs) op R1_val(rt))                          [ (sim-val), (a5), (10), (13) ]
       sim_c  R2_col(rt) (R2_val(rs) op R2_val(rt))

(15) R1' sim_c R2'                                                  [ (2), (8), (14), (sim-R), handwave ]
(16) (R1',C,M,Q1,.) sim_c (R2',C,M,Q2,.)                            [ (sim-S), (15), (3) ]

(17) S2 -->_0 S2' and S1' sim_c S2' and () = ()                          [ (5), (16) ]
 *
```

SUB-CASE BASIC 1.2: R1_col(rt) = c

```
(a5) R1_col(rt) = c

(11) R1_col(rt) (R1_val(rs) op R1_val(rt))                          [ (sim-val-zap), (a5), (10) ]
       sim_c  R2_col(rt) (R2_val(rs) op R2_val(rt))
```

```
(12) R1' sim_c R2'                                    [ (2), (8), (11), (sim-R), handwave ]
(13) (R1',C,M,Q1,.) sim_c (R2',C,M,Q2,.)              [ (sim-S), (12), (3) ]

(14) S2 -->_0 S2' and S1' sim_c S2' and () = ()           [ (5), (13) ]
 *
```

CASE BASIC 2: op1r

```
(s1) R1' = R1++[ rd -> c' (R1_val(rs) op n) ]
----------------------------------------------------------(op1r)
(R1,C,M,Q1, op rd, rs, c' n )  -->_0  (R1',C,M,Q1,.)



(4) let R2' = R2++[ rd -> c' (R2_val(rs) op n) ]
(5) (R2,C,M,Q2,.) -->_0 (R2',C,M,Q2,.)                    [ (op1r, (4) ]

(6) R1(rs) sim_c R2(rs)                               [ (sim-R), (2) ]
(7) R1++ sim_c R2++                                   [ (2), def of ++, (sim-R), handwave ]

Case on c =? c'
```

SUB-CASE BASIC 2.1: c' =/= c

```
(a5) c' =/= c

(8) R1_val(rs) = R2_val(rs)                           [ (6), (sim-val), (a5) ]
(9) R1_val(rs) op n = R2_val(rs) op n                 [ (8) ]
(10) c' (R1_val(rs) op n)  sim_c  c' (R2_val(rs) op n)    [ (sim-val), (9) ]

(11) R1' sim_c R2'                                    [ (sim-R), (7), (10), handwave ]
(12) (R1',C,M,Q1,.) sim_c (R2',C,M,Q2,.)              [ (sim-S), (11), (3) ]

(13) S2 -->_0 S2' and S1' sim_c S2' and () = ()           [ (5), (12) ]
 *
```

SUB-CASE BASIC 2.2: c' = c

```
(a5) c' = c
```

```
(8) c' (R1_val(rs) op n)  sim_c  c' (R2_val(rs) op n)                   [ (sim-val-zap), (a5) ]


(9) R1' sim_c R2'                                                       [ (sim-R), (7), (8), handwave ]

(10) (R1',C,M,Q1,.) sim_c (R2',C,M,Q2,.)                                [ (sim-S), (9), (3) ]


(11) S2 -->_0 S2' and S1' sim_c S2' and () = ()                              [ (5), (10) ]
 *
```

CASE BASIC 3: mov

```
-----------------------------------------------------------(mov)
(R1,C,M,Q1, mv rd, v )  -->_0  (R1++[rd -> v] ,C,M,Q1,.)



(4) (R2,C,M,Q2, mv rd, v ) -->_0  (R2++[rd -> v], C,M,Q2,.)               [ (mov) ]
(5) R1++ sim_c  R2++                                                    [ (2), def of ++, (sim-R), handwave ]

(6) v sim_c v                                                           [ (sim-val) ]
(7) R1++[rd -> v]  sim_c  R2++[rd -> v]                                   [ (sim-R), (5), (6) ]

(8) (R1++[rd -> v] ,C,M,Q1,.) sim_c (R2++[rd -> v] ,C,M,Q2,.)             [ (sim-S), (7), (3) ]
(9) S2 -->_0 S2' and S1' sim_c S2' and () = ()                               [ (4), (8) ]
 *
```

CASE LOAD 1: ld_G-queue

```
(p1) find(Q1,R1_val(rs)) = (R1_val(rs),n)
----------------------------------------------------------------------(ld_G-queue)
(R1,C,M,Q1, ld_G rd, rs )  -->_0  (R1++[rd -> G n] ,C,M,Q1,.)


Case on c ?= B,
  and if not, does find(Q2,R2_val(rs)) ?= (R2_val(rs),n')
  and if not, is R2_val(rs) not in? Dom(M)
```

SUB-CASE LOAD 1.1: the value zapped was blue

```
(a5) c = B


(4) find(Q2,R2_val(rs)) = (R2_val(rs),n)                                [ (sim-Q), (3), (sim-val), (a5), handwave ]
```

```
(5) (R2,C,M,Q2, ld_G rd, rs )  -->_0  (R2++[rd -> G n] ,C,M,Q2,.) [ (ld_G-queue), (4) ]

(6) R1++ sim_B R2++                                                    [ (2), def of ++, (sim-R), handwave ]

(7) R1++[rd -> G n] sim_B R2++[rd -> G n]                        [ (sim-R), (6), (sim-val) ]


(8) (R1++[rd -> G n] ,C,M,Q1,.) sim_B (R2++[rd -> G n] ,C,M,Q2,.) [ (sim-S), (7), (3) ]

(9) S2 -->_0 S2' and S1' sim_B S2' and () = ()                              [ (5), (8) ]

 *
```

SUB-CASE LOAD 1.2: rs was not zapped, associated queue value may have been zapped

```
(a5) c = G

(a6) find(Q2,R2_val(rs)) = (R2_val(rs),n')


(4) (R2,C,M,Q2, ld_G rd, rs )  -->_0  (R2++[rd -> G n] ,C,M,Q2,.) [ (ld_G-queue), (a5) ]

(5) R1++ sim_G R2++                                                    [ (2), def of ++, (sim-R), handwave ]

(6) G n  sim_G  G n'                                                   [ (sim-val-zap) ]

(7) R1++[rd -> G n] sim_G R2++[rd -> G n']                               [ (sim-R), (5), (6) ]


(8) (R1++[rd -> G n] ,C,M,Q1,.) sim_B (R2++[rd -> G n'] ,C,M,Q2,.)       [ (sim-S), (7), (3) ]

(9) S2 -->_0 S2' and S1' sim_B S2' and () = ()                              [ (4), (8) ]

 *
```

SUB-CASE LOAD 1.3: rs was zapped to an invalid address

```
(a5) c = G

(a6) find(Q2,R2_val(rs)) = ()

(a7) R2_val(rs) not in Dom(M)


(4) (R2,C,M,Q2, ld_G rd, rs )  -->_0  (R2++[rd -> G n'] ,C,M,Q2,.)       [ (ld_G-rand), (a6), (a7) ]

(5) R1++[rd -> G n] sim_G R2++[rd -> G n']                               [ (sim-R), (2), def++, (sim-val-zap) ]


(6) S2 -->_0 S2'  and S1' sim_G S2' and () = ()                             [ (4), (5) ]

 *
```

SUB-CASE LOAD 1.4: rs was zapped to a valid address or the queue address was zapped

```
(a5) c = G

(a6) find(Q2,R2_val(rs)) = ()

(a7) R2_val(rs) in Dom(M)
```

```
(4) let R2' = R2++[rd -> G M(R2_val(rs))]
(5) (R2,C,M,Q2, ld_G rd, rs )  -->_0  (R2',C,M,Q2,.)                [ (ld_G-mem), (a6), (a7), (4) ]

(6) G n  sim_G   G M(R2_val(rs))                                       [ (sim-val-zap) ]
(7) R1++  sim_G   R2++                                                 [ (2), def of ++, (sim-R), handwave ]
(8) R1++[rd -> G n]  sim_G  R2++[rd -> G M(R2_val(rs))]                  [ (sim-R), (7), (6) ]
(9) (R1++[rd -> G n] ,C,M,Q1,.)  sim_G  (R2',C,M,Q2,.)                      [ (sim-S), (8), (3) ]

(9) S2 -->_0 S2'  and  S1' sim_c S2' and () = ()                    [ (5), (9) ]
 *
```

CASE LOAD 2: ld_G-mem

```
(p1) find(Q1,R1_val(rs)) = ()
(p2) R1_val(rs) in Dom(M)
(s1) R1' = R1++[rd -> G M(R1_val(rs)) ]
---------------------------------------------------(ld_G-mem)
(R1,C,M,Q1, ld_G rd, rs )  -->_0  (R1',C,M,Q1,.)



Case on c ?= B,
  and if not, does find(Q2,R2_val(rs)) ?= (R2_val(rs),n')
  and if not, is R2_val(rs) not in? Dom(M)
```

SUB-CASE LOAD 2.1: the value zapped was blue

```
(a5) c = B

(4) R1_col(rs) = G                                                     [ (a3), inversion of (ld_G-t), handwave ]
(5) R1_col(rs) = R2_col(rs)                                            [ (sim-R), (2), (sim-val) ]
(6) R1_val(rs) = R2_val(rs)                                            [ (sim-R), (2), (4), (5), (a5), (sim-val) ]
(7) R2_val(rs) in Dom(M)                                               [ (6), (p2) ]

(8) find(Q2,R2_val(rs)) = ()                                           [ (sim-Q), (a5), (6), (p1) ]

(9)  let R2' = R2++[rd -> G M(R2_val(rs))]
(10) (R2,C,M,Q2, ld_G rd, rs ) -->_0 (R2',C,M,Q2,.)                 [ (ld_G-queue), (7), (8), (9) ]

(11) R1++ sim_B R2++                                                   [ (2), def of ++, (sim-val), handwave ]
```

```
(12) M(R1_val(rs)) = M(R2_val(rs))                                        [ (6) ]

(13) G M(R1_val(rs)) sim_B  G M(R2_val(rs))                               [ (sim-val), (6) ]

(14) R1' sim_B R2'                                                        [ (sim-R), (11), (13) ]

(15) (R1',C,M,Q1,.) sim_B (R2',C,M,Q2,.)                                  [ (sim-S), (14), (3) ]


(16) S2 -->_0 S2' and S1' sim_B S2' and () = ()                                [ (10), (15) ]
 *
```

SUB-CASE LOAD 2.2: rs zapped to match a queue address or queue address zapped to match rs

```
(a5) c = G

(a6) find(Q2,R2_val(rs)) = (R2_val(rs),n')


(4) let R2' = R2++[rd -> G n']

(5) (R2,C,M,Q2, ld_G rd, rs ) -->_0 (R2',C,M,Q2,.)                     [ (ld_G-queue), (a6), (4) ]

(6) R1++ sim_G R2++                                                       [ (2), def of ++, (sim-val), handwave ]

(7) G M(R1_val(rs) sim_G G n'                                             [ (sim-val-zap) ]

(8) R1' sim_G R2'                                                         [ (sim-R), (6), (7) ]

(9) (R1',C,M,Q1,.) sim_G (R2',C,M,Q2,.)                                   [ (sim-S), (8), (3) ]


(10) S2 -->_0 S2' and S1' sim_c S2' and () = ()                                [ (5), (9) ]
 *
```

SUB-CASE LOAD 2.3: rs zapped to an invalid address

```
(a5) c = G

(a6) find(Q2,R2_val(rs)) = ()

(a7) R2_val(rs) not in Dom(M)


(4) let R2' = R2++[rd -> G n]

(5) G n  sim_G  G M(R1_val(rs))                                           [ (sim-val-zap) ]

(6) R2' sim_G R1'                                                         [ (2), def of ++, (5) ]

(7) (R1',C,M,Q1,.) sim_G (R2',C,M,Q2,.)                                   [ (sim-S), (6), (3) ]


(8) (R2,C,M,Q2,.) -->_0 (R2',C,M,Q2,.)                                         [ (ld_G-rand), (a6), (a7), (4) ]


(9) S2 -->_0 S2' and S1' sim_G S2' and () = ()                                 [ (7), (8) ]
 *
```

SUB-CASE LOAD 2.4: rs zapped to a valid address not in the queue or nothing related was zapped

```
(a5) c = G
(a6) find(Q2,R2_val(rs)) = ()
(a7) R2_val(rs) in Dom(M)

(4) let R2' = R2++[rd -> G M(R2_val(rs))]
(5) (R2,C,M,Q2,.) -->_0 (R2',C,M,Q2,.)                                              [ (ld_G-mem), (a6), (a7), (4) ]

(6) G M(R2_val(rs))  sim_G  G M(R1_val(rs))                                    [ (sim-val-zap) ]
(7) R1' sim_G R2'                                                              [ (2), def of ++, (6) ]
(8) (R1',C,M,Q1,.) sim_G (R2',C,M,Q2,.)                                        [ (sim-S), (3), (7) ]

(9) S2 -->_0 S2' and S1' sim_G S2' and () = ()                                      [ (5), (8) ]
 *
```

CASE LOAD 3: ld_G-rand

```
find(Q1,R1_val(rs)) = ()
R1_val(rs) not in Dom(M)
R1' = R1++[rd -> G n ]
---------------------------------------------------(ld_G-rand)
(R1,C,M,Q1, ld_G rd, rs )  -->_0  (R1',C,M,Q1,.)


(1) R_val(r_s) in Dom(M)                                                       [ Well-Typed Domain Lemma, (a3) ]
(2) case does not apply
*
```

CASE LOAD 4: ld_G-fail

```
find(Q1,R1_val(rs)) = ()
R1_val(rs) not in Dom(M)
-----------------------------------------------------------(ld_G-fail)
(R1,C,M,Q1, ld_G r_d, r_s)  -->_0  fault
```

```
(4) S1 -/->_0 fault                                                    [ (Progress 1), (a3) ]

(5) case does not apply

 *
```

CASE LOAD 5: ld_B-mem

```
R1_val(rs) in Dom(M)

R1' = R1++[ rd -> B M(R1_val(rs)) ]

---------------------------------------------------(ld_B-mem)

(R1,C,M,Q1, ld_B rd, rs )  -->_0  (R',C,M,Q,.)



Case on R2_val(rs) in? Dom M
```

SUB-CASE LOAD 5.1: rs is zapped to an invalid address

```
(a5) R2_val(rs) not in Dom(M)


(4) (R2,C,M,Q2, ld_B rd, rs ) -->_0 fault                              [ (ld_B-fail), (a5) ]

(5) S2 -->_0 fault and () = ()                                                 [ (4) ]

 *
```

SUB-CASE LOAD 5.1: rz is zapped to valid address or rs is not zapped

```
(a5) R2_val(rs) in Dom(M)


(4) let R2' = R2++[ rd -> B M(R2_val(rs)) ]

(5) (R2,C,M,Q2, ld_B rd, rs ) -->_0 (R2',C,M,Q2,.)                     [ (ld_B-mem), (a5), (4) ]


(6) R1_col(rs) = B                                                        [ (a3), inversion of (ld_B-t) ]

(7) R2_col(rs) = B                                                        [ (2), (sim-R), (sim-val), (6) ]


(8) c = G ==> R1_val(rs) = R2_val(rs)                                          [ (sim-val), (2), (6), (7) ]

(9) c = G ==> B M(R1_val(rs)) sim_G  B M(R2_val(rs))                   [ (sim-val), (8) ]


(10) c = B ==> B M(R1_val(rs)) sim_B B M(R2_val(rs))                   [ (sim-val-zap) ]


(11) B M(R1_val(rs)) sim_c B M(R2_val(rs))                                [ (9), (10) ]

(12) R1'  sim_c  R2'                                                      [ (2), def of ++, (11) ]
```

```
(13) (R1',C,M,Q2,.)  sim_c  (R2',C,M,Q2,.)                              [ (sim-S), (12), (3) ]


(14) S2 -->_0 S2' and S1' sim_c S2' and ()=()                                [ (5), (13) ]
 *
```

CASE LOAD 6: ld_B-rand

```
R_val(rs) not in Dom(M)
R' = R++[ rd -> B n ]
---------------------------------------------------(ld_B-rand)
(R,C,M,Q, ld_B rd, rs )  -->_0  (R',C,M,Q,.)

(1) R_val(r_s) in Dom(M)                                                [ Well-Typed Domain Lemma, (a3) ]
(2) case does not apply
*
```

CASE LOAD 7: ld_B-fail

```
Rval(rs) not in Dom(M)
-------------------------------------------(ld_B-fail)
(R,C,M,Q, ld_B rd, rs )  -->_0  fault


(4) S1 -/->_0 fault                                                  [ (Progress 1), (a3) ]
(5) case does not apply
 *
```

CASE STORE 1: st_G-queue

```
(s1) Q1' = ( (R1_val(rd), R1_val(rs)), Q1 )
-------------------------------------------------------(st_G-queue)
(R1,C,M,Q1, st_G rd, rs )  -->_0  (R1++,C,M,Q1',.)



(4) let Q2' = ( R2_val(rd), R2_val(rs) ), Q2
(5) (R2,C,M,Q2, st_G rd, rs )  -->_0  (R2++,C,M,Q2',.)                       [ (st_G-queue), (4) ]


(6) R1_col(rs) = G                                                      [ (a3), inversion on (ld_G-t) ]
(7) R1_col(rd) = G                                                      [ (a3), inversion on (ld_G-t) ]
```

```
(8) R2_col(rs) = G                                          [ (6), (2), (sim-val) ]

(9) R2_col(rd) = G                                          [ (7), (2), (sim-val) ]


(10) c = G ==> G R1_val(rd)  sim_G  G R2_val(rd)         [ (sim_val-zap) ]

(11) c = G ==> G R1_val(rs)  sim_G  G R2_val(rs)         [ (sim_val-zap) ]

(12) c = G ==> Q1' sim_G Q2'                              [ (sim-Q), (3), (10), (11) ]


(13) c = B ==> R1_val(rd) = R2_val(rd)                           [ (sim-R), (2), (7), (9), (sim-val) ]

(14) c = B ==> R1_val(rs) = R2_val(rs)                           [ (sim-R), (2), (6), (8), (sim-val) ]

(15) c = B ==> G R1_val(rs)  sim_B  G R2_val(rs)         [ (sim-val), (14) ]

(16) c = B ==> G R1_val(rd)  sim_B  G R2_val(rd)         [ (sim-val), (13) ]

(17) c = B ==> Q1' sim_B Q2'                              [ (sim-Q), (3), (15), (16) ]


(18) Q1' sim_c Q2'                                          [ (12), (17) ]

(19) (R1++,C,M,Q1',.) sim_c (R2++,C,M,Q2',.)               [ (sim-S), (2), def of ++, (18) ]


(20) S2 -->_0 S2' and S1' sim_c S2' and () = ()          [ (5), (19) ]
 *
```

CASE STORE 2: st_B-mem

```
(p1) R1_val(rd) = n1

(p2) R1_val(rs) = n1'

-------------------------------------------------------------------(st_B-mem)

(R1, C, M, ((seq(ns1,ns1'),(n1,n1')), st_B rd, rs)

      -->_0^(n1,n1')  (R1++, C, M[n1->n1'], seq(ns1,ns1') , .)




(4) Q2 = ((seq(ns2,ns2'),(n2,n2'))                          [ (3), (sim-Q) ]

(5) seq(ns1,ns1') sim_c seq(ns2,ns2')                       [ (sim-Q), (3) ]


Case on (R2_val(rd) = n2 and R1_val(rs) = n2') and then c =? G
```

SUB-CASE STORE 2.1: either rs/rd got zapped, or the end of the queue got zapped

```
(a5) R2_val(rd) =/= n2 or R2_val(rs) =/= n2'


(6) (R2,C,M,Q2, st_B rd, rs) -->_0 fault                  [ (st_B-mem-fail), (4), (a5) ]

(7) () prefix of (n1,n1')                                   [ duh ]

(8) S2 -->_0 fault and () prefix of (n1,n1')              [ (6), (7) ]
```

*

SUB-CASE STORE 2.2: zapped value was blue, but not rd/rs

```
(a5) R2_val(rd) = n2 and R2_val(rs) = n2'
(a6) c = B

(7)  G n1 sim_c G n2                                        [ (3), inversion on (sim-Q) ]
(8)  n1=n2                                                  [ (7), (a6), (sim-val) ]
(9)  G n1' sim_c G n2'                                      [ (3), inversion on (sim-Q) ]
(10) n1'=n2'                                                [ (9), (a5), (sim-val) ]
(11) M[n1->n1'] = M[n2->n2']                          [ (8), (10) ]

(12) (R1++,C,M[n1->n1'],seq(ns1,ns1'),.)                 [ (sim-S), (2), def of ++, (11), (5) ]
     sim_B (R2++,C,M[n2->n2'],seq(ns2,ns2'),.)
(13) (R2,C,M,Q2, st_B rd, rs)                               [ (st_B-mem), (a5) ]
      -->_0^(n2,n2') (R2++,C,M[n2->n2'],seq(ns2,ns2'),.)
(14) (n1,n1') = (n2,n2')                                    [ (8), (10) ]

(15) S2 -->_0^ss2 S2' and S1' sim_B S2' and ss1 = ss2            [ (13), (12), (14) ]
```

*

SUB-CASE STORE 2.2: zapped value was green, but not the end of the queue

```
(a5) R2_val(rd) = n2 and R2_val(rs) = n2'
(a6) c = G

(6) (n2,n2') sim_G (n1,n1)'                          [ (3), (sim-Q) ]
(7) n2=n1                                                   [ (a5), (p1) ]
(8) n2'=n1'                                                 [ (a5), (p2) ]

(9) M[n1->n1'] = M[n2->n2']                           [ (7), (8) ]

(10) (R1++,C,M[n1->n1'],seq(ns1,ns1'),.)                 [ (sim-S), (2), def of ++, (9), (5) ]
     sim_B (R2++,C,M[n2->n2'],seq(ns2,ns2'),.)
(11) (R2,C,M,Q2, st_B rd, rs)                               [ (st_B-mem), (a5) ]
      -->_0^(n2,n2') (R2++,C,M[n2->n2'],seq(ns2,ns2'),.)
(12) (n1,n1') = (n2,n2')                                    [ (7), (8) ]
```

```
(13) S2 -->_0^ss2 S2' and S1' sim_B S2' and ss1 = ss2                [ (11), (10), (12) ]
 *
```

CASE STORE 3: st_B-mem-fail

```
Q = (seq(n,n'),(n1,n1'))
Rval(rd) =/= n1  or Rval(rs) =/= n1'
------------------------------------------(st_B-mem-fail)
(R,C,M,Q, ld_B rd, rs )  -->_0  fault
```

```
(4) S1 -/->_0 fault                                                  [ (Progress 1), (a3) ]
(5) case does not apply
 *
```

CASE BRANCH 1: bz-untaken

```
(p1) R1_val(d) = 0      (p2) R1_val(rz) =/= 0
-----------------------------------------------------(bz-untaken)
(R1,C,M,Q1, bz_c' rz, rd )  -->_0  (R1++,C,M,Q1,.)
```

```
Case on c' =? B and c =? B
```

SUB-CASE BRANCH 1.1: bz_B and a green value was zapped

```
(a5) c' = B
(a6) c = G

(6) R1_col(d) = R1_col(rz) = B                                          [ (a3), inversion on (bz_B-t) ]
(7) R2_val(d) = R1_val(d) and R2_val(rz) = R1_val(rz)                   [ (2), (sim-val), (6), (a5), (a6) ]
(8) R2_val(d) = 0                                                               [ (p1), (7) ]
(9) R2_val(rz) =/= 0                                                            [ (p2), (7) ]
(10) (R2,C,M,Q2, bz_B rz, rd )  -->_0  (R2++,C,M,Q1,.)                     [ (bz-untaken), (8), (9) ]

(11) (R1++,C,M,Q1,.) sim_G (R2++,C,M,Q1,.)                                      [ (2), (3), def of ++ ]

(12) S2 -->_0 S2' and S1' sim_G S2' and () = ()                            [ (10), (11) ]
```

*

SUB-CASE BRANCH 1.2: bz_G and a green value was zapped

```
(a5) c' = G

(a6) c = G

(x) R2_col(rz) = G and R2_col(d) = G                                    [ (a3), inversion on (bz_G-t), (2) ]

Case on R2_val(rz) =? 0 and R2_val(d) =? 0

SUB-SUB-CASE BRANCH 1.2.1 - rz and d both consistent
(a7) R2_val(rz) =/= 0
(a8) R2_val(d) = 0
(4) (R2,C,M,Q2, bz_G rz, rd )  -->_0  (R2++,C,M,Q1,.)                        [ (bz-untaken), (a7), (a8) ]
(5) (R1++,C,M,Q1,.) sim_G (R2++,C,M,Q2,.)                                       [ (2), (3), def of ++ ]
(6) S2 -->_0 S2' and S1' sim_G S2' and () = ()                               [ (4), (5) ]
*

SUB-SUB-CASE BRANCH 1.2.2 - rz inconsistent
(a7) R2_val(rz) = 0
(a8) R2_val(d) = 0
(4) (R2,C,M,Q2, bz_G rz, rd )  -->_0  (R2++[d->R2(r_d)],C,M,Q1,.)        [ (bz_G-taken), (a7), (a8) ]
(5) R2_col(rd) = G                                                              [ (a3), inversion on (bz_G-t), (2), (a6) ]
(6) R1_col(d) = G                                                               [ (a3), inversion on (bz_G-t) ]
(7) R1(d) sim_G R2(r_d)                                                         [ (sim-val-zap), (5), (6) ]
(8) R1++ sim_G R2++[d->R2(r_d)]                                              [ (2), def of ++, (6) ]
(9) S2 -->_0 S2' and S1' sim_G S2' and () = ()                               [ (4), (8) ]
 *

SUB-SUB-CASE BRANCH 1.2.3 - d inconsistent
(a7) R2_val(rz) =/= 0
(a8) R2_val(d) =/= 0
(4) (R2,C,M,Q2, bz_G rz, rd )  -->_0  fault                           [ (bz-untaken-fail), (a7), (a8) ]
(5) S2 -->_0 fault and S1' and () = ()                                [ (4) ]
 *

SUB-SUB-CASE BRANCH 1.2.4 - both inconsistent
(a7) R2_val(rz) = 0
```

```
(a8) R2_val(d) =/= 0

(4) (R2,C,M,Q2, bz_G rz, rd )  -->_0  fault                                        [ (bz_G-taken-fail), (a7), (a8) ]

(5) S2 -->_0 fault and () = ()                                                     [ (4) ]

 *
```

SUB-CASE BRANCH 1.3: bz_B and and a blue value was zapped

```
(a5) c' = B

(a6) c = B


(x) R1_col(d) = G                                                     [ (a3), inversion on (bz_B-t) ]

(y) R1(d) = R2(d)                                                     [ (2), (x), (a6) ]

(z) R2_val(d) = 0                                                     [ (p1), (y) ]

(w) R2_col(rz) = B                                                    [ (a3), inversion on (bz_B-t), (2) ]


Case on R2_val(rz) ?= 0


SUB-SUB-CASE BRANCH 1.3.1 - rz  consistent

(a7) R2_val(rz) =/= 0

(4) (R2,C,M,Q2, bz_B rz, rd )  -->_0  (R2++,C,M,Q1,.)              [ (bz-untaken), (a7), (z) ]

(5) (R1++,C,M,Q1,.) sim_B (R2++,C,M,Q1,.)                             [ (2), (3), def of ++ ]

(6) S2 -->_0 S2' and S1' sim_G S2' and () = ()                     [ (4), (5) ]

 *


SUB-SUB-CASE BRANCH 1.3.2 - rz inconsistent

(a7) R2_val(rz) = 0

(4) (R2,C,M,Q2, bz_B rz, rd )  -->_0  fault                        [ (bz_B-taken-fail), (a7), (z) ]

(5) S2 -->_0 fault and () = ()                                     [ (4) ]

 *
```

SUB-CASE BRANCH 1.4: bz_G and a blue value was zapped

```
(a5) c' = G

(a6) c = B


(6) R1_col(d) = R1_col(rz) = G                                        [ (a3), inversion on (bz_G-t) ]

(7) R2_val(d) = R1_val(d) and R2_val(rz) = R1_val(rz)                 [ (2), (sim-val), (6), (a6) ]

(8) R2_val(d) = 0                                                     [ (p1), (7) ]

(9) R2_val(rz) =/= 0                                                  [ (p2), (7) ]
```

```
(10) (R2,C,M,Q2, bz_G rz, rd )  -->_0  (R2++,C,M,Q1,.)               [ (bz-untaken), (8), (9) ]


(11) (R1++,C,M,Q1,.) sim_B (R2++,C,M,Q1,.)                              [ (2), (3), def of ++ ]


(12) S2 -->_0 S2' and S1' sim_G S2' and () = ()                      [ (10), (11) ]
 *
```

CASE BRANCH 2: bz-untaken-fail

```
R1_val(rz) =/= 0    R1_val(d) =/= 0
-------------------------------------------(bz-untaken-fail)
(R1,C,M,Q, bz_c rz, rd )  -->_0  fault



(4) S1 -/->_0 fault                                                  [ (Progress 1), (a3) ]
(5) case does not apply
 *
```

CASE BRANCH 3: bz_G-taken

```
R1_val(d) = 0    R1_val(r_z) = 0    R1' = R1++[d -> R1(r_d)]
------------------------------------------------------------(bz_G-taken)
(R1,C,M,Q1, bz_G rz, rd )  -->_0  (R1',C,M,Q1,.)



Case on c =? B
```

SUB-CASE BRANCH 3.1: a blue value was zapped

```
(a5) c = B


(4) R1_col(d) = R1_col(r_z) = G                                         [ (a3), inversion on (bz_G-t) ]
(5) R1_val(d) = R2_val(d) and R1_val(r_z) = R2_val(r_z)                 [ (4), (2), (sim-val) ]
(6) R2_val(d) = 0                                                               [ (5), (p1) ]
(7) R2_val(r_z) = 0                                                             [ (5), (p2) ]


(8) (R2,C,M,Q2, bz_G r_z, r_d ) -->_0 (R2++[d -> R2(r_d)],C,M,Q2,.)       [ (bz_G-taken), (6), (7) ]
(9) R1(r_d) sim_B R2(r_d)                                                       [ (5), (sim-val) ]
(10) R1++[d -> R1(r_d)],C,M,Q1,.) sim_B R2++[d -> R2(r_d)],C,M,Q2,.)      [ (sim-Q), (2), (9), (3) ]
(11) S2 -->_0 S2' and S1' sim_c S2' and () = ()                              [ (8), (10) ]
```

*

SUB-CASE BRANCH 3.2: a green value was zapped

```
(a5) c = G

(x) R2_col(rz) = G and R2_col(d) = G                                    [ (a3), inversion on (bz_G-t), (2) ]

Case on R2_val(rz) =? 0 and R2_val(d) =? 0

SUB-SUB-CASE BRANCH 3.2.1 - rz and d both consistent
(a7) R2_val(rz) =/= 0
(a8) R2_val(d) = 0
(4) (R2,C,M,Q2, bz_G rz, rd )  -->_0  (R2++,C,M,Q1,.)                        [ (bz-untaken), (a7), (a8) ]
(5) (R1++,C,M,Q1,.) sim_G (R2++,C,M,Q1,.)                                      [ (2), (3), def of ++ ]
(6) S2 -->_0 S2' and S1' sim_G S2' and () = ()                               [ (4), (5) ]
*

SUB-SUB-CASE BRANCH 3.2.2 - rz inconsistent
(a7) R2_val(rz) = 0
(a8) R2_val(d) = 0
(4) (R2,C,M,Q2, bz_G rz, rd )  -->_0  (R2++[d->R2(r_d)],C,M,Q1,.)         [ (bz_G-taken), (a7), (a8) ]
(5) R2_col(rd) = G                                                             [ (a3), inversion on (bz_G-t), (2), (a6) ]
(6) R1_col(d) = G                                                              [ (a3), inversion on (bz_G-t) ]
(7) R1(d) sim_G R2(r_d)                                                        [ (sim-val-zap), (5), (6) ]
(8) R1++ sim_G R2++[d->R2(r_d)]                                              [ (2), def of ++, (6) ]
(9) S2 -->_0 S2' and S1' sim_G S2' and () = ()                               [ (4), (8) ]
 *

SUB-SUB-CASE BRANCH 3.2.3 - d inconsistent
(a7) R2_val(rz) =/= 0
(a8) R2_val(d) =/= 0
(4) (R2,C,M,Q2, bz_G rz, rd )  -->_0  fault                [ (bz-untaken-fail), (a7), (a8) ]
(5) S2 -->_0 fault and S1' and () = ()                     [ (4) ]
 *

SUB-SUB-CASE BRANCH 3.2.4 - both inconsistent
(a7) R2_val(rz) = 0
(a8) R2_val(d) =/= 0
```

```
(4) (R2,C,M,Q2, bz_G rz, rd )  -->_0  fault                [ (bz_G-taken-fail), (a7), (a8) ]

(5) S2 -->_0 fault and () = ()                              [ (4) ]

 *
```

CASE BRANCH 4: bz_G-taken-fail

```
Rval(rz) = 0    Rval(d) =/= 0

--------------------------------------------(bz_G-taken-fail)

(R,C,M,Q, bz_G rz, rd )  -->_0  fault



(4) S1 -/->_0 fault                                         [ (Progress 1), (a3) ]

(5) case does not apply

 *
```

CASE BRANCH 5: bz_B-taken

```
(p1) R1_val(d) =/= 0

(p2) R1_val(r_z) = 0

(p3) R1_val(r_d) = R1_val(d)

R1' = R1[pc_G -> R1(d)][pc_B -> R1(rd)][d -> G 0]

--------------------------------------------------(bz_B-taken)

(R1,C,M,Q1, bz_B rz, rd )  -->_0  (R1',C,M,Q1,.)


Case on c =? B
```

SUB-CASE BRANCH 5.1: a blue value was zapped

```
(a5) c = B


(4) G(r_z) = <B,int,E_z>                                   [ (a3), inversion on (bz_B-t) ]

(5) D |- E_z = E_z'                                         [ (a3), inversion on (bz_B-t) ]

(6) G(d) = E_z' = 0 => <G, T->void,E_r'>[ (a3), inversion on (bz_B-t) ]

(7) D |- E_z = R1_val(r_z)                                  [ (a3), inversion on (val-t) ]

(8) D |- E_z' = 0                                           [ (5), (7), (p2) ]

(9) G(d) = <G,T->void,E_r'>                                [ (6), (8) ]

(10) R1_col(d) = G                                          [ (9) ]

(11) R1(d) = R2(d)                                          [ (2), (sim-val), (10), (a5) ]

(12) R2_val(d) =/= 0                                        [ (11), (p1) ]
```

```
Case on R2_val(r_z) =? 0 and R2_val(r_d) =? R2_val(d)

SUB-SUB-CASE BRANCH 5.1.1 - r_z and r_d consistent

(a7) R2_val(r_z) = 0

(a8) R2_val(d) = R2_val(r_d)

(13) let R2' = R2[pc_G -> R2(d)][pc_B -> R2(rd)][d -> G 0]

(14) (R2,C,M,Q2, bz_B rz, rd )  -->_0  (R2',C,M,Q2,.)          [ (bz_B-taken), (12), (a7),(a8) ]

(15) R1(d) sim_B R2(d)                                            [ (11), (sim-val) ]

(16) R1(r_d) sim_B R2(r_d)                                        [ (2), (sim-val), (a8) ]

(17) R1' sim_B R2'                                                [ (sim-R), (15), (16) ]

(18) (R1',C,M,Q1,.) sim_G (R2',C,M,Q2,.)                          [ (2), (3), (17) ]

(19) S2 -->_0 S2' and S1' sim_G S2' and () = ()                 [ (14), (18) ]
 *


SUB-SUB-CASE BRANCH 5.1.2 - r_z inconsistent

(a7) R2_val(r_z) =/= 0

(a8) R2_val(d) = R2_val(r_d)

(13) (R2,C,M,Q2, bz_B rz, rd )  -->_0  fault                    [ (bz_G-untaken-fail), (a7), (12) ]

(14) S2 -->_0 fault and () = ()                                 [ (13) ]
 *


SUB-SUB-CASE BRANCH 5.1.3 - r_d inconsistent

(a7) R2_val(r_z) = 0

(a8) R2_val(d) =/= R2_val(r_d)

(13) (R2,C,M,Q2, bz_B rz, rd )  -->_0  fault                    [ (bz_B-taken-fail), (a7), (a8) ]

(14) S2 -->_0 fault and S1' and () = ()                         [ (13) ]
 *


SUB-SUB-CASE BRANCH 5.1.4 - both inconsistent

(a7) R2_val(r_z) =/= 0

(a8) R2_val(d) =/= R2_val(r_d)

(13) (R2,C,M,Q2, bz_B rz, rd )  -->_0  fault        [ (bz_G-untaken-fail), (a7), (12) ]

(14) S2 -->_0 fault and () = ()                     [ (13) ]
 *
```

SUB-CASE BRANCH 5.2: a green value was zapped

```
(a5) c = G
```

```
(4) R1_col(r_d) = R1_col(r_z) = B                                       [ (a3), inversion on (bz_B-t) ]

(5) R1_val(r_d) = R2_val(r_d) and R1_val(r_z) = R2_val(r_z)             [ (4), (2), (sim-val), (a5) ]

(6) R2_val(r_d) =/= 0                                                   [ (5), (p3), (p2) ]

(7) R2_val(r_z) = 0                                                     [ (5), (p2) ]


Case on R2_val(r_d) ?= R2_val(d)


SUB-SUB-CASE BRANCH 5.2.1 - R2_val(d) consistent

(a7) R2_val(r_d) = R2_val(d)

(8) R2_val(d) =/= 0                                                     [ (6), (a7) ]

(9) let R2' = R2[pc_G -> R2(d)][pc_B -> R2(r_d)][d -> G 0]

(10) (R2,C,M,Q2, bz_B rz, rd )  -->_0  (R2',C,M,Q2,.)                [ (bz_B-taken), (8) (7), (a7) ]

(11) R2_val(d) = R1_val(d)                                              [ (p3), (5), (a7) ]

(12) R2(d) sim_G R1(d)                                                  [ (sim-val), (11) ]

(13) R2(r_d) sim_G R1(r_d)                                              [ (sim-val), (5) ]

(14) R1' sim_G R2'                                                      [ (2), (11), (13), (sim-R) ]

(15) (R1',C,M,Q1,.) sim_G (R2',C,M,Q2,.)                                [ (sim-S), (14), (3) ]

(16) S2 -->_0 S2' and S1' sim_G S2' and () = ()                              [ (4), (5) ]
 *


SUB-SUB-CASE BRANCH 5.2.2 - R2_val(d) inconsistent

(a7) R2_val(r_d) =/= R2_val(d)

(4) (R2,C,M,Q2, bz_B rz, rd )  -->_0  fault                          [ (bz_B-taken-fail), (a7), (7) ]

(5) S2 -->_0 fault and () = ()                                       [ (4) ]
 *
```

CASE BRANCH 6: bz_B-taken-fail

```
Rval(rz) = 0

Rval(rd) =/= Rval(d)  or  Rval(d) = 0

--------------------------------------------(bz_B-taken-fail)

(R,C,M,Q, bz_B rz, rd )  -->_0  fault



(4) S1 -/->_0 fault                                                  [ (Progress 1), (a3) ]

(5) case does not apply
 *
```

CASE JUMP 1: jmp_G

```
(p1) R1_val(d) = 0     R1' = R1++[d -> R1(r_d)]
------------------------------------------------(jmp_G)
(R1,C,M,Q1, jmp_G rd )  -->_0  (R1',C,M,Q1,.)



Case on c =? B
```

SUB-CASE JUMP 1.1: a blue value was zapped

```
(a5) c = B

(4) R1_col(d) = G                                                         [ (a3), inversion on (jmp_G-t) ]
(5) R1(d) = R2(d)                                                         [ (2), (sim-R), (sim-val), (4) ]
(6) R2_val(d) = 0                                                         [ (p1), (5) ]

(7) let R2' = R2++[d -> R2(r_d)]
(8) (R2,C,M,Q2, jmp_G rd )  -->_0  (R2',C,M,Q2,.)                    [ (jmp_G), (6) ]

(9)  R1_col(r_d) = G                                                      [ (a3), inversion on (jmp_G-t) ]
(10) R1(r_d) = R2(r_d)                                                    [ (2), (sim-R), (sim-val), (4) ]
(11) R1' sim_B R2'                                                        [ (2), def of ++, (10), (sim-R) ]

(12) (R1',C,M,Q1,.) sim_B (R2',C,M,Q2,.)                                  [ (11), (3) ]

(13) S2 -->_0 S2' and S1' sim_B S2' and () = ()                                [ (8), (12) ]
 *
```

SUB-CASE JUMP 1.2: a green value was zapped

```
(a5) c = G

Case on R2_val(d) ?= 0

SUB-SUB-CASE JUMP 1.2.1 - R2_val(d) consistent
(a6) R2_val(d) = 0
(4) let R2' = R2++[d -> R2(r_d)]
(5) (R2,C,M,Q2, jmp_G r_d )  -->_0  (R2',C,M,Q2,.)                   [ (jmp_G), (a6) ]
(6) R1_col(d) = G                                                         [ (a3), inversion on (jmp_G-t) ]
(7) R1(d) sim_G R2(d)                                                     [ (sim-val), (6) ]
(8) R1' sim_G R2'                                                         [ (2), def of ++, (7) ]
```

```
(9) (R1',C,M,Q1,.) sim_G (R2',C,M,Q2,.)                          [ (sim-S), (8), (3) ]

(10) S2 -->_0 S2' and S1' sim_G S2' and () = ()               [ (5), (9) ]

 *
```

```
SUB-SUB-CASE JUMP 1.2.2 - R2_val(d) inconsistent

(a6) R2_val(d) =/= 0

(4) (R2,C,M,Q2, jmp_G r_d )  -->_0 fault                      [ (jmp_G-fail), (a6) ]

(5) S2 -->_0 fault and () = ()                                        [ (4) ]

 *
```

CASE JUMP 2: jmp_G-fail

```
Rval(d) =/= 0

-------------------------------------------(jmp_G-fail)

(R,C,M,Q, jmp_G rd )  -->_0  fault
```

```
(4) S1 -/->_0 fault                                           [ (Progress 1), (a3) ]

(5) case does not apply

 *
```

CASE JUMP 3: jmp_B

```
(p1) R1_val(d) =/= 0

(p2) R1_val(r_d) = R1_val(d)

R1' = R1[pc_G -> R1(d)][pc_B -> R1(r_d)][d -> G 0]

---------------------------------------------------(jmp_B)

(R1,C,M,Q1, jmp_B rd )  -->_0  (R1',C,M,Q1,.)
```

```
Case on c =? B
```

SUB-CASE JUMP 3.1: a blue value was zapped

```
(a5) c = B

(4) R1_col(d) = G                                        [ (a3), inversion on (jmp_B-t) ]

(5) R1(d) = R2(d)                                        [ (2), (sim-val), (a5), (4) ]

(6) R2_val(d) =/= 0


(7) R1_col(r_d) = B                                      [ (a3), inversion on (jmp_B-t) ]
```

```
Case on R2_val(r_d) =? R2_val(d)


SUB-SUB-CASE JUMP 3.1.1 - R2_val(r_d) consistent

(a6) R2_val(r_d) = R2_val(d)

(8) let R2' = R2[pc_G -> R2(d)][pc_B -> R2(r_d)][d -> G 0]

(9) (R2,C,M,Q2,jmp_B rd ) -->_0 (R2',C,M,Q2',.)              [ (jmp_B), (6), (a6), (8) ]

(10) R1_col(r_d) = B                                            [ (a3), inversion on (jmp_G-t) ]

(11) R1(r_d) sim_B R2(r_d)                                      [ (sim-val), (10) ]

(12) R1' sim_B R2'                                              [ (sim-R), (2), (5), (11) ]

(13) (R1',C,M,Q1',.) sim_B (R2',C,M,Q2',.)                      [ (sim-S), (12), (3) ]

(14) S2 -->_0 S2' and S1' sim_B S2' and () = ()                      [ (9), (13) ]

 *


SUB-SUB-CASE JUMP 3.1.2 - R2_val(r_d) inconsistent

(a6) R2_val(r_d) =/= R2_val(d)

(8) (R2,C,M,Q2,jmp_B rd ) -->_0 fault                         [ (jmp_B-fail), (a6) ]

(9) S2 --> fault and () = ()                                  [ (8) ]

 *
```

SUB-CASE JUMP 3.2: a green value was zapped

```
(a5) c = G

(4) R1_col(r_d) = B                                             [ (a3), inversion on (jmp_G-t) ]

(5) R1(r_d) = R2(r_d)                                           [ (2), (sim-val), (4), (a5) ]


(6) R1_col(d) = G                                               [ (a3), inversion on (jmp_G-t) ]


Case on R2_val(d) =? R2_val(r_d)


SUB-SUB-CASE JUMP 3.1.1 - R2_val(d) consistent

(a6) R2_val(d) = R2_val(r_d)

(7) R2_val(d) =/= 0                                             [ (a6), (5), (p2), {p1) ]

(8) let R2' = R2[pc_G -> R2(d)][pc_B -> R2(r_d)][d -> G 0]

(9) (R2,C,M,Q2,jmp_B rd ) -->_0 (R2',C,M,Q2',.)              [ (jmp_B), (7), (a6), (8) ]

(10) R1(d) = R2(d)                                              [ (a6), (5), (p2) ]

(11) R1' sim_G R2'                                              [ (2), (sim-R), (10), (a6) ]

(12) S2 -->_0 S2' and S1' sim_G S2' and () = ()              [ (9), (11) ]

 *


SUB-SUB-CASE JUMP 3.1.2 - R2_val(d) inconsistent
```

```
(a6) R2_val(d) =/= R2_val(r_d)

(7) (R2,C,M,Q2,jmp_B rd ) -->_0 fault                              [ (jmp_B-fail), (a6) ]

(8) S2 -->_0 fault and () = ()                                     [ (7) ]

 *
```

CASE JUMP 4: jmp_B-fail

```
Rval(rd) =/= Rval(d)  or  Rval(d) = 0

--------------------------------------------(jmp_B-fail)

(R,C,M,Q, jmp_B rd )  -->_0  fault
```

```
(4) S1 -/->_0 fault                                                [ (Progress 1), (a3) ]

(5) case does not apply

 *
```

Fault Tolerance Theorem and Associated Lemmas

Notes:
`S -->_k^s S'` is my textual representation for the usual single step rule: "Machine State S takes a step with k faults and output s to state S' "

Multistep Dynamic Semantics Definition:

`S -n->_k^ss S'` is a sequence of n steps with k faults resulting in an output sequence ss.

```
--------------(multi-single)

S -0->_0^() S
```

```
S -->_k1^s1 S''    S'' -(n-1)->_k2^ss2  S'

--------------------------------------------(multi-compose)

S -n->_(k1+k2)^(s1,ss2) S'
```

Multistep Split Lemma

If `S -n->_0^ss S'` then Exists `n1,n2,S'',ss1,ss2` such that `n = n1 + n2` and `S -n1->_0^ss1 S''` and `S'' -n2->_0^ss2 S'` and `ss = (ss1,ss2)`

"If a machine state evaluates in a sequence of steps with no faults to a final state, then this computation can be divided into a sequence of non-faulty steps reaching an intermediate state, and a sequence of non-faulty steps from this intermediate state to the final state."

Proof: by induction on `S -n->_0^ss S'` (omitted)

Mulitstep Faulty Combine Lemma

If `S -n1->_0^ss1 S'` and `S' -->_1 Sf'` and `Sf' -n2->_0^ss2 S''` then `S -(n1+1+n2)->_1^(ss1,ss2) S''`

"If a machine state evaluates in a sequence of n1 non-faulty steps another state, that state faults to a third state, and the third state evaluates in a sequence of n2 non-faulty steps to a final state, then the original state can reach the faulty state in a sequence of (n1+1+n2) steps including one faulty step."

Proof: by induction on `S -n->_k^ss S'` (omitted)

Multistep Fault Detection Lemma:

If `|- S1` and `S1 sim_c S2` and `S1 -n->_0^ss1 S1'` then either

1. `S2 -n->_0^ss2 S2'` and `S1' sim_c S2'` and `ss1=ss2`
2. Exists m <= n. `S2 -m->_0^ss2 fault` and ss2 is a prefix of ss1

"If a fault has occurred, then either it will eventually result in a fault state or the output will continue to be indistinguishable from the non-faulty case."

Proof: By induction on the structure of `S1 -n->_0^ss1 S1'`

//FJP -- need to modify so that singlestep is (simulates & output equal) or (reaches fault & output empty)
CASE 1: multi-single

```
------------------(multi-single)

S1 -0->_0^() S1



(a1) |- S1

(a2) S1 sim_c S2

(a3) S1 -n->_0^() S1


MP: S2 -0->_0^() S2 and S1 sim_c S2 and () = ()


(1) S2 -0->_0^() S2                                                        [ (multi-single) ]

(2) S2 -0->_0^() S2 and S1 sim_c S2 and () = ()                            [ (1), (a2), obvious ]
 *
```

CASE 2: multi-compose

```
(p1) S1 -->_0^s1 S1''     (p2) S1'' -(n-1)->_0^ss1  S1'

-----------------------------------------------------------(multi-compose)

S1 -n->_0^(s1,ss1) S1'




(a1) |- S1

(a2) S1 sim_c S2

(a3) S1 -n->_0^ss1 S1'


(1) S2 -->_0^s2 S2''                                                       [ Singlestep Fault Detection Lemma, (a1), (a2), (p1) ]

(2) s1 = s2
```

```
(3) S1'' sim_c S2''  or  S2'' = fault


(4) |- S1''                                                           [ Preservation Part 1, (a1), (p1) ]
```

SUB-CASE 2.1: ***fault does not occur in first step***

```
(a4) S1'' sim_c S2''                                                  [ (3) ]

(a5) S2 =/= fault


(5) either                                                            [ IH, (4), (3), (p2) ]

    S2'' -(n-1)->_0^ss2 S2'  and S1' sim_c S2' and ss1=ss2

    or

    Exists m2 <= (n-1). S2'' -m2->_0^ss2 fault and ss2 prefixof ss1
```

SUB-SUB-CASE 2.1.1: fault never occurs

```
(a6) S2'' -(n-1)->_0^ss2 S2'                                          [ (5) ]

(a7) S1' sim_c S2'

(a8) ss1=ss2


MP: S2 -n->_0^(s2,ss2) S2' and S1' sim_c S2' and (s1,ss1)=(s2,ss2)


(6) S2 -n->_0^(s2,ss2) S2'                                            [ (multi-compose), (1), (a6) ]

(7) (s2,ss2) = (s1,ss1)                                               [ (2), (a8) ]

(8) S2 -n->_0^(s2,ss2) S2' and S1' sim_c S2' and (s1,ss1)=(s2,ss2)    [ (8), (a7), (7) ]
 *
```

SUB-SUB-CASE 2.1.2: fault occurs during remainder of execution

```
(a6) Exists m2 <= n-1                                                 [ (5) ]

(a7) S2'' -m2->_0^ss2 fault

(a8) ss2 prefixof ss1


MP: Exists m <= n. S2 -m->_0^(s2,ss2) fault and (s2,ss2) prefixof (s1,ss1)


(6) S2 -(m2+1)->_0^(s2,ss2) fault                                     [ (multi-compose), (1), (a7) ]

(7) m2 + 1 <= n                                                       [ (a6) ]

(8) (s2,ss2) prefixof (s1,ss1)                                        [ (2), (a8) ]
```

```
(9) S2 -(m2+1)->_0^(s2,ss2) fault and (s2,ss2) prefix (s1,ss1)        [ (6), (7), (8) ]
 *
```

SUB-CASE 2.2: fault occurs during first step

```
(a6) S2'' = fault

MP: Exists m <= n. S2 -m->_0^(s2,ss2) fault and (s2) prefix (s1,ss1)

(5) fault -0->_0^() fault                                             [ (multi-single) ]
(6) S2 -1->_0^(s2) fault                                              [ (multi-compose), (1), (5) ]
(7) 1 <= n                                                            [ (mulit-compose) ]
(8) (s2,()) prefix of (s1,ss1)                                        [ (2) ]
(8) S2 -1->_0^(s2) fault and (s2) prefix (s1,ss1)                     [ (6), (7), (8) ]
 *
```

Fault Similarity Lemma:

If `S -->_1 Sf` then Exists c. `S sim_c Sf`

"If there is a faulty step, the machine states before and after the fault are similar with regard to the fault color."

Proof: By case analysis on the definition of `S -->_1 Sf`

CASE 1: reg-zap

```
(p1) R(a) = c n
------------------------------------------------ (reg-zap)
(R,C,M,Q,ir) -->_1 (R[a -> c n'],C,M,Q,ir)

(1) c n sim_c c n'                                       [ (sim-val-zap) ]
(2) Forall a' in R. Rval(a') sim_c Rval(a')              [ (sim-val) ]
(3) R sim_c R[a -> c n']                                 [ (sim-R), (1), (p1), (2), handwave ]
(4) (R,C,M,Q,ir) sim_c (R[a -> c n'],C,M,Q,ir)           [ (sim-S), (3), (sim-Q-Z) ]
 *
```

CASE 2: Q1-zap

```
(p1) Q1 = ( seq(n1,n1'), (m1, m2), seq(n2,n2') )

(p2) Q2 = ( seq(n1,n1'), (m1',m2), seq(n2,n2') )

------------------------------------------------ (Q1-zap)

(R,C,M,Q1,ir) -->_1 (R,C,M,Q2,ir)


(1) let c = G

(2) Q1 sim_G Q2                                                  [ (sim-Q-G) ]

(3) R sim_G R                                                    [ (sim-val), Color Weakening Lemma ]

(3) (R,C,M,Q1,ir) sim_G (R,C,M,Q2,ir)                            [ (sim-S), (2), (3) ]
 *
```

CASE 3: Q2-zap

```
(p1) Q1 = ( seq(n1,n1'), (m1,m2),  seq(n2,n2') )

(p2) Q2 = ( seq(n1,n1'), (m1,m2'), seq(n2,n2') )

------------------------------------------------ (Q2-zap)

(R,C,M,Q1,ir) -->_1 (R,C,M,Q2,ir)


(1) let c = G

(2) Q1 sim_G Q2                                                  [ (sim-Q-G) ]

(3) R sim_G R                                                    [ (sim-val), Color Weakening Lemma ]

(3) (R,C,M,Q1,ir) sim_G (R,C,M,Q2,ir)                            [ (sim-S), (2), (3) ]
 *
```

Fault Tolerance Theorem:

If `|- S` and `S -n->_0^ss S'` then either

1. Exists m <= n+1. `S -m->_1^ssf fault` and ssf is a prefix of ss
2. `S -(n+1)->_1^ssf Sf` and Exists c. `S sim_c Sf` and ss = ssf

"Faulty computation is equivalent to non-faulty computation up until the point where a fault state is reached (if it is reached at all)."

Proof: By case analysis on the definition of `S -n->_k^ss S'`

CASE: multi-single

```
--------------(multi-single)
```

```
S -0->_0^() S


(a1) |- S

(a2) S -0->_0^() S'

MP case 2: S -1->_1^() Sf and Exists c. S sim_c Sf and () = ()


(1) S -->_1 Sf                                                 [ def of -->_1 ]

(2) Exists c. S sim_c Sf                                       [ Fault Similarity Lemma, (1) ]

(3) () = ()                                                    [ def of -->_1 ]

(4) S -1->_1^() Sf and Exists c. S sim_c Sf and () = ()         [ (1), (2), (3) ]
 *
```

CASE: multi-compose

```
(p1) S -->_0^s1 S''    (p2) S'' -(n-1)->_0^ss2  S'    (s1) ss = (s1,ss2)
-------------------------------------------------------------------------(multi-compose)
S -n->_0^(ss) S'



(a1) |- S

(a2) S -n->_0^ss S'



(1) n = n1 + n2                                                [ Multistep Split Lemma, (a2) ]

(2) S -n1->_0^ssa Sn1

(3) Sn1 -n2->_0^ssb S'

(5) ss = (ssa,ssb)


(6) Sn1 -->_1 Sn1f                                             [ by inspection of def of -->_1 ]

(7) Exists c. Sn1 sim_c Sn1f                                   [ Lemma Fault-Similarity, (6) ]


(8) |- Sn1                                                     [ (a1), Preservation, (2), handwave ]


(9) either                                                     [ Lemma Multistep-Fault-Detection, (8),(7),(3) ]
    (a) Sn1f -n2->_0^ssbf Sf'  and  S' sim_c Sf' and ssbf = ssb
    or
    (b) Exists m2<=n2.  Sn1f -m2->_0^ssbf fault  and  ssbf is a prefix of ssb
```

SUB-CASE: fault not reached yet

```
MP: S -(n+1)->_1^ssf Sf'  and  Exists c. S sim_c Sf'  and  ss = ssf


(a4) Sn1f -n2->_0^ssbf Sf'

(a5) S' sim_c Sf'

(a6) ssbf = ssb


(10) let ssf = (ssa,ssbf)

(11) S -(n+1)->_1^ssf Sf'                                [ Multistep Combine Lemma, (2), (6), (a4) ]

(12) ss = ssf                                            [ (5), (10), (a6) ]


(13) S -(n+1)->_1^ssf Sf'  and  S' sim_c Sf'  and  ss = ssf     [ (a4), (a5), (12) ]
 *
```

SUB-CASE: fault reached

```
MP: Exists m<=n+1.  S -m->_0^ssf fault  and  ssf is a prefix of ss


(a4) Exists m2 <= n2

(a5) Sn1f -m2->_0^ssbf fault

(a6) ssbf is a prefix of ssb


(10) let m = n1 + 1 + m2

(11) m <= n+1                                            [ (1), (a4), (10) ]

(12) let ssf = (ssa,ssbf)

(13) S -m->_1^ssf  fault                                 [ Multistep Combine Lemma, (2), (6), (a5) ]

(14) ssf is a prefix of ss                               [ (12), (5), (a6) ]


(15) m <= n+1 and S -m->_1^ssf fault and ssf prefix of ss       [ (13), (14), (11) ]
 *
```

Fault-tolerant Typed Assembly Language

Extended Rules for the Translation

$$
\begin{array}{llcl}
colors & c & ::= & G \mid B \\
colored\ values & v & ::= & n \\
registers & r & ::= & r_n \\
general\ regs & a & ::= & r \mid d \mid pc_c \mid sp_c \\
register\ file & R & ::= & \cdot \mid R, a \rightarrow n \\
code\ memory & C & ::= & \cdot \mid C, n \rightarrow i \\
value\ memory & M & ::= & \cdot \mid M, n \rightarrow n \\
store\ queue & Q & ::= & \overline{(n, n)} \\
ALU\ ops & op & ::= & add \mid sub \mid mul \\
instructions & i & ::= & op\ r_d, r_s, r_t \mid op\ r_d, r_s, n \mid mov\ r_d, n \mid mov\ r_d, r_s \\
 & & \mid & ld_c\ r_d, r_s \mid sld_c\ r_d\ n \mid st_c\ r_d, r_s \mid sst\ n\ r_v \\
 & & \mid & bz_c\ r_z, r_d \mid jmp_c\ r_d \\
 & & \mid & malloc[b]\ r_g,\ r_b \mid salloc\ n \mid sfree\ n \\
inst\ register & ir & ::= & i \mid \cdot \\
state & \Sigma & ::= & (R, C, M, Q, ir) \mid fault
\end{array}
$$

Figure 1. Syntax of FM states

$$
\begin{array}{lcl}
sld_c\ r_d\ n & \equiv & add\ r\ sp_c\ n;\ ld_G\ r_d\ r; \\
sst\ n\ r_v & \equiv & add\ r\ sp_G\ n;\ st_G\ r\ r_v;\ add\ r\ sp_B\ n;\ st_B\ r\ r_v; \\
salloc\ n & \equiv & sub\ sp_G\ sp_G\ n;\ sub\ sp_B\ sp_B\ n; \\
sfree\ n & \equiv & add\ sp_G\ sp_G\ n;\ add\ sp_B\ sp_B\ n;
\end{array}
$$

Figure 2. Expanded Stack Instructions

$$
\frac{R(a) = n}{(R, C, M, Q, ir) \longrightarrow_1 (R[a \mapsto n'], C, M, Q, ir)}\ (\textit{reg-zap})
$$

$$
\frac{\begin{array}{c} Q_1 = \overline{(n_1, n_1')}, (m_1, m'), \overline{(n_2, n_2')} \\ Q_2 = \overline{(n_1, n_1')}, (m_2, m'), \overline{(n_2, n_2')} \end{array}}{(R, C, M, Q_1, ir) \longrightarrow_1 (R, C, M, Q_2, ir)}\ (Q_1\textit{-zap})
$$

$$
\frac{\begin{array}{c} Q_1 = \overline{(n_1, n_1')}, (m, m_1'), \overline{(n_2, n_2')} \\ Q_2 = \overline{(n_1, n_1')}, (m, m_2'), \overline{(n_2, n_2')} \end{array}}{(R, C, M, Q_1, ir) \longrightarrow_1 (R, C, M, Q_2, ir)}\ (Q_2\textit{-zap})
$$

Figure 3. Fault Rules

Instruction Fetch:

$$\frac{R(pc_G) = R(pc_B) \quad R(pc_G) \in \ Dom(C)}{(R, C, M, Q, \cdot) \longrightarrow_0 (R, C, M, Q, C(R(pc_G)))} \ (\textit{fetch})$$

$$\frac{R(pc_G) \neq R(pc_B)}{(R, C, M, Q, \cdot) \longrightarrow_0 \ fault} \ (\textit{fetch-fail})$$

Basic Instructions:

$$\frac{R' = R\text{++}[r_d \mapsto R(r_s) \ op \ R(r_t))]}{(R, C, M, Q, op \ \ r_d, r_s, r_t) \longrightarrow_0 (R', C, M, Q, \cdot)} \ (\textit{op2r})$$

$$\frac{R' = R\text{++}[r_d \mapsto R(r_s) \ op \ n]}{(R, C, M, Q, op \ \ r_d, r_s, n) \longrightarrow_0 (R', C, M, Q, \cdot)} \ (\textit{op1r})$$

$$\frac{R' = R\text{++}[r_d \mapsto n]}{(R, C, M, Q, mov \ r_d \ n) \longrightarrow_0 (R', C, M', Q, \cdot)} \ (\textit{mov-n})$$

$$\frac{R' = R\text{++}[r_d \mapsto R(r_s)]}{(R, C, M, Q, mov \ r_d \ r_s) \longrightarrow_0 (R', C, M', Q, \cdot)} \ (\textit{mov-reg})$$

Figure 4. Operational rules for basic instructions

$\boxed{\Sigma \longrightarrow_k^s \Sigma'}$

$$\frac{\begin{array}{c} n = max(Dom(M)) + 1 \\ R' = R{+}{+}[r_g \mapsto n][r_b \mapsto n] \end{array}}{(R, C, M, Q, malloc[b]\ r_g\ r_b) \longrightarrow_0 (R', C, (M, n \mapsto 0), Q, \cdot)} \ (\textit{malloc})$$

$$\frac{\begin{array}{c} R' = R{+}{+}[sp_G \mapsto R(sp_G) - n][sp_B \mapsto R(sp_B) - n] \\ m = min(Dom(M)) \\ M' = (M, m-1 \mapsto 0, ..., m-n \mapsto 0) \end{array}}{(R, C, M, Q, salloc\ n) \longrightarrow_0 (R', C, M', Q, \cdot)} \ (\textit{salloc})$$

$$\frac{\begin{array}{c} R' = R{+}{+}[sp_G \mapsto R(sp_G) + n][sp_B \mapsto R(sp_B) + n] \\ m = min(Dom(M)) \\ M = M', m \mapsto v_m, ..., (m+n-1) \mapsto v_1 \end{array}}{(R, C, M, Q, sfree\ n) \longrightarrow_0 (R', C, M', Q, \cdot)} \ (\textit{sfree})$$

$$\frac{Q' = ((R(r_d), R(r_s)), Q)}{(R, C, M, Q, st_G\ \ r_d, r_s) \longrightarrow_0 (R{+}{+}, C, M, Q', \cdot)} \ (\textit{st}_G\textit{-queue})$$

$$\frac{R(rd) = n_1 \qquad R(r_s) = n_1'}{(R, C, M, (\overline{(n, n')}, (n_l, n_l')), st_B\ \ r_d, r_s) \longrightarrow_0^{(n_l, n_l')} (R{+}{+}, C, M[n_l \mapsto n_l'], \overline{(n, n')}, \cdot)} \ (\textit{st}_B\textit{-mem})$$

$$\frac{Q = (\overline{(n, n')}, (n_l, n_l')) \qquad R(r_d) \neq n_l \textit{ or } R(r_s) \neq n_l'}{(R, C, M, Q, st_B\ \ r_d, r_s) \longrightarrow_0 \ fault} \ (\textit{st}_B\textit{-mem-fail})$$

$$\frac{R(sp_B) = R(sp_G) \qquad R(sp_G) + n \in Dom(M)}{(R, C, M, Q, sst\ n,\ r_v) \longrightarrow_0^{(R(sp_G)+n, R(r_v))} (R{+}{+}, C, M[R(sp_G) + n \mapsto R(r_v)], Q, \cdot)} \ (\textit{sst})$$

$$\frac{R(sp_G) \neq R(sp_B) \textit{ or } R(sp_B) + n \notin Dom(M)}{(R, C, M, Q, sst\ n,\ r_v) \longrightarrow_0 \ fault} \ (\textit{sst-fail})$$

Figure 5. Operational rules for malloc and store instructions.

$$\boxed{\Sigma \longrightarrow^{s}_{k} \Sigma'}$$

$$\frac{\begin{array}{c} find(Q, R(r_s)) = (R(r_s), n) \\ R' = R{+}{+}[r_d \mapsto n] \end{array}}{(R, C, M, Q, ld_G\ \ r_d, r_s) \longrightarrow_0 (R', C, M, Q, \cdot)}\ (ld_G\text{-}queue)$$

$$\frac{\begin{array}{c} find(Q, R(r_s)) = () \\ R(r_s) \in\ Dom(M) \\ R' = R{+}{+}[r_d \mapsto M(R(r_s))] \end{array}}{(R, C, M, Q, ld_G\ \ r_d, r_s) \longrightarrow_0 (R', C, M, Q, \cdot)}\ (ld_G\text{-}mem)$$

$$\frac{\begin{array}{c} R(r_s) \in\ Dom(M) \\ R' = R{+}{+}[r_d \mapsto M(R(r_s))] \end{array}}{(R, C, M, Q, ld_B\ \ r_d, r_s) \longrightarrow_0 (R', C, M, Q, \cdot)}\ (ld_B\text{-}mem)$$

$$\frac{\begin{array}{c} find(Q, R(r_s)) = () \\ R(r_s) \notin\ Dom(M) \end{array}}{(R, C, M, Q, ld_G\ \ r_d, r_s) \longrightarrow_0\ fault}\ (ld_G\text{-}fail)$$

$$\frac{R(r_s) \notin\ Dom(M)}{(R, C, M, Q, ld_B\ \ r_d, r_s) \longrightarrow_0\ fault}\ (ld_B\text{-}fail)$$

$$\frac{\begin{array}{c} find(Q, R(r_s)) = () \\ R(r_s) \notin\ Dom(M) \\ R' = R{+}{+}[r_d \mapsto n] \end{array}}{(R, C, M, Q, ld_G\ \ r_d, r_s) \longrightarrow_0 (R', C, M, Q, \cdot)}\ (ld_G\text{-}rand)$$

$$\frac{\begin{array}{c} R(r_s) \notin\ Dom(M) \\ R' = R{+}{+}[r_d \mapsto n] \end{array}}{(R, C, M, Q, ld_B\ \ r_d, r_s) \longrightarrow_0 (R', C, M, Q, \cdot)}\ (ld_B\text{-}rand)$$

$$\frac{\begin{array}{c} R(sp_c) + n \in Dom(M) \\ R' = R{+}{+}[r_d \mapsto M(R(sp_c) + n)] \end{array}}{(R, C, M, Q, sld_c\, r_d,\ n) \longrightarrow_0 (R', C, M, Q, \cdot)}\ (sld_c)$$

$$\frac{R(sp_c) + n \notin Dom(M)}{(R, C, M, Q, sld_c\, r_d,\ n) \longrightarrow_0 fault}\ (sld_c\text{-}fail)$$

Figure 6. Operational rules for load instructions.

$\boxed{\Sigma \longrightarrow_k^s \Sigma'}$

$$\frac{R(d) = 0 \qquad R' = R\text{++}[d \mapsto R(r_d)]}{(R, C, M, Q, jmp_G\ \ r_d) \longrightarrow_0 (R', C, M, Q, \cdot)}\ (\textit{jmp}_G)$$

$$\frac{R(d) \neq\ 0}{(R, C, M, Q, jmp_G\ \ r_d) \longrightarrow_0\ \ fault}\ (jmp_G\textit{-fail})$$

$$\frac{\begin{array}{c} R(d) \neq\ 0 \qquad R(r_d) = R(d) \\ R' = R[pc_G \mapsto R(d)][pc_B \mapsto R(r_d)][d \mapsto 0] \end{array}}{(R, C, M, Q, jmp_B\ \ r_d) \longrightarrow_0 (R', C, M, Q, \cdot)}\ (\textit{jmp}_B)$$

$$\frac{R(r_d) \neq R(d)\ \ or\ R(d) = 0}{(R, C, M, Q, jmp_B\ \ r_d) \longrightarrow_0\ \ fault}\ (jmp_B\textit{-fail})$$

$$\frac{R(d) = 0 \qquad R(r_z) \neq 0}{(R, C, M, Q, bz_c\ \ r_z, r_d) \longrightarrow_0 (R\text{++}, C, M, Q, \cdot)}\ (\textit{bz-untaken})$$

$$\frac{\begin{array}{c} R(r_z) \neq\ 0 \\ R(d) \neq\ 0 \end{array}}{(R, C, M, Q, bz_c\ \ r_z, r_d) \longrightarrow_0\ \ fault}\ (\textit{bz-untaken-fail})$$

$$\frac{\begin{array}{c} R(d) = 0 \qquad R(r_z) = 0 \\ R' = R\text{++}[d \mapsto R(r_d)] \end{array}}{(R, C, M, Q, bz_G\ \ r_z, r_d) \longrightarrow_0 (R', C, M, Q, \cdot)}\ (\textit{bz}_G\textit{-taken})$$

$$\frac{\begin{array}{c} R(r_z) = 0 \\ R(d) \neq\ 0 \end{array}}{(R, C, M, Q, bz_G\ \ r_z, r_d) \longrightarrow_0\ \ fault}\ (\textit{bz}_G\textit{-taken-fail})$$

$$\frac{\begin{array}{c} R(d) \neq 0 \qquad R(r_z) = 0 \\ R(r_d) = R(d) \\ R' = R[pc_G \mapsto R(d)][pc_B \mapsto R(r_d)][d \mapsto 0] \end{array}}{(R, C, M, Q, bz_B\ \ r_z, r_d) \longrightarrow_0 (R', C, M, Q, \cdot)}\ (\textit{bz}_B\textit{-taken})$$

$$\frac{\begin{array}{c} R(r_z) = 0 \\ R(r_d) \neq R(d)\ \ or\ R(d) = 0 \end{array}}{(R, C, M, Q, bz_B\ \ r_z, r_d) \longrightarrow_0\ \ fault}\ (\textit{bz}_B\textit{-taken-fail})$$

Figure 7. Operational rules for control flow instructions.

Static Expressions			
exp kinds	κ	$::=$	$\kappa_{int} \mid \kappa_{mem} \mid \kappa_{\sigma}$
exp contexts	Δ	$::=$	$\cdot \mid \Delta, x : \kappa$
exps	E	$::=$	$x \mid n \mid E \; op \; E \mid sel \; E_m \; E_n$
			$\mid emp \mid upd \; E_m \; E_{n_1} \; E_{n_2}$
substitutions	S	$::=$	$\cdot \mid S, E/x$
Types			
zap tags	Z	$::=$	$\cdot \mid c$
initialization flags	φ	$::=$	$1 \mid \frac{1}{2} \mid 0$
base types	b	$::=$	$int \mid \Theta \rightarrow void \mid b \; ref^{\varphi} \mid sptr$
reg types	t	$::=$	$\langle c, b, E \rangle \mid E' = 0 \Rightarrow \langle c, b, E \rangle \mid ns$
reg file types	Γ	$::=$	$\cdot \mid \Gamma, a \rightarrow t$
unlabeled stack	σ	$::=$	$sbase \mid \rho \mid t :: \varsigma$
labeled stack	ς	$::=$	$E : \sigma$
result types	RT	$::=$	$\Theta \mid void$
Contexts			
heap typing	Ψ	$::=$	$\cdot \mid \Psi, n : b$
static context	Θ	$::=$	$\Delta; \Gamma; \overline{(E_d, E_s)}; E_m; \varsigma$

Figure 8. $\mathrm{TAL}_{\mathrm{FT}}$ type syntax.

$\boxed{\Delta \vdash\ E : \kappa}$

$$\frac{x \in Dom(\Delta)}{\Delta \vdash\ x : \Delta(x)}\ (\textit{E-var-t})$$

$$\frac{}{\Delta \vdash\ n : \kappa_{int}}\ (\textit{E-int-t})$$

$$\frac{\begin{array}{c}\Delta \vdash\ E_1 : \kappa_{int}\\ \Delta \vdash\ E_2 : \kappa_{int}\end{array}}{\Delta \vdash\ E_1\ op\ E_2 : \kappa_{int}}\ (\textit{E-op-t})$$

$$\frac{\begin{array}{c}\Delta \vdash\ E_m : \kappa_{mem}\\ \Delta \vdash\ E_n : \kappa_{int}\end{array}}{\Delta \vdash\ sel\ E_m\ E_n : \kappa_{int}}\ (\textit{E-sel-t})$$

$$\frac{\begin{array}{c}\Delta \vdash\ E_m : \kappa_{mem}\\ \Delta \vdash\ E_{n_1} : \kappa_{int}\\ \Delta \vdash\ E_{n_2} : \kappa_{int}\end{array}}{\Delta \vdash\ upd\ E_m\ E_{n_1}\ E_{n_2} : \kappa_{mem}}\ (\textit{E-upd-t})$$

$$\frac{}{\Delta \vdash\ emp : \kappa_{mem}}\ (\textit{E-emp-t})$$

$\boxed{\Delta \vdash\ S : \Delta'}$

$$\frac{}{\Delta \vdash\ \cdot : \cdot}\ (\textit{sub-emp-t})$$

$$\frac{\begin{array}{c}\Delta \vdash\ S : \Delta'\\ \Delta \vdash\ E : \kappa\\ x \notin Dom(\Delta) \cup Dom(\Delta')\end{array}}{\Delta \vdash\ S, E/x : \Delta', x : \kappa}\ (\textit{sub-t})$$

$\boxed{[\![E]\!]}$

$$\begin{array}{lcl}
[\![n]\!] & = & n\\
[\![emp]\!] & = & \cdot\\
[\![E_1\ op\ E_2]\!] & = & [\![E_1]\!]\ op\ [\![E_2]\!]\\
[\![sel\ E_m\ E_n]\!] & = & [\![E_m]\!]([\![E_n]\!])\\
[\![upd\ E_m\ E_1\ E_2]\!] & = & [\![E_m]\!][\ [\![E_1]\!] \mapsto [\![E_2]\!]\ ]
\end{array}$$

$\boxed{\Delta \vdash\ E_1\ op\ E_2}$

$$\frac{\begin{array}{c}\Delta \vdash E_1 : \kappa_{int} \qquad \Delta \vdash E_2 : \kappa_{int}\\ \forall S.\ \cdot \vdash S : \Delta \implies [\![S(E_1)]\!] = [\![S(E_2)]\!]\end{array}}{\Delta \vdash\ E_1 = E_2}\ (\textit{E-eq})$$

$$\frac{\begin{array}{c}\Delta \vdash E_1 : \kappa_{int} \qquad \Delta \vdash E_2 : \kappa_{int}\\ \forall S.\ \cdot \vdash S : \Delta \implies [\![S(E_1)]\!] \neq [\![S(E_2)]\!]\end{array}}{\Delta \vdash\ E_1 \neq E_2}\ (\textit{E-neq})$$

$$\frac{\begin{array}{c}\Delta \vdash E_1 : \kappa_{mem} \qquad \Delta \vdash E_2 : \kappa_{mem}\\ \forall \ell \in\ Dom([\![S(E_1)]\!]) \cup Dom([\![S(E_2)]\!]).\ [\![S(E_1)]\!](\ell) = [\![S(E_2)]\!](\ell)\end{array}}{\Delta \vdash\ E_1 = E_2}\ (\textit{E-mem-eq})$$

Figure 9. Properties of Static Expressions

$\boxed{\Psi \vdash\ n : b}$

$$\frac{}{\Psi \vdash\ n : \mathit{int}}\ (\textit{int-t}) \qquad \frac{}{\Psi \vdash\ n : \Psi(n)}\ (\textit{addr-heap-t})$$

$$\frac{}{\Psi \vdash\ n : \mathit{sptr}}\ (\textit{addr-stack-t}) \qquad \frac{\Psi \vdash\ n : b\ \mathit{ref}^{\varphi} \qquad \varphi \preceq \varphi'}{\Psi \vdash\ n : b\ \mathit{ref}^{\varphi'}}\ (\textit{addr-subtp-t})$$

$\boxed{\Psi; \Delta \vdash^{Z}\ n : t}$

$$\frac{\Psi \vdash\ n : b \qquad \Delta \vdash\ E = n}{\Psi; \Delta \vdash^{Z}\ n : \langle c, b, E\rangle}\ (\textit{val-t})$$

$$\frac{\Psi; \Delta \vdash^{Z}\ n : t' \qquad \Delta \vdash t' \preceq t}{\Psi; \Delta \vdash^{Z}\ n : t}\ (\textit{val-subtp-t})$$

$$\frac{n \neq 0 \qquad \Psi; \Delta \vdash^{Z}\ n : \langle c, b, E\rangle \qquad \Delta \vdash\ E' = 0}{\Psi; \Delta \vdash^{Z}\ n : E' = 0 \Rightarrow \langle c, b, E\rangle}\ (\textit{cond-t})$$

$$\frac{\Delta \vdash\ E' \neq 0}{\Psi; \Delta \vdash^{Z}\ 0 : E' = 0 \Rightarrow \langle c, b, E\rangle}\ (\textit{cond-n0-t})$$

$$\frac{\Delta \vdash\ E : \kappa_{int}}{\Psi; \Delta \vdash^{c}\ n : \langle c, b, E\rangle}\ (\textit{val-zap-t})$$

$$\frac{\Delta \vdash\ E' : \kappa_{int} \qquad \Delta \vdash\ E : \kappa_{int}}{\Psi; \Delta \vdash^{c}\ n : E' = 0 \Rightarrow \langle c, b, E\rangle}\ (\textit{val-zap-cond-t})$$

$$\frac{}{\Psi; \Delta \vdash^{Z}\ n : ns}\ (ns\textit{-t})$$

Figure 10. Value Typing

$\boxed{\varphi\uparrow}$

$$\begin{array}{rcl} 0\uparrow & = & \frac{1}{2} \\ \frac{1}{2}\uparrow & = & \frac{1}{2} \\ 1\uparrow & = & 1 \end{array}$$

$\boxed{\varphi \preceq \varphi'}$

$$1 \preceq \frac{1}{2} \preceq 0 \qquad \varphi \preceq \varphi$$

$\boxed{b \preceq b'}$

$$\frac{}{b \preceq b}\ (\textit{subtp-b-reflex}) \qquad \frac{\varphi \preceq \varphi'}{b\ \mathit{ref}^{\varphi} \preceq b\ \mathit{ref}^{\varphi'}}\ (\textit{subtp-b-ref}) \qquad \frac{}{b \preceq \mathit{int}}\ (\textit{subtp-b-int})$$

$\boxed{\Delta \vdash\ t \preceq t'}$

$$\frac{\Delta \vdash\ E_1 = E_2 \qquad b_1 \preceq b_2}{\Delta \vdash\ \langle c, b_1, E_1\rangle \preceq \langle c, b_2, E_2\rangle}\ (\textit{subtp-t-triple})$$

$$\frac{\Delta \vdash\ t \preceq t' \qquad \Delta \vdash\ E = E'}{\Delta \vdash\ (E = 0 \Rightarrow t) \preceq (E' = 0 \Rightarrow t')}\ (\textit{subtp-t-cond})$$

$$\frac{}{\Delta \vdash\ t \preceq ns}\ (\textit{subtp-t-ns})$$

$\boxed{\Delta \vdash\ \Gamma_1 \preceq \Gamma_2}$

$$\frac{\forall r \in \mathit{Dom}(\Gamma_2).\ \Gamma_1(r) \preceq \Gamma_2(r)}{\Delta \vdash\ \Gamma_1 \preceq \Gamma_2}\ (\textit{reg-file-comp})$$

$\boxed{\Delta \vdash\ \varsigma \preceq \varsigma'}$

$$\frac{\Delta \vdash E = E'}{\Delta \vdash\ E : \mathit{sbase} \preceq E' : \mathit{sbase}}\ (\textit{subtp-}\varsigma\textit{-base})$$

$$\frac{\Delta \vdash E = E'}{\Delta \vdash\ E : \rho \preceq E' : \rho}\ (\textit{subtp-}\varsigma\textit{-var})$$

$$\frac{\Delta \vdash E = E' \qquad \Delta \vdash t \preceq t' \qquad \Delta \vdash \varsigma \preceq \varsigma'}{\Delta \vdash\ E : (t :: \varsigma) \preceq E' : (t' :: \varsigma')}\ (\textit{subtp-}\varsigma\textit{-cons})$$

Figure 11. Subtyping

$\boxed{\Delta \vdash \varsigma\ wf}$

$$\frac{\Delta \vdash E : \kappa_{int}}{\Delta \vdash E : sbase\ wf}\ (\varsigma\text{-}wf\text{-}base) \qquad \frac{\Delta(\rho) = \kappa_\sigma \qquad \Delta \vdash E : \kappa_{int}}{\Delta \vdash E : \rho\ wf}\ (\varsigma\text{-}wf\text{-}var)$$

$$\frac{\Delta \vdash E + 1 = E' \qquad \Delta \vdash (E' : \sigma')\ wf}{\Delta \vdash E : (t :: (E' : \sigma')\ wf}\ (\varsigma\text{-}wf\text{-}cons)$$

$\boxed{\Delta; \varsigma \vdash E : t}$

$$\frac{\Delta \vdash E_s = E}{\Delta;\ E_s : (t :: \varsigma')\ \vdash E : t}\ (\varsigma\text{-}lookup\text{-}top) \qquad \frac{\Delta \vdash E_s \neq E \qquad \Delta; \varsigma' \vdash E : t}{\Delta;\ E_s : (t :: \varsigma')\ \vdash E : t}\ (\varsigma\text{-}lookup\text{-}tail)$$

$\boxed{\Delta \vdash \varsigma[E \mapsto t] = \varsigma'}$

$$\frac{\Delta \vdash E_s = E}{\Delta \vdash (E_s : (t_s :: \varsigma))[E \mapsto t] = E_s : (t :: \varsigma)}\ (\varsigma\text{-}update\text{-}top) \qquad \frac{\Delta \vdash E_s \neq E \qquad \Delta \vdash \varsigma[E \mapsto t] = \varsigma'}{\Delta \vdash (E_s : (t_s :: \varsigma))[E \mapsto t] = E_s : (t_s :: \varsigma')}\ (\varsigma\text{-}update\text{-}tail)$$

Figure 12. Stack Typing Judgments

$\boxed{\Psi;\Theta \vdash\ ir\ \Rightarrow RT}$

$$\frac{}{\Psi;(\Delta;\Gamma;\overline{(E_d,E_s)};E_m;\varsigma) \vdash\ \cdot\ \Rightarrow (\Delta;\Gamma;\overline{(E_d,E_s)};E_m;\varsigma)}\ (\cdot\text{-}t)$$

$$\frac{\Gamma(r_s) = \langle c, int, E'_s\rangle \qquad \Gamma(r_t) = \langle c, int, E'_t\rangle}{\Psi;(\Delta;\Gamma;\overline{(E_d,E_s)};E_m;\varsigma) \vdash\ op\ \ r_d, r_s, r_t\ \Rightarrow (\Delta;\Gamma{+}{+}[r_d \mapsto \langle c, int, E'_s\ op\ E'_t\rangle];\overline{(E_d,E_s)};E_m;\varsigma)}\ (op2r\text{-}t)$$

$$\frac{\Gamma(r_s) = \langle c, int, E'_s\rangle}{\Psi;(\Delta;\Gamma;\overline{(E_d,E_s)};E_m;\varsigma) \vdash\ op\ \ r_d, r_s, n\ \Rightarrow (\Delta;\Gamma{+}{+}[r_d \mapsto \langle c, int, E'_s\ op\ n\rangle];\overline{(E_d,E_s)};E_m;\varsigma)}\ (op1r\text{-}t)$$

$$\frac{\Psi;\Delta \vdash\ n : t}{\Psi;(\Delta;\Gamma;\overline{(E_d,E_s)};E_m;\varsigma) \vdash\ mov\ \ r_d, n\ \Rightarrow (\Delta;\Gamma{+}{+}[r_d \mapsto t];\overline{(E_d,E_s)};E_m;\varsigma)}\ (mov\text{-}n\text{-}t)$$

$$\frac{\Gamma(r_s) = t}{\Psi;(\Delta;\Gamma;\overline{(E_d,E_s)};E_m;\varsigma) \vdash\ mov\ \ r_d, r_s\ \Rightarrow (\Delta;\Gamma{+}{+}[r_d \mapsto t];\overline{(E_d,E_s)};E_m;\varsigma)}\ (mov\text{-}reg\text{-}t)$$

$$\frac{\begin{array}{c} x \notin \Delta \\ \Gamma' = \Gamma{+}{+}[r_g \mapsto \langle G, b\ ref^0, x\rangle][r_b \mapsto \langle B, b\ ref^0, x\rangle] \end{array}}{\Psi;(\Delta;\Gamma;\overline{(E_d,E_s)};E_m;\varsigma) \vdash\ malloc[b]\ r_g\ r_b\ \Rightarrow (\Delta, x : \kappa_{int};\Gamma';\overline{(E_d,E_s)};upd\ E_m\ x\ 0;\varsigma)}\ (malloc\text{-}t)$$

$$\frac{\begin{array}{c} \Gamma(sp_G) = \langle G, sptr, E_g\rangle \qquad \Gamma(sp_B) = \langle B, sptr, E_b\rangle \\ \Delta \vdash E_g = E_b \qquad \Delta \vdash E_g = E_t \\ \Gamma' = \Gamma{+}{+}[sp_G \mapsto \langle G, sptr, E_g{-}n\rangle][sp_B \mapsto \langle B, sptr, E_b{-}n\rangle] \\ \varsigma' = E_t{-}n : ns :: E_t{-}n{+}1 : ns :: \ldots :: E_t : \sigma \end{array}}{\Psi;(\Delta;\Gamma;\overline{(E_d,E_s)};E_m;(E_t : \sigma)) \vdash\ salloc\ n\ \Rightarrow (\Delta;\Gamma';\overline{(E_d,E_s)};E_m;\varsigma')}\ (salloc\text{-}t)$$

$$\frac{\begin{array}{c} \Gamma(sp_G) = \langle G, sptr, E_g\rangle \qquad \Gamma(sp_B) = \langle B, sptr, E_b\rangle \\ \Delta \vdash E_g = E_b \qquad \Delta \vdash E_g = E_t \\ \varsigma = E_t : t :: \ldots :: E_f : \sigma \qquad \Delta \vdash E_f = E_g + n \\ \Gamma' = \Gamma{+}{+}[sp_G \mapsto \langle G, sptr, E_g{+}n\rangle][sp_B \mapsto \langle B, sptr, E_b{+}n\rangle] \end{array}}{\Psi;(\Delta;\Gamma;\overline{(E_d,E_s)};E_m;\varsigma) \vdash\ sfree\ n\ \Rightarrow (\Delta;\Gamma';\overline{(E_d,E_s)};E_m;E_f : \sigma)}\ (sfree\text{-}t)$$

$$\frac{\Delta \vdash \Gamma(r_s) \preceq \langle G, b\ ref^{\frac{1}{2}}, E'_s\rangle \qquad E\ =\ sel\ (\overline{upd}\ E_m\ \overline{(E_d,E_s)})\ E'_s}{\Psi;(\Delta;\Gamma;\overline{(E_d,E_s)};E_m;\varsigma) \vdash\ ld_G\ r_d\ r_s\ \Rightarrow (\Delta;\Gamma{+}{+}[r_d \mapsto \langle G, b, E\rangle];\overline{(E_d,E_s)};E_m;\varsigma)}\ (ld_G\text{-}t)$$

$$\frac{\Gamma(r_s) = \langle B, b\ ref^1, E'_s\rangle \qquad E\ =\ sel\ E_m\ E'_s}{\Psi;(\Delta;\Gamma;\overline{(E_d,E_s)};E_m;\varsigma) \vdash\ ld_B\ r_d\ r_s\ \Rightarrow (\Delta;\Gamma{+}{+}[r_d \mapsto \langle B, b, E\rangle];\overline{(E_d,E_s)};E_m;\varsigma)}\ (ld_B\text{-}t)$$

$$\frac{\Gamma(sp_c) = \langle c, sptr, E_c\rangle \qquad \Delta \vdash E_c + n = E_n \qquad \Delta;\varsigma \vdash E_n : \langle c, b, E\rangle}{\Psi;(\Delta;\Gamma;\overline{(E_d,E_s)};E_m;\varsigma) \vdash\ sld_c\ r_d\ n\ \Rightarrow (\Delta;\Gamma{+}{+}[r_d \mapsto \langle c, b, E\rangle];\overline{(E_d,E_s)};E_m;\varsigma)}\ (sld_c\text{-}t)$$

$$\frac{\begin{array}{c} \Gamma(r_d) = \langle G, b\ ref^{\varphi}, E'_d\rangle \qquad \Gamma(r_s) = \langle G, b, E'_s\rangle \\ \Gamma' = \Gamma{+}{+}\ \textit{except}\ \forall\, r\ \textit{where}\ \Gamma(r) = \langle c_r, b\ ref^{\varphi}, E_r\rangle\ \textit{and}\ \Delta \vdash E_r = E'_d\ .\ \Gamma'(r) = \langle c_r, b\ ref^{\varphi\uparrow}, E_r\rangle \end{array}}{\Psi;(\Delta;\Gamma;\overline{(E_d,E_s)};E_m;\varsigma) \vdash\ st_G\ r_d\ r_s\ \Rightarrow (\Delta;\Gamma';(E'_d,E'_s),\overline{(E_d,E_s)};E_m;\varsigma)}\ (st_G\text{-}t)$$

$$\frac{\begin{array}{c} \Delta \vdash \Gamma(r_d) \preceq \langle B, b\ ref^{\frac{1}{2}}, E''_d\rangle \qquad \Gamma(r_s) = \langle B, b, E''_s\rangle \\ \Delta \vdash\ E'_s = E''_s \qquad \Delta \vdash\ E'_d = E''_d \\ \Gamma' = \Gamma{+}{+}\ \textit{except}\ \forall\, r\ \textit{where}\ \Gamma(r) = \langle c_r, b\ ref^{\frac{1}{2}}, E_r\rangle\ \textit{and}\ \Delta \vdash E_r = E'_d\ .\ \Gamma'(r) = \langle c_r, b\ ref^1, E_r\rangle \end{array}}{\Psi;(\Delta;\Gamma;\overline{(E_d,E_s)},(E'_d,E'_s);E_m;\varsigma) \vdash\ st_B\ r_d\ r_s\ \Rightarrow (\Delta;\Gamma';\overline{(E_d,E_s)};upd\ E_m\ E'_d\ E'_s;\varsigma)}\ (st_B\text{-}t)$$

$$\frac{\begin{array}{c} \Gamma(sp_G) = \langle G, sptr, E_g\rangle \qquad \Gamma(sp_B) = \langle B, sptr, E_b\rangle \qquad \Delta \vdash E_g = E_b \\ \Delta \vdash E_g + n = E_n \qquad \Gamma(r_v) = \langle c, b, E_v\rangle \qquad \Delta \vdash \varsigma[E_n \mapsto \langle c, b, E_v\rangle] = \varsigma' \end{array}}{\Psi;(\Delta;\Gamma;\overline{(E_d,E_s)};E_m;\varsigma) \vdash\ sst\ n\ r_v\ \Rightarrow (\Delta;\Gamma{+}{+};\overline{(E_d,E_s)};E_m;\varsigma')}\ (sst\text{-}t)$$

$$\frac{\begin{array}{c}\Gamma(d) = \langle G, int, 0\rangle \qquad \Gamma(r_z) = \langle G, int, E_z\rangle \\ \Gamma(r_d) = \langle G, \Theta \to void, E'_d\rangle \qquad \Theta = (\Delta'; \Gamma'; \overline{(E'_d, E'_s)}; E'_m) \qquad \Gamma'(d) = \langle G, int, 0\rangle\end{array}}{\Psi; (\Delta; \Gamma; \overline{(E_d, E_s)}; E_m; \varsigma) \vdash \ bz_G\ r_z\ r_d \ \Rightarrow (\Delta; \Gamma{+}{+}[d \mapsto \ E_z = 0 \Rightarrow \langle G, \Theta \to void, E'_d\rangle]; \overline{(E_d, E_s)}; E_m; \varsigma)}\ (bz_G\text{-}t)$$

$$\frac{\begin{array}{c}\Gamma(r_d) = \langle G, \Theta \to void, E_{rd'}\rangle \qquad \Theta = (\Delta'; \Gamma'; \overline{(E'_d, E'_s)}; E'_m) \\ \Gamma(d) = \langle G, int, 0\rangle \qquad \Gamma'(d) = \langle G, int, 0\rangle\end{array}}{\Psi; (\Delta; \Gamma; \overline{(E_d, E_s)}; E_m; \varsigma) \vdash \ jmp_G\ r_d \ \Rightarrow (\Delta; \Gamma{+}{+}[d \mapsto \ \langle G, \Theta \to void, E_{rd'}\rangle]; \overline{(E_d, E_s)}; E_m; \varsigma)}\ (jmp_G\text{-}t)$$

$$\frac{\begin{array}{c}\Gamma(r_z) = \langle B, int, E_z\rangle \\ \Gamma(r_d) = \langle B, (\Delta'; \Gamma'; \overline{(E'_d, E'_s)}; E'_m; \varsigma') \to void, E_r\rangle \\ \Gamma(d) = E'_z = 0 \Rightarrow \langle G, (\Delta'; \Gamma'; \overline{(E'_d, E'_s)}; E'_m; \varsigma') \to void, E'_r\rangle \\ \Delta \vdash \ E_z = E'_z \\ \Delta \vdash \ E_r = E'_r \\ \exists S. \Delta \vdash \ S : \Delta' \\ S(\Gamma')(d) = \langle G, int, 0\rangle \\ S(\Gamma')(pc_G) = \langle G, int, E'_r\rangle \\ S(\Gamma')(pc_B) = \langle B, int, E_r\rangle \\ \Delta \vdash \Gamma \preceq \ S(\Gamma') \\ \Delta \vdash \ \overline{(E_d, E_s)} = S(\overline{(E'_d, E'_s)}) \\ \Delta \vdash \ E_m = S(E'_m) \\ \Delta \vdash \ \varsigma \preceq S(\varsigma')\end{array}}{\begin{array}{c}\Psi; (\Delta; \Gamma; \overline{(E_d, E_s)}; E_m; \varsigma) \vdash \ bz_B\ r_z\ r_d \ \Rightarrow \\ (\Delta; \Gamma{+}{+}; \overline{(E_d, E_s)}; E_m; \varsigma)\end{array}}\ (bz_B\text{-}t)$$

$$\frac{\begin{array}{c}\Gamma(d) = \langle G, (\Delta'; \Gamma'; \overline{(E'_d, E'_s)}; E'_m; \varsigma') \to void, E'_r\rangle \\ \Gamma(r_d) = \langle B, (\Delta'; \Gamma'; \overline{(E'_d, E'_s)}; E'_m; \varsigma') \to void, E_r\rangle \\ \Delta \vdash \ E_r = E'_r \\ \exists S. \Delta \vdash \ S : \Delta' \\ S(\Gamma')(d) = \langle G, int, 0\rangle \\ S(\Gamma')(pc_G) = \langle G, int, E'_r\rangle \\ S(\Gamma')(pc_B) = \langle B, int, E_r\rangle \\ \Delta \vdash \Gamma \preceq \ S(\Gamma') \\ \Delta \vdash \ \overline{(E_d, E_s)} = S(\overline{(E'_d, E'_s)}) \\ \Delta \vdash \ E_m = S(E'_m) \\ \Delta \vdash \ \varsigma \preceq S(\varsigma')\end{array}}{\Psi; (\Delta; \Gamma; \overline{(E_d, E_s)}; E_m; \varsigma) \vdash \ jmp_B\ r_d \ \Rightarrow void}\ (jmp_B\text{-}t)$$

Figure 14. Instruction Typing Rules – Control Flow

$\boxed{\Psi \vdash^{Z} \; R : \Gamma}$

$$\frac{\begin{array}{c}\forall a in Dom(\Gamma).\ \Psi; \cdot \vdash^{Z} \; R(a) : \Gamma(a) \\ \cdot \vdash \Gamma(pc_G) \preceq \langle G, int, E_G \rangle \\ \cdot \vdash \Gamma(pc_B) \preceq \langle B, int, E_B \rangle \\ \cdot \vdash \; E_G = E_B\end{array}}{\Psi \vdash^{Z} \; R : \Gamma} \; (\textit{reg-file-t})$$

$\boxed{\Psi \vdash \; C}$

$$\frac{\begin{array}{l}0 \notin Dom(C) \\ \forall n \in Dom(C). \quad \Psi(n) = \Theta \rightarrow void \; \wedge \; \Psi; \Theta \vdash \; C(n) \Rightarrow RT \; \wedge \\ \qquad\qquad\qquad\quad (RT = \Theta' \; implies \; \Psi(n+1) = \Theta' \rightarrow void)\end{array}}{\Psi \vdash \; C} \; (\textit{C-t})$$

$\boxed{M = M_s \overset{\mathcal{L}}{\#} M_m}$

$$\frac{\begin{array}{c}Dom(M) = Dom(M_1) \cup Dom(M_2) \\ Dom(M_1) \cap Dom(M_2) = \emptyset \\ \forall \ell_1 \in Dom(M_1). \forall \ell_{\mathcal{L}} \in \mathcal{L}. \forall \ell_2 \in Dom(M_2).\ \ell_1 < \ell_{\mathcal{L}} < \ell_2\end{array}}{M = M_1 \overset{\mathcal{L}}{\#} M_2} \; (\textit{\#-def})$$

$\boxed{\Psi; M; Q \vdash^{Z} \; \ell : b \; ref^{\varphi}}$

$$\frac{\Psi(\ell) = b \; ref^{1} \qquad \Psi \vdash M(\ell) : b}{\Psi; M; Q \vdash^{Z} \; \ell : b \; ref^{1}} \; (\textit{init-t})$$

$$\frac{\Psi(\ell) = b \; ref^{0}}{\Psi; M; Q \vdash^{Z} \; \ell : b \; ref^{0}} \; (\textit{uninit-t})$$

$$\frac{\begin{array}{c}\Psi(\ell) = b \; ref^{\frac{1}{2}} \\ Z \neq G \implies \exists n.\ (\ell, n) \in Q\end{array}}{\Psi; M; Q \vdash^{Z} \; \ell : b \; ref^{\frac{1}{2}}} \; (\textit{halfinit-t})$$

$\boxed{\Psi \vdash^{Z} \; Q : \overline{(E_d, E_s)}}$

$$\frac{}{\Psi \vdash^{Z} \; () : ()} \; (\textit{Q-emp-t})$$

$$\frac{\begin{array}{c}Z \neq G \\ \Psi \vdash^{Z} \; \overline{(n_1', n_2')} : \overline{(E_d', E_s')} \\ \cdot \vdash E_d = n_1 \qquad \cdot \vdash E_s = n_2 \\ \Psi \vdash n1 : b \; ref^{\varphi} \qquad \varphi \preceq \frac{1}{2} \qquad \Psi \vdash n2 : b\end{array}}{\Psi \vdash^{Z} \; (n_1, n_2), \overline{(n_1', n_2')} : (E_d, E_s), \overline{(E_d', E_s')}} \; (\textit{Q-t}) \qquad \frac{\begin{array}{c}\Psi \vdash^{G} \; \overline{(n_1', n_2')} : \overline{(E_d', E_s')} \\ \cdot \vdash \; E_d : \kappa_{int} \qquad \cdot \vdash \; E_s : \kappa_{int}\end{array}}{\Psi \vdash^{G} \; (n_1, n_2), \overline{(n_1', n_2')} : (E_d, E_s), \overline{(E_d', E_s')}} \; (\textit{Q-zap-t})$$

$\boxed{\Psi \vdash^{Z} \; (M, Q) : (E_m, \overline{(E_d, E_s)})}$

$$\frac{\begin{array}{l}\forall \ell \in Dom(M).\ \exists \varphi.\ \Psi; M; Q \vdash^{Z} \; \ell : b \; ref^{\varphi} \\ [\![E_m]\!] = M \qquad \Psi \vdash^{Z} Q : \overline{(E_d, E_s)}\end{array}}{\Psi \vdash^{Z} \; (M, Q) : (E_m, \overline{(E_d', E_s')})} \; (\textit{heap-t})$$

$\boxed{\Psi \vdash^{Z} \; M : \varsigma}$

$$\frac{\cdot \vdash E = \ell \qquad Dom(M) = \{\ell\}}{\Psi \vdash^{Z} \; M : (E : sbase)} \; (\textit{ς-t-base})$$

$$\frac{\begin{array}{l}\cdot \vdash (E : t :: \varsigma') \; wf \qquad \cdot \vdash E = \ell \\ M = \{\ell \rightarrow n\} \overset{\cdot}{\#} M' \\ \Psi; \cdot \vdash^{Z} \; n : t \qquad \Psi \vdash^{Z} \; M' : \varsigma'\end{array}}{\Psi \vdash^{Z} \; M : (E : t :: \varsigma')} \; (\textit{ς-t-cons})$$

Figure 15. Machine State Element Typing

$\boxed{\vdash^{Z} (R, C, M, Q, ir)}$

$\boxed{\vdash^{Z} (R, C, M, Q, ir) : \Psi, \Gamma, \varsigma}$

$$\frac{\begin{array}{l} Dom(\Psi) = Dom(C) \cup Dom(M_m) \\ M = M_s \overset{Dom(C)}{\#} M_m \\ \Psi \vdash C \\ \forall c \neq Z.\ ir \neq \cdot \implies C(R(pc_c)) = ir \\ \forall c \neq Z.\ \Psi(R(pc_c)) = (\Delta; \Gamma; \overline{(E_d, E_s)}; E_m; \varsigma) \rightarrow void \\ \exists S.\ \cdot \vdash S : \Delta \\ \Psi \vdash^{Z} M_s : S(\varsigma) \\ \Psi \vdash^{Z} (M_m, Q) : (S(E_m), S(\overline{(E_d, E_s)})) \\ \Psi \vdash^{Z} R : S(\Gamma) \\ K = extractK(R, \Gamma),\ extractK(M_h),\ extractK(M_s, \varsigma) \end{array}}{\vdash^{Z} (R, C, M, Q, ir) \text{: } K} \ (\Sigma\text{-}t)$$

Figure 16. Machine State Typing

$$
\begin{array}{llll}
\textit{extended color} & k & ::= & c \mid none \\
\textit{coloring} & K & ::= & \cdot \mid a \mapsto k \mid \ell \mapsto k
\end{array}
$$

$$
\begin{array}{lll}
extractK_t(\langle c, b, E\rangle) & = & c \\
extractK_t(E_z = 0 \Rightarrow \langle c, b, E\rangle) & = & c \\
extractK_t(ns) & = & none \\
\\
extractK(R, \Gamma) & = & \forall a.\ a \in Dom(\Gamma)\ ?\ a \mapsto extractK_t(\Gamma(a))\ :\ a \mapsto none \\
\\
extractK(M_s, \varsigma) & = & \forall \ell \in Dom(M_s).\ .; \varsigma \vdash \ell : t\ \Rightarrow\ \ell \mapsto extractK_t(t) \\
\\
extractK(M_h) & = & \forall \ell \in Dom(M_h).\ \ell \mapsto none
\end{array}
$$

$\boxed{k\ n_1\ sim^Z\ k\ n_2}$

$$
\frac{}{k\ n\ sim^Z\ k\ n}\ (\textit{sim-val}) \qquad \frac{}{c\ n\ sim^c\ c\ n'}\ (\textit{sim-val-zap})
$$

$\boxed{K \vdash R\ sim^Z\ R'}$

$$
\frac{\forall a.\ \ K(a)\ R(a)\ sim^Z\ K(a)\ R'(a)}{K \vdash R\ sim^Z\ R'}\ (\textit{sim-R})
$$

$\boxed{K \vdash M\ sim^Z M'}$

$$
\frac{\begin{array}{c} Dom(M) = Dom(M') \\ \forall \ell \in Dom(M).\ \ K(\ell)\ M(\ell)\ \ sim^Z\ \ K(\ell)\ M'(\ell) \end{array}}{K \vdash M\ sim^Z\ M'}\ (\textit{sim-M})
$$

$\boxed{Q\ sim^Z\ Q'}$

$$
\frac{}{\cdot\ sim^Z\ \cdot}\ (\textit{sim-Q-empty})
$$

$$
\frac{G\ n_1\ sim^Z\ G\ n_1' \qquad G\ n_2\ sim^Z\ G\ n_2' \qquad Q\ sim^Z\ Q'}{((n_1, n_2), Q)\ \ sim^Z\ \ ((n_1', n_2'), Q')}\ (\textit{sim-Q})
$$

$\boxed{\Sigma_1\ sim^Z\ \Sigma_2}$

$$
\frac{\begin{array}{c} \vdash (R, C, M, Q, ir) : K \\ \vdash^Z (R', C, M', Q', ir) : K \\ K \vdash R\ sim^Z\ R' \\ K \vdash M\ sim^Z\ M' \\ Q\ sim^Z\ Q' \end{array}}{(R, C, M, Q, ir)\ sim^Z\ (R', C, M', Q', ir)}\ (\textit{sim-}\Sigma)
$$

Figure 17. Similarity of Machine States

Lemmas for Progress, Preservation, and Simulation

```
==========================
NEW LEMMAS FOR TRANSLATION
==========================


Valid Stack Location Lemma
--------------------------

1. If  P |-z M_s : s  and  .;s |- E' : t                then  . |- E' = l  and l in Dom(M)
2. If  P |-z M_s : s  and  . |- s[E' -> t] = s'         then  . |- E' = l  and l in Dom(M)
3. If  P |-z M_s : s  and  s = E1:t1 :: ... :: En : us  then  . |- Ei = li and li in Dom(M)

Proof: By repeated inversion of (s-t-cons) and (s-t-base), all expressions E on spine of s are equal to
an l in Dom(M). By definition of .;s |- E' : <c,b,E> and . |- s[E' ->t] = s, E' is equal to an expression on the spine of s.
Thus E' is equal to some l that is in Dom(M).





Stack Lookup Lemma
------------------

If  P |-Z M_s : s  and  .;s |- E : t    and  . |- E = l  then  P;. |-Z M_s(l) : t

Proof: by induction on the structure of .;s |- E : t


case (s-lookup-top):

1. s = Es : t : s'                                     | inspection of (s-lookup-top)
2. . |- Es = E                                         | inversion of (s-lookup-top)

3. P |-Z M_s : Es : t : s'                             | assumption, 1
4. . |- Es = l'  and  M(l') = n  and  P;. |-Z n : t     | inversion of (s-t-cons), 3

5. l' = l                                              | Exp Eq Trans Lemma, assumption, 2, 4
6. P;. |-Z M(l) : t                                    | 4, 5


case (s-lookup-tail):

1. s = Es : t : s'                                     | inspection of (s-lookup-tail)
2. . |- Es =/= E                                       | inversion of (s-lookup-tail)
3. . ; s' |- E : t                                     | inversion of (s-lookup-tail)

4. P |-Z M_s : Es : t : s'                             | assumption, 1
5. P |-Z M_s' : s'                                     | inversion of (s-t-cons)
6. P;. |-Z M_s'(l) : t                                 | I.H. 5, 3, assumption

7. M_s = M_s' #{} {l'->n}                              | inversion of (s-t-cons)
8. l =/= l'                                            | Exp Eq Trans, assumption, 2
9. P;. |-Z M_s(l) : t                                  | 6, 7, def of #, 8

* stack lookup lemma compete





Stack Update Lemma
------------------

If  P |-Z M : s   and  . |- s[E -> t] = s'   and  . |- E = l  and  P;. |-Z n : t
then  P |-Z M[l -> n] : s'

Proof: by induction on the structure of . |- s[E -> t] = s'

case (s-update-top):
1. s  = Es : ts : s''  and  s' = Es : t : s''  and  . |- Es = E     | inspection/inversion of (s-update-top)
```

```
2. . |- Es = l'                                                     | Inversion of (s-t-cons), assumption, 1
3. . |- Es : ts : s'' wf
4. M = {l' -> n'} # M'
5. P; . |-Z n' : ts
6. P |-Z M' : s''

7. . |- Es = l                                                      | Exp Eq Trans, assumption, 1
8. . |- Es : t : s'' wf                                             | Inversion/reconstruction of (s-wf-cons), 3
9. M[l -> n] = {l -> n} # M'                                        | Inversion/reconstruction of #, 4, (Exp Eq Trans, assumption, 1, 2)
10. P;. |-Z n : t                                                   | assumption
11. P |-Z M[l -> n] : Es : t : s''                                  | (s-t-cons), 8, 7, 9, 10, assumption

case (s-update-tail):
1. = Es : ts : s''  and  s' = Es : ts : s'''                        | inspection of (s-update-tail)
2. . |- Es =/= E  and  . |- s''[E->t] = s'''                        | inversion of (s-update-tail), 1

3. . |- Es = l'                                                     | Inversion of (s-t-cons), assumption, 1
4. . |- Es : ts : s'' wf
5. M = {l' -> n'} # M'
6. P; . |-Z n' : ts
7. P |-Z M' : s''

9. P |-Z M'[l -> n] : s'''                                          | I.H. 7, 2, assumptions

10. l =/= l'                                                        | Exp Eq Trans, assumption 2, 3

11. . |- Es : ts : s''' wf                                          | (s-wf-cons), Inversion of (s-t-cons), 9
12. M = {l' -> n'} # M'[l->n]                                       | 5, 10

13. P |- M[l->n] : Es: ts : s'''                                    | (s-t-cons), assumption, 11, 12, 6, 9

* Stack update lemma complete





Heap Extension Lemma
--------------------

1. If  P;. |-Z v : t         and n not in Dom(P)   then  P,n->t |-Z v : t
2. If  P |-Z M : s           and n not in Dom(P)   then  P,n->t |-Z M : s
3. If  P;M;Q |-Z l : b reff  and n not in Dom(P)   then  P,n->t;M;Q |-Z l : b reff
4. If  P;T |- ir => RT       and n not in Dom(P)   then  P,n->t;T |- ir => RT

Proof: By induction on the appropriate derivations.



Psi Subtyping Lemma
--------------------

1. If P; . |-Z v : t        and  b' <= P(n)               then P[n -> b']; . |-Z v : t
2. If P;M;Q |-Z l : b reff  and  b' <= P(n)  and l =/= n  then P[n -> b'];M;Q |-Z l : b reff
3. If P;T |- ir => RT       and  b' <= P(n)               then P[n -> b'];T |- ir => RT

Proof:
1. if derivation of  P; . |-Z v : t  depends on P(n),  then insert (val-subtp-t) to rebuild derivation

2. by case analysis of P;M;Q |-Z l : b reff
   based on subtyping relationship, b' <= P(n) must be  b reff <= brefp
   if derivation depends on P(n), insert (addr-subtp-t)

3. by case analysis of P;T |- ir => RT.
   only use of P is in (mov-t). insert (addr-subtp-t) if necessary.




Substitution Extension Lemma
-----------------------------

1. If P;. |-Z v : S(t)  and  . |- S : D  and  x not in D  then  P;. |-Z v : (S,E/x)(t)
2. If P |-Z M : S(s)    and  . |- S : D  and  x not in D  then  P |-Z M : (S,E/x)(s)
3. If P |-Z Q : S(seq(E_d,E_m))  and . |- S : D  and x not in D  then   P |-Z Q : (S,E,x)(seq(E_d,E_m))
4. If P |-Z R : S(G)    and  . |- S : D  and  x not in D  then  P |-Z R : (S,E/x)(G)
5. If M = [[S(E_m)]]    and  . |- S : D  and  x not in D  then  M = [[S2(E_m)]]
Proof: by induction!
```

```
=================================
MODIFIED LEMMAS FOR TRANSLATION
=================================



Canonical Forms Lemma
---------------------

If Dom(P) = Dom(C) union Dom(M), and
 P |-Z (M,Q) : (E_m,seq(E_d,E_s)) and                                  [here M only describes heap]
 P |- C, and
 P; . |-z n : t
then
1. If t = <c,b,E> or t = (E' = 0) => <c,b,E>, and c = z
   then . |- E : kint (and . |- E' : kint), but no particular properties of n are known

2. If t = <c,int,E> and c not= z then . |- E = n.

3. If t = <c,T-->void,E> and c not= z then
    P(n) = T --> void and n in Dom(C) and . |- E = n and n =/= 0.

4. If t = <c,b reff,E> and c not= z then                               ** modified case (init flag)
    a. P(n) = b refp  and  p <= f
    b. n in Dom(M) and
    c. . |- E = n
    d. p=h ==> ( Z=/=G ==> exists (n,v) in Q  and  all (n,v) in Q. P |- v : b )
    e. p=1 ==> P |- M(l) : b and ( Z=/=G ==> all (n,v) in Q. P |- v : b )

5. If t = (E' = 0) => t, and c not= z and . |- E' = 0 then n is not 0.
6. If t = (E' = 0) => t, and c not= z and . |- E' not= 0 then n is 0.

7. If t = ns then no particular properties of n are known.             ** new case

8. If t = <c,sptr,E> and c not= z then . |- E = n                      ** new case



Proof:  By induction on the derivation P; . |-z n : t

subtyping rule does not break the following because:
- initflag does not affect information known about ref types
- ns requires no special properties
- all types other than ns know at least as much as int

1. if c=z, either (val-zap-t) or (val-zap-cond-t) may apply, so inversion gives no guaranteed info about n.
   in all rules, inversion gives that expressions are wellformed (either directly or by another inversion of D |- E = E')

2. only rule (val-t) applies. inversion of (val-t) gives . |- E = n

3. only rule (val-t) applies. inversion gives . |- E = n and P |- n : T-->void.
   only way to derive P |- n : T --> void is if P(n) = T --> void
   Dom(P) = Dom(M) union Dom(C) and so n might be in M  or C
   - P |-Z (M,Q) : (E_m,seq(E_d,E_s)) ==> all l in Dom(Mm). P;M;Q |- l : b reff ==> P |- l : b reff ==> n is not in Dom(Mm). case doesn't apply
   - P |- C ==> n in Dom (C)  and  by inversion of (C-t), n =/= 0

4. only rule that applies is (val-t). inversion gives . |- E = n and P |- n : b refp.
   only way to derive P |- n : b refp is if P(n) = b reff
   Dom(P) = Dom(M) union Dom(C) and so n might be in M or C
   - P |- C ==> by inversion, all n' in Dom(C) have P(n') = T --> void. case doesn't apply
   - P |-Z (M,Q) : (E_m,seq(E_d,E_s)) ==> n in Dom(M)
   inversion of (heap-t), gives possibilites for initialization
   only one of these can apply depending on value of p

5/6. either rule (cond-n0-t) or (cond-t) might apply. by inversion, E' ?= 0 determines which one does apply.

7. only rule (ns-t) might apply. inversion gives us nothing.

8. only rule (val-t) applies. by inversion . |- E = n and P |- n : sptr, but this gives us nothing useful about n
   (stack pointers may be out of date, doesn't matter as long as we don't dereference them)




Substitution Lemma:
-------------------
1. If D,x:k |- E':k' and D |- E:k then D |- E'[E/x]:k'.
2. If D,x:k |- E1 = E2 and D |- E:k then D |- E1[E/x] = E2[E/x].
3. If D,x:k |- E1 not= E2 and D |- E:k then D |- E1[E/x] not= E2[E/x].
4. If P;D,x:k |-z v:t and D |- E:k then P;D |- v:t[E/x]
5. If P;D,x:k;G;seq(Ed,Es);Em;s |-z S(ir) => RT and D |- E:k then           ** modified -- added stack, ir can contain base type
        P;D;G[E/x];seq(Ed,Es)[E/x];Em[E/x];s[E/x] |-z ir => RT[E/x]
6. If D' |- S : D and P;D |-z v:t then P; D' |-z v : S(t).                  ** modified - subst may be incomplete
7. If D' |- S : D and P;D;G;seq(Ed,Es);Em;s |-z ir => (D;G';seq(Ed',Es');Em';s') then ** modified -- subst may be incomplete,
        P;.;S(G);S(seq(Ed,Es));S(Em);S(s) |-z S(ir) => (D';S(G');S(seq(Ed',Es'));S(Em');S(s)) ** add stack, ir can contain base type
8. If D,x:k |- E1 <= E2 and D |- E:k then D |- E1[E/x] <= E2[E/x].          ** new case

Proof:
```

```
By induction on the respective typing derivation for parts 1, 4, 5.
Parts 6, 7 by induction on the size of D, using parts 4 and 5 respectively.
Parts 2 and 3 and 8 are assumed true of the expression algebra.  Note that
part 3 is slightly unusual.  It may be trivially implemented simply
by requiring that E1 not= E2 holds only when E1 and E2 are closed.  This
judgement is only needed to type states during the proof of preservation
after a conditional branch has been executed, when, indeed, the expressions
E1 and E2 will be closed.




Subtyping Lemma:
----------------

1. If D |- t <= t'  and  P;D |-Z v:t    then  P;D |-Z v:t'
2. If D |- G <= G'  and  P |-Z R : G    then  P |-Z R : G'
3. If D |- s <= s'  and  P |-Z M : s    then  P |-Z M : s'
Proof:

1. By induction on the derivation of P;D |-z v:t using rule (val-subtp-t)

2. Inversion/Reconstruction of (reg-file-t) using Part 1.

3. Inversion/Reconstruction of (s-t-cons) and (s-t-base) Using part 1.



Color Weakening Lemma
---------------------

1. If P;.|- v:t                              then forall c. P;.|-c v:t
2. If  P |- R : G                            then forall c.  P |-c R : G
3. If  P |- (M_m,Q) : (E_m, seq(E_d,E_m))   then forall c.  P |-c (M_m,Q) : (E_m, seq(E_d,E_m))
4. If  P |- M : s                            then forall c.  P |-c M : s

Proof:
1. By induction on the value typing judgement.
2/3/4. By inversion/reconstruction of the appropriate rules using Part 1 as necessary.




==========================================================
UNMODIFIED (OR OBVIOUSLY MODIFIED) LEMMAS FOR TRANSLATION
==========================================================

Int Kinding Lemma
------------------

If  P;D |-Z v : <c,b,E>  then D |- E : kint.

Proof: By case analysis on the value typing judgment.




Queue Lemma:
------------
1. If P |-z Q : seq(Ed,Es) then length(Q) = length(seq(Ed,Es)).
2. If P |-z seq(n1,n2) : seq(Ed,Es) and z not= G then
    for k:1..length(seq(n1,n2)),
      .|- Edk = n1k and .|- Eds = n2k and for some b, P |- n1k : b ref and P |- n2k : b.

Proof:  Both parts by induction on the queue typing judgement.




Find Lemma
-----------

1. If   find(Q,n1) = ()       and  |-Z Q : seq(E_d,E_s)
   then for k:1...length(Q). . |- Edk =/= n1

2. If   find(Q,n1) = (n1,n2)  and  |-Z Q : seq(E_d,E_s)   and  Z =/= G
   then   n2 = sel (sequpd E_m (seq(E_d,E_s)) n1

Proof: By definition of find(), sequpd, sel




Irrelevant Update Lemma
------------------------
```

```
If E = sel (upd E_m E_s E_d) E_s' and . |- E_s =/= E_s' then E = sel E_m Es'

Proof: By definition of sel/upd



Exp Evaluation Lemma
--------------------

1. If . |- E : kint <==> Exists n. [[E]] = n
2. If . |- E : kmem <==> Exists M. [[E]] = M

Proof by induction on D |- E : k and [[]]




Exp Eq Transitivity
-------------------

If D |- E1 = E2 and D |- E2 = E3  then D |- E1 = E3

Proof: Inversion and reconstruction on (E-eq)






Substituting Closed Expressions
-------------------------------

If . |- E : k then Forall S. . |- S(E) : k

Proof: by induction on D |- E : k



Well-Typed Domain Lemma
------------------------

If |- (R1,C,M,Q1, ld_G rd, rs) then R1_val(r_d) in Dom(M)

Proof: By inversion of the ld_G-t type rule, inversion of the
register file typing rule and the Canonical Forms Lemma.
```

Progress Part 1

```
1. If |- (R,C,M,Q,ir) then (R,C,M,Q,ir) -->_0^s  (R',C',M',Q',ir')


Proof by case analysis on ir.                                        ** Change Summary:** COMPLETE (4/11/08)



Case .:                                                              ** Change Summary:** simple (add s to typing judgment)
-------

1. |- (R,C,M,Q,.)                                                    | Given
2. P |- R : S(G)                                                     | Inversion of (S-t), 1
3. . |- S(G)(pc_G) =< <G,int,E_G>                                    | Inversion of (reg-file-t), 2
   . |- S(G)(pc_B) =< <B,int,E_B>
4. P;.|- R(pc_G) : <G,int,E_G>                                       | 3, Inversion of (reg-file-t), 2, Subtyping Lemma
   P;.|- R(pc_B) : <B,int,E_B>
5. .|- E_G = R_val(pc_G)                                             | Canonical Forms, (Inversion of (heap-t), 1)
   .|- E_B = R_val(pc_B)
6. .|- E_G = E_B                                                     | Inversion of (reg-file-t), 2
7. R_val(pc_G) = R_val(pc_B)                                         | Exp Eq Trans 5, 6
8. all c. P(R_val(pc_c)) = (D;G;seq(E_d,E_s),E_m,s) --> void         | Inversion of (S-t), 1
9. R_val(pc_G) in Dom(C)                                             | 8
10. (R,C,M,Q,.) -->_0 (R,C,M,Q,C(R_val(pc_G)))                       | (fetch) 7, 9
*



Case op2r:                                                           ** Change Summary:** simple (add s to typing judgment, remove colors)
-------

1. |- (R,C,M,Q, op r_d,r_s,r_t)                                      | Given
2. P;(.;S(G);S(seq(E_d,E_s));S(E_m);S(s)) |- op  r_d,r_s,r_t => RT   | Inversion of (S-t, C-t), substitution, 1
3. S(G)(r_s) = <c,int,E_s'>                                          | Inversion of (op2r-t), 2
   S(G)(r_t) = <c,int,E_t'>
4. P |- R : S(G)                                                     | Inversion of (S-t), 1
5. P;. |- R(r_s) : < c,int,E_s'>                                     | Inversion of (reg-file-t), 4, 3
   P;. |- R(r_t) : < c,int,E_t'>
6. r_s in Dom(R)                                                     | 5
   r_t in Dom(R)
7. pc_G, pc_B in Dom(R)                                              | Inversion of (S-t)
8. (R,C,M,Q,op r_d,r_s,r_t)                                          | (op2r), 6, 7
    -->_0 (R++[r_d --> R(r_s) op R(r_t)],C,M,Q,.)                    [From this point on, will assume existence of
                                                                      registers proved as in this case]

*



Case op1r:
-------

Similar to op2r.
*



Case mov-n:                                                            ** Change Summary:** simple (v -> n)
-------

1.  pc_G, pc_B in Dom(R)                                             | Inversion of (S-t)
2. (R,C,M,Q, mov r_d, n) -->_0 (R++[r_d--> n],C,M,Q,.)               | mov
*



Case mov-reg:
-------------

Similar to mov-n.
*
```

```
Case ld_G:                                                           ** Change Summary:** simple (add init flag)
-------

1. |- (R,C,M,Q, ld_G r_d r_s)                                        | Given
2. P;(.;S(G);S(seq(E_d,E_s));S(E_m);S(s)) |- ld_G r_d r_s => RT      | Inversion of (S-t, C-t), substitution, 1
3. S(G)(r_s) =< <G,b refh,E_s'>                                      | Inversion of (ld_G-t), 2
4. P|- R : S(G)                                                      | Inversion of (S-t), 1
5. P;.|- R(r_s) : < G,b refh,E_s'>                                   | Inversion of (reg-file-t), 4, 3
subcase on the definition of find

subcase a.  find(Q,R_val(r_s)) = ()
6a. R_val(r_s) in Dom(M)                                             | Canonical Forms, Inversion of (S-t), 1, 5
7a. (R,C,M,Q, ld_G r_d,r_s)                                          | (ld_G-mem), assumption, 6a
      -->_0 (R++[r_d--> G M(R(r_s))],C,M,Q,.)

subcase b.  find(Q,R_val(r_s)) = (R_val(r_s),n)
6b. (R,C,M,Q, ld_G r_d,r_s) -->_0 (R++[r_d --> n],C,M,Q,.)           | (ld_G-queue), assumption
*



Case ld_B:
----------

Similar to ld_G.
*



Case sld_c:                                                          ** Change Summary:** New Case
-----------

1. |- (R,C,M,Q, sldc rd n)                                           | Given
2. P;(.;S(G);S(seq(E_d,E_s));S(E_m);S(s)) |- sldc rd n => RT         | Inversion of (S-t, C-t), substitution, 1
3. P |- R : S(G)                                                     | Inversion of (S-t), 1

4. S(G)(spc) = <c,sptr,Es'>                                          | Inversion of (sldc-t), 2
5. P;. |- R(spc) : <c,sptr,Ec>                                       | Inversion of (reg-file-t), 3, 4
   . |- Ec + n = En
   .;S(s) |- En : <c,b,E>

6. . |- En = R(spc) + n                                              | Canonical Forms, (Inversion on (S-t), 1), 5, Exp Eq Trans

7. Exists l. . |- En = l and l in Dom(M)                             | Valid Stack Loc Lemma, (Inversion of (S-t), 1), 7
8. R(spc)+n in Dom(M)                                                | Exp Eq Trans Lemma, 6, 7

10. |- (R,C,M,Q, sldc r_d n) -->_0 (R++[rd -> M(R(spc)+n)],C,M,Q,.) | (sldc), 8
*



Case st_G:
----------

1. |- (R,C,M,Q, st_G r_d, r_s) -->_0 (R++,C,M,((R(r_d),R(r_s)),Q),.) | (st_G-queue)
*



Case st_B:                                                           ** Change Summary:**  simple (add s, init flags)
----------

a1. |- (R,C,M,Q, st_B r_d r_s)                                       | Given

1. P |- C                                                            | Inversion of (S-t), a1
2. Forall c=/=Z. C(R(pc_c)) = st_b rd rs                             | Inversion of (S-t), a1
3. P;(D;G;seq(E_d,E_s)(E_d',E_s');E_m;s) |- st_b rd rs               | Inversion of (C-t), 1, 2, inspection of (st_B-t)
   ==> (D;G++;seq(E_d,E_s);upd E_m E_d' E_s')
4. Exists S. . |- S : D                                              | Inversion of (S-t), a1
5. P;(.;S(G); S(seq(E_d,E_s),(E_d',E_s'));S(E_m);S(s))               | substitution lemma, 4, 3
   |- st_B r_d r_s
    => (.;S(G)++;S(seq{(E_d,E_s)});S(upd E_m E_d' E_s');S(s))

6. P |- R : S(G)                                                     | Inversion of (S-t), a1
7. S(G)(r_d) =< <B,b refh,E_d''>                                     | Inversion of (st_B-t), 5
   S(G)(r_s) = <B,b,E_s''>
8.  P;.|- R(r_s) : < B,b,E_s''>                                      | Inversion of (R-t), 6, 7
    P;.|- R(r_d) : < B,b refh,E_d''>
9. .|- R_val(r_s) = E_s''                                            | Canonical Forms, (Inversion of (S-t), 1), 8, 9
   .|- R_val(r_d) = E_d''

10. .|- S(E_s') = E_s''                                              | Inversion of (st_B-t), 5
```

```
    .|- S(E_d') = E_d''

11. P |- Q : S(seq(E_d,E_s),(E_d',E_s'))                          | Inversion of (S-t), a1, Inversion of (heap-t)
12. Q = (seq(n,n'),(n_l,n_l')) where .|- S(E_d')=n_l  and .|- S(E_s')=n_l' | Inversion of (Q-t), 11

13. R_val(r_s) = n_l'  and R_val(r_d) = n_l                        | Exp Eq Transitivity, 9, 10, 11
14. (R,C,M,(seq(n,n'),(n_l,n_l')),st_B  r_d,r_s)                   | (st_B-mem), 14
     -->_0^(n_l,n_l') (R++,C,M[n_l --> n_l'],seq{(n,n')},.)
*




Case sst:                                                          ** Change Summary:** New Case
---------

a1. |- (R,C,M,Q,sst n rv)                                          | Given

1. P;(.;S(G);S(seq(E_d,E_s));S(E_m);S(s)) |- sst n rv => RT        | Inversion of (S-t, C-t), substitution, a1
2. P |- R : S(G)                                                   | Inversion of (S-t), 1

3. S(G)(spg) = <G,sptr,Eg>                                         | Inversion of (sst-t),1
4. S(G)(spb) = <B,sptr,Eb>                                         | Inversion of (sst-t), 1
5. . |- Eg = Eb                                                    | Inversion of (sst-t), 1

6. P;. |- R(spg) = <G,sptr,Eg>                                     | Inversion of (reg-file-t), 2, 3
7. P;. |- R(spb) = <B,sptr,Eb>                                     | Inversion of (reg-file-t), 2, 4
8. . |- R(spg) = Eg  and . |- R(spb) = Eb                          | Canonical Forms, Inversion of S-t, a1, 6, 7
9. . |- R(spg) = R(spb)                                            | Exp Eq Trans, 5, 8

10. . |- Eg + n = En                                               | Inversion of (sst-t)
11. . |- S(s)[En --> <c,b,Ev>] = s'                                | Inversion of (sst-t)
12. Exists l. . |- En = l and l in Dom(M)                          | Valid Stack Loc Lemma, (Inversion of (S-t),1), 11
13. R(spg) + n in Dom(M)                                           | Exp Eq Trans, 8, 10, 12

14. (R,C,M,Q,sst n rv) -->_0 (R++,C,M[R(spg)+n -> R(rv)],Q,.)      | (sst), 13, 9
*






Case bz_G:                                                         ** Change Summary:** simple (add s)
----------

1. |- (R,C,M,Q, bz_G r_z,r_d)                                      | Given
2. P;(.;S(G); S(seq(E_d,E_s));S(E_m);S(s)) |- bz_G r_z r_d => RT   | Inversion of (S-t, C-t), substitution, 1
3. S(G)(d) = <G,int,0>                                             | Inversion of (bz_G-t), 2
   S(G)(r_z) = <G,int,E_z>
   S(G)(r_d) = <G,T->void,E_d'>
4. P |- R : S(G)                                                   | Inversion of (S-t), 1
5. P;.|- R(d) : <G,int,0>                                          | Inversion of (reg-file-t), 4, 3
   P;.|- R(r_z) : < G,int,E_z>
   P;.|- R(r_d) : <G,T->void,E_d'>
6. R_val(d) = 0                                                    | Canonical Forms, (Inversion of (S-t), 1)), 5
   r_z in Dom(R), r_d in Dom(R)
7. (R,C,M,Q, bz_G r_z,r_d) -->_0 (R++[d--> R(r_d)],C,M,Q,.) or     | bz_G-taken or bz-untaken, 6
   (R,C,M,Q, bz_G r_z,r_d) -->_0 (R++,C,M,Q,.)
*




Case bz_B:                                                         ** Change Summary:** simple (add s, remove color)
----------

1. |- (R,C,M,Q, bz_G r_z,r_d)                                      | Given
2. P;(.;S(G); S(seq(E_d,E_s));S(E_m);S(s)) |- bz_B r_z r_d => RT   | Inversion of (S-t,C-t), substitution, 1
3. S(G)(r_z) = <B,int,E_z>                                         | Inversion of (bz_B-t), 2
   S(G)(r_d) = <B,(D';G';seq(E_d',E_s');E_m')--> void,E_r>
   S(G)(d) = ( E_z'=0 => < G,T'--> void,E_r'> )
   T' = (D';G';seq(E_d',E_s');E_m')
   .|- E_z = E_z'
   .|- E_r = E_r'
4. P |- R : S(G)                                                   | Inversion of (S-t), 1
5. P;.|- R(d) : E_z'=0 => < G,T'--> void,E_r'>                     | Inversion of (reg-file-t), 4, 3
   P;.|- R(r_z) : < B,int,E_z>
   P;.|- R(r_d) : < B,(D';G';seq{(E_d',E_s')};E_m')--> void,E_r>
6. .|- R(r_z) = E_z                                                | Canonical Forms, Inversion of (S-t), 1, 5
   .|- R(r_d) = E_r
7. . |- R(d)=0 and .|- E_z'=/=0                                    | Canonical Forms, Inversion of (S-t), 1, 5
   or .|- R(d) = E_r' and .|- E_z'=0 and . |- E_r'=/=0

Case a: R(d) = 0 and .|- E_z'=/=0
8a. . |- R(r_z) = E_z'                                             | Exp Eq Transitivity, 3, 6
9a. R(r_z) =/= 0                                                   | Exp Eq Transitivity, 8a, assumption
```

```
10a. (R,C,M,Q,bz_B r_z,r_d) -->_0 (R++,C,M,Q,.)                    | bz-untaken, assumption, 9a

Case b: .|-R(d) = E_r' and  .|- E_z'=0 and . |- E_r'=/=0
8b. R(r_z) = 0                                                      | Exp Eq Transitivity, 6, 3, assumption
9b. R(r_d) = R_val(d)                                               | Exp Eq Transitivity, 3, 6, assumption
10b. R(d) =/= 0                                                     | Exp Eq Transitivity, assumption
11b. (R,C,M,Q, bz_B r_z,r_d)                                        | (bz_B-taken), 8b, 9b, 10b
     -->_0 (R[pc_G--> R(d)][pc_B--> R(r_d)][d--> 0],C,M,Q,.)
*



Case jmp_G:
-----------

Similar to bz_G.
*



Case jmp_B:
-----------

Similar to bz_B.
*




Case malloc:                                                       ** Change Summary:** new case
------------

1. |- (R,C,M,Q, malloc[b] rg rb)                                    | Given
2. M = Ms #Dom(C) Ms                                                | Inversion of (S-t), 1
3. P |- M_s : s                                                     | Inversion of (S-t), 1
4. Dom(Ms) =/= .                                                    | Inversion of (s-t-cons) and (s-t-base), 2
5. Dom(M) =/= .                                                     | Inversion of #, 2, 4
6. n = max(Dom(M)) + 1                                              | 5
7. (R,C,M,Q, malloc[b] rg rb) -->_0 (R++[rg->n][rb->n],C,M,Q,.)     | (malloc), 6
*



Case salloc:                                                       ** Change Summary:** new case
------------

1. |- (R,C,M,Q, salloc n)                                           | Given
2. M = Ms #Dom(C) Ms                                                | Inversion of (S-t), 1
3. P |- M_s : s                                                     | Inversion of (S-t), 1
4. Dom(Ms) =/= .                                                    | Inversion of (s-t-cons) and (s-t-base), 2
5. Dom(M) =/= .                                                     | Inversion of #, 2, 4
6. m = min(Dom(M))                                                  | 5
7. (R,C,M,Q,salloc n) -->_0                                         | (salloc), 6
   (R++[spg -> R(spg)-n][spb -> R(spb)-n],C,
    (M,m-1 -> 0, ..., m-n -> 0), Q, .)
*


Case sfree:                                                        ** Change Summary:** new case
------------

1. |- (R,C,M,Q, sfree n)                                            | Given

2. M = Ms #Dom(C) Ms                                                | Inversion of (S-t), 1
3. P |- M_s : s                                                     | Inversion of (S-t), 1
4. Dom(Ms) =/= .                                                    | Inversion of (s-t-cons) and (s-t-base), 2
5. Dom(M) =/= .                                                     | Inversion of #, 2, 4
6. m = min(Dom(M))                                                  | 5

7. P;(.;S(G); S(seq(E_d,E_s));S(E_m);S(s)) |- sfree n => RT         | Inversion of (S-t,C-t), substitution, 1

8. S(s) = Et:t :: ... :: Ef: us                                     | Inversion of (sfree-t), 8
9. . |- Ef = Et + n
10. . |- Et = Eg
11. G(spg) = <G,sptr,Eg>

12. . |- R(spg) = Eg                                                | Canonical Forms, Inversion of (S-t), 1, 12
13. . |- R(spg) + n = Ef                                            | Exp Eq Trans, 9, 10, 11
14. R(spg) through R(spg)+n  in Dom(M)                              | Valid Stack Loc Lemma, 9, (Exp Eq Trans, 9, 10, 11, 13)
15. M = M',m -> nm, (m+n-1) -> nmn1                                 | 15

16. (R,C,M,Q,sfree n) -->_0                                         | (sfree), 6, 15
   (R++[spg -> R(spg)+n][spb -> R(spb)+n],C, M',Q,.)
*
```

Progress Part 2

```
2. If |-c (R,C,M,Q,ir) then (R,C,M,Q,ir) -->_0^s  S


Proof by case analysis on ir.

** Change Summary:**: COMPLETE 4/11/08

For all cases, operand registers can be shown to exist using typing information.


Case .:
~~~~~~~

subcase: R(pc_G) = R(pc_B)
1a. all c' =/= c. P(R_val(pc_c')) = (D;G;seq(E_d,E_s),E_m,s) --> void   | Inversion of (S-t), 1
2a. either R(pc_G) in Dom(C) or R(pc_B) in Dom(C)                        | 1a
3a. R(pc_G) in Dom(C)                                                    | assumption, 2a
4a. rule fetch applies                                                   | assumption, 3a

subcase: R(pc_G) =/= R(pc_B)
1b. rule fetch-fail applies                                              | assumption

*



Case op2r:
~~~~~~~~~~

Rule op2r applies.
*



Case op1r:
~~~~~~~~~~

Rule op1r applies.
*




Case mov:
~~~~~~~~~

Rule mov-n or mov-reg applies depending on syntax.
*



Case ld_G:
~~~~~~~~~~

subcase a: find(Q,R(r_s)) = (R(r_s),n)
Rule ld_G-queue applies.

subcase b: find(Q,R(r_s)) = () and R(r_s) in Dom(M)
Rule ld_G-mem applies

subcase c: find(Q,R(r_s)) = () and R(r_s) not in Dom(M)
Rule ld_G-fail or ld_G-rand applies
*



Case ld_B:
```

```
~~~~~~~~~~

Similar to case ld_G.
Rules ld_B-mem or ld_B-fail or ld_B-rand apply.
*




Case sld:
~~~~~~~~~

subcase a: R(spc) + n in Dom(M)
Rule sld_c applies

subcase R(spc) + n not n Dom(M)
Rule sld_c-fail applies
*





Case st_G:
~~~~~~~~~~

Rule st-G-queue applies.
*








Case st_B:
~~~~~~~~~~

a1. |- (R,C,M,Q, st_B r_d r_s)                                     | Given

1. P |- Q : S(seq(E_d,E_s),(E_d',E_s'))                            | Inversion of (S-t), a1, Inversion of (heap-t)
2. Q = (seq(n,n'),(n_l,n_l'))                                      | Queue Lemma, 1

subcase a: end of queue matches r_d / r_s
aa2. Rval(r_d) = n_l and Rval(r_s) = n_l'
3a. (R,C,M,(seq(n,n'),(n_l,n_l')),st_B  r_d,r_s)                   | st_B-mem, 2, aa2
     -->_0^(n_l,n_l') (R++,C,M[n_l --> n_l'],seq{(n,n')},.)
subcase 2: end of queue does not match r_d / r_s
ab2. Rval(r_d) =/= n_l or Rval(r_s) =/= n_l'
3b. (R,C,M,Q,st_B  r_d,r_s) --> fault                              | st_B-mem-fail, 2, ab2
*




Case bz_G:
~~~~~~~~~~

subcase a:  R(d) =/= 0
8a. either bz-untaken-fail or bz_G-taken-fail applies               | assumption

subcase b:  R_val(d) = 0
10b. either bz-untaken or bz_G-taken applies                        | assumption
*






Case bz_B:


subcase a: Rval(r_z) not= 0
Rule bz-untaken or bz-untaken-fail applies

subcase b: Rval(r_z) = 0
subsubcase ba: Rval(d) not= 0  and Rval(rd) = Rval(d)
Rule bz_B-taken applies
subsubcase bb: Rval(d) = 0  or Rval(rd) not= Rval(d)
Rule bz_B-taken-fail applies
*
```

```
Case jmp_G:

Similar but simpler than case bz_G.
Rule jmp_G or jmp_G-fail applies.
*


Case jmp_B:

Similar but simpler than case bz_B.
Rule jmp_B or jmp_B-fail applies.
*
```

Preservation Part 1

```
1. If |-Z (R,C,M,Q,ir)
   and (R,C,M,Q,ir)  -->_0^s  (R',C',M',Q',ir')
   then |-Z (R',C',M',Q',ir')


Proof by induction on the structure of the derivation of (R,C,M,Q,ir)  -->_0^s  (R',C',M',Q',ir').


***CHANGE SUMMARY:*** COMPLETE 4/10/08, changes 5/30/08

for all cases, replaced old #2 with new #2, old #7 with modified #7, old #8 with new #8,
changed colored values c n to regular values n, modified step 5 to include stack type



CASE fetch:                                                             ** Change Summary:** simple

(p1) R(pc_G) = R(pc_B)
(p2) R(pc_G) in Dom(C)
 ---------------------------------------- (fetch)
(R,C,M,Q,.) -->_0 (R,C,M,Q,C(R(pc_G)))


0. |-Z (R,C,M,Q,.)                                                      | Given
1. Dom(P) = Dom(C) union Dom(M_m)                                       | Inversion of (S-t), 0
2. M = M_s #Dom(C) M_m                                                  | Inversion of (S-t), 0
3. P |- C                                                               | Inversion of (S-t), 0
4. <unnecessary>
5. Forall c=/=Z. P(R(pc_c)) = (D;G;seq(E_d,E_s);E_m;s)-->void           | Inversion of (S-t), 0
6. Exists S. .|- S : D                                                  | Inversion of (S-t), 0
7. P |-Z M_s : S(s)                                                     | Inversion of (S-t), 0
8. P |-Z (M_m,Q) : (S(E_m), S(seq(E_d,E_m)))                            | Inversion of (S-t), 0
9. P |-Z R : S(G)                                                       | Inversion of (S-t), 0

4'. Forall c=/=Z. C(R(pc_c)) = C(R(pc_G))                               | (p1),(p2)
10.  |-Z (R,C,M,Q,C(R(pc_G)))                                           | (S-t), 1,2,3,4',5,6,7,8,9
*



CASE fetch-fail:                                                        ** Change Summary:** none

R(pc_G) =/= R(pc_B)
--------------------------(fetch-fail)
(R,C,M,Q,.)  -->_0  fault

does not apply (fails second assumption)
*




CASE op2r:                                                              ** Change Summary:** simple

R2 = R++[ r_d -> R(r_s) op R(r_t) ]
---------------------------------------------------------(op2r)
(R,C,M,Q, op r_d, r_s, r_t )  -->_0  (R2,C,M,Q,.)


0. |-Z (R,C,M,Q,op r_d, r_s, r_t)                                       | Given
1. Dom(P) = Dom(C) union Dom(M_m)                                       | Inversion of (S-t), 0
2. M = M_s #Dom(C) M_m                                                  | Inversion of (S-t), 0
3. P |- C                                                               | Inversion of (S-t), 0
4. Forall c=/=Z. C(R(pc_c)) = op r_d, r_s, r_t                          | Inversion of (S-t), 0
5. Forall c=/=Z. P(R(pc_c)) = (D;G;seq(E_d,E_s);E_m;s)-->void           | Inversion of (S-t), 0
6. Exists S. . |- S : D                                                 | Inversion of (S-t), 0
7. P |-Z M_s : S(s)                                                     | Inversion of (S-t), 0
8. P |-Z (M_m,Q) : (S(E_m), S(seq(E_d,E_m)))                            | Inversion of (S-t), 0
9. P |-Z R : S(G)                                                       | Inversion of (S-t), 0
```

```
Let R2 = R++[r_d --> R(r_s) op R(r_t)]
Let G2 = G++[r_d --> <c,int, E_s' op E_t'> ]

10. P;(D;G;seq(E_d,E_s);E_m;s) |- op r_s,r_t,r_d ==> RT2              | Inversion of (C-t), 3, 4
11. RT2 = (D;G2;seq(E_d,E_s);E_m;s)                                  | Inspection of (op2r-t), def of G2
12. Forall c=/=Z. P(R(pc_c)+1) = RT2 --> void                        | Inversion of (C-t), 3, 4, 11

5'. Forall c=/=Z. P(R2_val(pc_c)) = (D;G2;seq(E_d,E_s);E_m;s) --> void  | def of ++, def of R2, 11, 12

13. P;(D;G;seq(E_d,E_s);E_m;s)|- op r_s,r_t,r_d                      | 10, 11
    ==>(D;G2;seq(E_d,E_s);E_m;s)
14. P;(.;S(G);S(seq(E_d,E_s));S(E_m);S(s))|- op r_s,r_t,r_d          | substitution, 6, 13
      ==> (.;S(G2);S(seq(E_d,E_s));S(E_m);S(s))

15. S(G)(r_s) = <c,int,E_s'>                                         | Inversion of (op2r-t), 13
16. S(G)(r_t) = <c,int,E_t'>                                         | Inversion of (op2r-t), 13
17. Forall a. P;. |-Z R(a) : S(G)(a)                                 | Inversion of (reg-file-t), 9

18. S(G)(pc_G) =< <G,int,E_G>    and  S(G)(pc_B)   =< <B,int,E_B>    | Inversion of (reg-file-t), 9
19. S(G)++(pc_G)=< <G,int,E_G+1> and  S(G)++(pc_B) =< <B,int,E_B+1>  | 18, def G++
19a. . |- E_G = E_B                                                  | Inversion of (reg-file-t), 9
19b. [[E_G]] = [[E_B]]                                               | Inversion of 19a, def of [[]]
19c. [[E_G]] + [[1]] = [[E_B]] + [[1]]                               | 19b, def of [[]]
19d. [[E_G + 1]] = [[E_B + 1]]                                       | 19c, def of [[]]
19e. . |- E_G + 1 = E_B + 1                                          | 19d, (E-eq)
20. P |-Z R++ : S(G++)                                               | (reg-file-t), def of R++, 19, 19e

21. P;. |-Z R(r_s) : <c,int,E_s'>  and P;. |-Z R(r_t) : <c,int,E_t'>    | Inversion of (reg-file-t), 9, 15, 16

SUBCASE a: Z =/= c
21a. . |- E_s' = R(r_s)  and . |- E_t' = R(r_t)                      | Cannonical Forms, 1, 7, 3, 21, assumption
22a. [[E_s']] = [[R(r_s)]]  and [[E_t']] = [[R(r_t)]]                | Inversion on (E-eq), 21a
23a. [[R(r_s)]] op [[R(r_t)]]  =  [[E_s']] op [[E_t']]               | Subst of Eq for Eq, 22a
24a. [[R(r_s) op R(r_t)]] = [[E_s' op E_t']]                         | def of [[]], 23a
25a. . |- E_s' : kint and . |- E_t' : kint                           | Inversion on (E-eq), 21a
26a. . |- (E_s' op E_t') : kint                                      | (E-op-t), 25a
27a. . |- (R(r_s) op R(r_t)) : kint                                  | (E-int-t)
28a. . |- (R(r_s) op R(r_t)) = (E_s' op E_t')                        | (E-eq), 26a, 27a, 24a
29a. P |- (R(r_s) op R(r_t)) : int                                   | (int-t)
30a. P;.|-Z (R(r_s) op R(r_t)) : <c,int, E_s' op E_t'>               | (val-t), 29a, 28a

SUBCASE b: Z = c
20b. . |- E_s' : kint and . |- E_t' : kint                           | Int Kinding Lemma, 21
21b. . |- E_s' op E_t' : kint                                        | (E-op-t), 20b
22b. P;.|-Z (R(r_s) op R(r_t)) : <c,int, E_s' op E_t'>               | (val-zap-t), assumption, 21b

MERGE:
31. P;.|-Z R2(r_d) : S(G2(r_d))                                      | 30a/22b, def of R2, def of G2,
9'. P |-Z R2 : S(G2)                                                 | def of R2, def of G2, 20, 31

25. |-Z (R2,C,M,Q,.)                                                 | (S-t), 1,2,3,ir=.,5',6,7,8,9'
*




CASE op1r:                                                           ** Change Summary:** none
~~~~~~~~~~

 R2 = R++[ rd -> R(rs) op n ]
----------------------------------------------(op1r)
(R,C,M,Q, op rd, rs, n )  -->_0  (R2,C,M,Q,.)

Similar to op2r.
*




CASE mov:                                                            ** Change Summary:** none
~~~~~~~~~

-----------------------------------------------------------(mov)
(R,C,M,Q, mv rd, v )  -->_0  (R++[rd -> v] ,C,M,Q,.)

Similar to op2r.
*


CASE mov-reg:                                                        ** Change Summary:** new case
~~~~~~~~~~~~~
```

```
----------------------------------------------------------(mov-reg)
(R,C,M,Q, mv rd, rs )  -->_0  (R++[rd -> R(rs)] ,C,M,Q,.)

Similar to op2r.
*



CASE malloc:                                                          ** Change Summary:** New Case -- not trivial
~~~~~~~~~~~~

(p1) n = max(Dom(M)) + 1
------------------------------------------------------------------------- (malloc)
(R,C,M,Q,malloc[b] rg rb) -->_0 (R++[rg -> n][rb -> n], C, (M,n->0), Q, .)

0. |-Z (R,C,M,Q,malloc[b] rg rb)                                        | Given
1. Dom(P) = Dom(C) union Dom(M_m)                                       | Inversion of (S-t), 0
2. M = M_s #Dom(C) M_m                                                  | Inversion of (S-t), 0
3. P |- C                                                               | Inversion of (S-t), 0
4. Forall c=/=Z. C(R(pc_c)) = malloc[b] rg rb                           | Inversion of (S-t), 0
5. Forall c=/=Z. P(R(pc_c)) = (D;G;seq(E_d,E_s);E_m;s)-->void           | Inversion of (S-t), 0
6. Exists S. . |- S : D                                                 | Inversion of (S-t), 0
7. P |-Z M_s : S(s)                                                     | Inversion of (S-t), 0
8. P |-Z (M_m,Q) : (S(E_m), S(seq(E_d,E_m)))                            | Inversion of (S-t), 0
9. P |-Z R : S(G)                                                       | Inversion of (S-t), 0

Let R2 = R++[rg -> n][rb -> n]
Let M_m2 = M_m,n->0
Let M2 = M, n-> 0

Let G2 = G++[rg -> <G,b ref0,x>][rb -> <B,b ref0, x>]
Let D2 = D,x:int
Let E_m2 = upd E_m x 0
Let P2 = P, n -> S(b) ref0
Let S2 = S,n/x



10. P;(D;G;seq(E_d,E_s);E_m;s) |- malloc[b] rg rb ==> RT2               | Inversion of (C-t), 3, 4
11. RT2 = (D2;G2;seq(E_d,E_s);Em2;s)                                    | Inspection of (op2r-t), def of G2
12. Forall c=/=Z. P(R(pc_c)+1) = RT2 --> void                           | Inversion of (C-t), 3, 4, 11
12. Forall c=/=Z. P(R2_val(pc_c)) = (D2;G2;seq(E_d,E_s);Em2;s) --> void | def of ++, def of R2, 11, 12
5'. Forall c=/=Z. P2(R2_val(pc_c)) = (D2;G2;seq(E_d,E_s);Em2;s) --> void| 12, 30 (forward reference)

13. P;(D;G;seq(E_d,E_s);E_m;s)|- malloc[b] rg rb                        | 10, 11
    ==>(D2;G2;seq(E_d,E_s);E_m2;s)
14. P;(.;S(G);S(seq(E_d,E_s));S(E_m);S(s))|- malloc[S(b)] rg rb         | substitution, 6, 13
      ==> (x:kint;S(G2);S(seq(E_d,E_s));S(E_m2);S(s))

15. x not in D                                                          | Inversion of (malloc-t), 13


16. Dom(M_m2) = Dom(M_m) union {n}                                      | def of M_m2
17. Dom(C) union Dom(M_m2) = Dom(C) union Dom(M_m) union {n}            | 16
18. Dom(P2) = Dom(P) union {n}                                          | def of P2
1'. Dom(P2) = Dom(C) union Dom(M_m2)                                    | 1, 18, 18


19. Dom(M) = Dom(M_s) union Dom(M_m)                                    | Def of #, 2
20. Dom(M_s) union Dom(M_m2) = Dom(M_s) union Dom(M_m) union {n}        | Def of M_m2
21. Dom(M2) = Dom(M) union {n}                                          | Def of M2
22. Dom(M2) = Dom(M_s) union Dom(M_m2)                                  | 19, 20, 21

23. Dom(M_s) intersect Dom(M_m) = empty                                 | Def of #, 2
24. n not in Dom(M_s)                                                   | (p1), Def of #, 2
25. Dom(M_s) intersect Dom(M_m2) = empty                                | 23, Def of M_m2, 24

26. All ls in Dom(M_s).All lc in Dom(C).All lm in Dom(M_m). ls<lc<lm    | Def of #, 2
27: All l in Dom(M). l < n                                              | (p1)
28. All ls in Dom(M_s) union Dom(C). l < n                              | 27, 26
29: All ls in Dom(M_s).All lc in Dom(C).All lm in Dom(M_m2). ls<lc<lm   | 26, 28

2'. M2 = M_s #Dom(C) M_m2                                               | Def of #, 22, 25, 29


30. n not in Dom(C)                                                     | 26, (p1)
3'. P2 |- C                                                             | (C-t), (Inversion of (C-t), 3), 30, Psi Extension Lemma

31. . |- n : kint
32. x not in Dom(.) union Dom(D)                                        | 15
6'. . |- S2 : D2                                                        | (sub-t), Def of S2, Def of D2, 6, 31, 32


33. [[S(E_m)]] = M_m                                                    | Inversion of (heap-t), 8
34. P |-Z Q : S(seq(E_d,E_s))
```

```
35. all l in Dom(M_m). exists f. P;M;Q |-Z l : b reff

36. n not in Dom(M_m)                                                 | (p1), def of #, 2
37. M_m2 = M_m[n -> 0]                                                | def of M_m2, 36
38. M_m2 = [[S(E_m)]] [ n -> 0 ]                                      | 35, 37
39. M_m2 = [[S(E_m)]] [ [[n]] -> [[0]]]                               | 38, def of [[]]
40: M_m2 = [[upd S(E_m) n 0]]                                         | def of [[]], 39
41. M_m2 = [[S2(upd E_m n 0)]]                                        | Substitution Extension Lemma, 6, 15, 40

42. P |-Z Q : S2(seq(E_d,E_s))                                        | Substitution Extension Lemma, 6, 15, 34

43. P2 |- n : S(b) ref0                                               | def of P2, (addr-heap-t)
44. P2;M;Q |- n : S(b) ref0
45. all l in Dom(M_m). exists f. P2;M;Q |-Z l : b reff                | Heap Extension Lemma, 35, def of P2
46. all l in Dom(M_m2). exists f. P2;M;Q |-Z l : b reff               | Def of M_m2, Def of P2, 44, 45

8'. P |-Z (M_m2,Q) : (S2(upd E_m n 0), S2(seq(E_d,E_s)))              | (heap-t), 31, 42, 46


47. P2 |- M_m2 : S2(upd E_m x 0)                                      | Substitution Extension Lemma, 6, def of S2, 15, 7
7'. P2 |-Z M_s : S2(s)                                                | Heap Extension Lemma, 7, Subst Extension Lemma, 6, 15


47. . |- n = n                                                        | (E-eq)
48. P2 |- n : S(b) ref0                                               | Def of P2, (addr-heap-t)
49. P2 |-Z n : <G, S(b) ref0, n>                                      | (val-t), 47, 48
50. P2 |-Z n : <G, S2(b) ref0, n>                                     | Subst Ext Lemma, 6, 15, def of S2
51. P2 |-Z n : S2(<G, b ref0, x>)                                     | 50, Def of S2
52. P2 |-Z R2(rg) : S2(G2)(rg)                                        | 51, Def of R2, def of G2
53. P2 |-Z R2(rb) : S2(G2)(rb)                                        | as in 52
54. P2 |-Z R++ : S(G++)                                               | 9, def of ++
55. P2 |-Z R++ : S2(G++)                                              | Subst Extension Lemma, 6, 15, 54
9'. P2 |-Z R2 : S2(G2)                                                | (R-t), 55, 53, 52, def of R2, def of G2

56. |-Z (R2, C, M2, Q, .)                                             | (S-t), 1', 2', 3', ir=., 5', 6', 7', 8', 9'
*




CASE salloc:                                                          ** Change Summary:** New Case -- not trivial
~~~~~~~~~~~~

(p1) m = min(Dom(M))
---------------------------------------------------------------------------------------------------- (salloc)
(R,C,M,Q,salloc n) -->_0 (R++[spg -> R(spg)-n][spb -> R(spb)-n], C, (M,m-1 -> 0,...,m-n->0), Q, .)


0. |-Z (R,C,M,Q,salloc n)                                             | Given
1. Dom(P) = Dom(C) union Dom(M_m)                                     | Inversion of (S-t), 0
2. M = M_s #Dom(C) M_m                                                | Inversion of (S-t), 0
3. P |- C                                                             | Inversion of (S-t), 0
4. Forall c=/=Z. C(R(pc_c)) = salloc n                                | Inversion of (S-t), 0
5. Forall c=/=Z. P(R(pc_c)) = (D;G;seq(E_d,E_s);E_m;Et:us)-->void     | Inversion of (S-t), 0
6. Exists S. . |- S : D                                               | Inversion of (S-t), 0
7. P |-Z M_s : S(Et:us)                                               | Inversion of (S-t), 0
8. P |-Z (M_m,Q) : (S(E_m), S(seq(E_d,E_m)))                          | Inversion of (S-t), 0
9. P |-Z R : S(G)                                                     | Inversion of (S-t), 0

Let R2 = R++[spg -> R(spg)-n][spb -> R(spb)-n]
Let M_s2 = M_s,m-1 -> 0,...,m-n->0
Let M2 = M,m-1 -> 0,...,m-n->0

Let G2 = G++[spg -> <G,sptr,Eg-n>][spb -> <B,sptr,Eb-n>]
Let s2 = (Et-n) : ns  ::  (Et-n+1) : ns  ::  ...  :: Et : us

10. P;(D;G;seq(E_d,E_s);E_m;s) |- salloc n ==> RT2                    | Inversion of (C-t), 3, 4
11. RT2 = (D;G2;seq(E_d,E_s);Em;s2)                                   | Inspection of (op2r-t), def of G2
12. Forall c=/=Z. P(R(pc_c)+1) = RT2 --> void                         | Inversion of (C-t), 3, 4, 11

5'. Forall c=/=Z. P(R2_val(pc_c)) = (D;G2;seq(E_d,E_s);Em2;s2) --> void | def of ++, def of R2, 11, 12

13. P;(D;G;seq(E_d,E_s);E_m;Et:us)|- salloc n                         | 10, 11
    ==>(D;G2;seq(E_d,E_s);E_m;s2)
14. P;(.;S(G);S(seq(E_d,E_s));S(E_m);S(s))|- salloc n                 | substitution, 6, 13
      ==> (.;S(G2);S(seq(E_d,E_s));S(E_m);S(s2))


15. S(G)(spg) = <G,sptr,Eg>                                           | Inversion of (salloc-t), 14
16. S(G)(spb) = <B,sptr,Eb>
17. . |- Eg = Eb
```

```
18. . |- Eg = Et


19. Dom(M) = Dom(M_s) union Dom(M_m)                                    | Inversion of #-def, 2
20. Dom(M2) = Dom(M_s2) union Dom(M_m)                                  | 19, def of M_s2, def of M_2
21. {m-1,...,m-n} intersect Dom(M) = empty                              | (p1)
22. Dom(M_s) intersect Dom(M_m) = empty                                 | Inversion of #-def, 2
23. Dom(M_s2) intersect Dom(M_m) = empty                                | 22, def of M_s2, 21, 19
24. All ls in Dom(M_s). all Lc in Dom(C). all Lm in Dom(M_m).           | Inversion of #-def, 2
    ls < lc < lm
25. All ls in Dom(M_s2). all Lc in Dom(C). all Lm in Dom(M_m).          | 24, (p1), def of M_s2
    ls < lc < lm
2'. M2 = M_s2 #Dom(C) M_m                                               | (#-def), def of M2, 20, 23, 25


26. P;. |-Z 0 : ns                                                      | (ns-t)
27. . |- Et - 1 + 1 = Et                                                | arithmetic
28. . |- S(Et : us) wf                                                  | Inversion of (s-t-cons), 7
29. . |- S(Et-1 : ns :: Et : us) wf                                     | (s-wf-cons), 27, 28, Subst Closed Exp Lemma
30. . |- m = Et                                                         | Inversion of (s-t-cons), 7, (p1)
31. . |- m-1 = Et-1                                                     | 30
27. P |-Z M_s,m-1->0 : S(Et-1 : ns :: Et : us)                          | (s-t-cons), 29, 31, 26, 7
7'. P |-Z M_s2 : S(s2)                                                  | repeated applications of (s-t-cons),
                                                                           def of M_s2, def of s2, 27


29. P;. |-Z R(spg) - n : <G,sptr,Eg - n>                                | 15, case on Z: deconstruct and reconstruct using
                                                                           whichever (val-t) rule applies
30. P;. |-Z R(spb) - n : <B,sptr,Eb - n>                                | 16, case on Z: deconstruct and reconstruct using
                                                                           whichever (val-t) rule applies
31. P; . |-Z R2(spg) : S(G2)(spg)                                       | 29, def of R2, G2
32. P; . |-Z R2(spb) : S(G2)(spb)                                       | 30, def of R2, G2
33. P |-Z R++ : S(G++)                                                  | 9, def of ++
9'. P |-Z R2 : S2(G2)                                                   | (R-t), def of R2, def of G2, 31, 32, 33

34. |-Z (R2, C, M2, Q, .)                                               | (S-t), 1, 2', 3, ir=., 5', 6, 7', 8, 9'
*
```

```
CASE sfree:                                                             ** Change Summary:** New Case -- not trivial
~~~~~~~~~~~~

(p1) m = min(Dom(M))
(p2) M = (M2, m -> v1, ..., m+n-1->vn)
----------------------------------------------------------------------------------- (sfree)
(R,C,M,Q,salloc n) -->_0 (R++[spg -> R(spg)+n][spb -> R(spb)+n], C, M2, Q, .)

0. |-Z (R,C,M,Q,salloc n)                                               | Given
1. Dom(P) = Dom(C) union Dom(M_m)                                       | Inversion of (S-t), 0
2. M = M_s #Dom(C) M_m                                                  | Inversion of (S-t), 0
3. P |- C                                                               | Inversion of (S-t), 0
4. Forall c=/=Z. C(R(pc_c)) = sfree n                                   | Inversion of (S-t), 0
5. Forall c=/=Z. P(R(pc_c)) = (D;G;seq(E_d,E_s);E_m;s)-->void           | Inversion of (S-t), 0
6. Exists S. . |- S : D                                                 | Inversion of (S-t), 0
7. P |-Z M_s : S(Es)                                                    | Inversion of (S-t), 0
8. P |-Z (M_m,Q) : (S(E_m), S(seq(E_d,E_m)))                            | Inversion of (S-t), 0
9. P |-Z R : S(G)                                                       | Inversion of (S-t), 0

Let R2 = R++[spg -> R(spg)+n][spb -> R(spb)+n]
Let G2 = G++[spg -> <G,sptr,Eg+n>][spb -> <B,sptr,Eb+n>]

10. P;(D;G;seq(E_d,E_s);E_m;s) |- sfree n ==> RT2                       | Inversion of (C-t), 3, 4
11. RT2 = (D;G2;seq(E_d,E_s);Em;s2)                                     | Inspection of (op2r-t), def of G2
12. Forall c=/=Z. P(R(pc_c)+1) = RT2 --> void                           | Inversion of (C-t), 3, 4, 11

5'. Forall c=/=Z. P(R2_val(pc_c)) = (D;G2;seq(E_d,E_s);Em2;s2) --> void | def of ++, def of R2, 11, 12

13. P;(D;G;seq(E_d,E_s);E_m;Et:us)|- sfree n                            | 10, 11
    ==>(D;G2;seq(E_d,E_s);E_m;s2)
14. P;(.;S(G);S(seq(E_d,E_s));S(E_m);S(s))|- sfree n                    | substitution, 6, 13
      ==> (.;S(G2);S(seq(E_d,E_s));S(E_m);S(s2))

15. S(G)(spg) = <G,sptr,Eg>                                             | Inversion of (salloc-t), 14
16. S(G)(spb) = <B,sptr,Eb>
17. . |- Eg = Eb
18. . |- Eg = Et
19. s = Et : t  ::  ...  ::  Ef : us
20. . |- Ef = Eg + n
```

```
21. Dom(M) = Dom(M_s) union Dom(M_m)                                    | Inversion of #-def, 2
22. Dom(M_s) intersect Dom(M_m) = empty
23. All ls in Dom(M_s). all Lc in Dom(C). all Lm in Dom(M_m).
    ls < lc < lm

24. . |- m = Et                                                         | Inversion of (s-cons-t), 7, (p1), 2
25. . |- m+n = Ef                                                       | Exp Eq Transitivity, 18, 20, 21
26. m = min(Dom(M_s))                                                   | 21, 23, (p1)
27. m through m+n in Dom(M_s)                                           | Repeated inversion of (s-cons-t), 7, 25, 19
28. M_s = M_s2, m -> v1, ..., m+n-1->vn                                 | 27, (p2)
29. m through m+n not in Dom(M_m)                                       | 22, 27

30. Dom(M2) = Dom(M_s2) union Dom(M_m)                                  | 21, 28, 29, (p2)
31. Dom(M_s2) intersect Dom(M_m) = empty                                | 22, 28
33. All ls in Dom(M_s2). all Lc in Dom(C). all Lm in Dom(M_m).          | 23, 28
    ls < lc < lm
2'. M2 = M_s2 #Dom(C) M_m                                               | (#-def), 30, 21, 23


7'. P |-Z M_s2 : S(Ef:us)                                               | n Inversions of (s-cons-t), 7, 19, 27

35. P;. |-Z R(spg) + n : <G,sptr,Eg + n>                                | 15, case on Z: deconstruct and reconstruct using
                                                                          whichever (val-t) rule applies
36. P;. |-Z R(spb) + n : <B,sptr,Eb + n>                                | 16, case on Z: deconstruct and reconstruct using
                                                                          whichever (val-t) rule applies
37. P; . |-Z R2(spg) : S(G2)(spg)                                       | 35, def of R2, G2
38. P; . |-Z R2(spb) : S(G2)(spb)                                       | 36, def of R2, G2
39. P |-Z R++ : S(G++)                                                  | 9, def of ++
9'. P |-Z R2 : S2(G2)                                                   | (R-t), def of R2, def of G2, 37, 38, 39


40. |-Z (R2, C, M2, Q, .)                                               | (S-t), 1, 2', 3, ir=., 5', 6, 7', 8, 9'
*
```

```
CASE ld_G-queue:                                                        ** Change Summary:** complete -- fixed bug in original
~~~~~~~~~~~~~~~~                                                                             version, redid init flags

(p1) find(Q,R(r_s)) = (R(r_s),n)
-----------------------------------------------------------------(ld_G-queue)
(R,C,M,Q, ld_G r_d, r_s )  -->_0  (R++[r_d -> G n] ,C,M,Q,.)


0. |-Z (R,C,M,Q,ld_G r_d, r_s)                                          | Given
1. Dom(P) = Dom(C) union Dom(M_m)                                       | Inversion of (S-t), 0
2. M = M_s #Dom(C) M_m                                                  | Inversion of (S-t), 0
3. P |- C                                                               | Inversion of (S-t), 0
4. Forall c=/=Z. C(R(pc_c)) = ld_G r_d, r_s                             | Inversion of (S-t), 0
5. Forall c=/=Z. P(R(pc_c)) = (D;G;seq(E_d,E_s);E_m;s)-->void           | Inversion of (S-t), 0
6. Exists S. . |- S : D                                                 | Inversion of (S-t), 0
7. P |-Z M_s : S(s)                                                     | Inversion of (S-t), 0
8. P |-Z (M_m,Q) : (S(E_m), S(seq(E_d,E_m)))                            | Inversion of (S-t), 0
9. P |-Z R : S(G)                                                       | Inversion of (S-t), 0

Let R2 = R++[r_d --> G n]
Let G2 = G++[r_d --> <G,b,E> ]

10. P;(D;G;seq(E_d,E_s);E_m;s) |- ld_G r_d, r_s ==> RT2                 | Inversion of (C-t), 3, 4
11. RT2 = (D;G2;seq(E_d,E_s);E_m;s)                                     | Inspection of (ld_G-t), def of G2
12. Forall c=/=Z. P(R(pc_c)+1) = RT2 -> void                            | Inversion of (C-t), 3, 4, 11
5'. Forall c=/=Z. P(R2_val(pc_c)) = (D;G2;seq(E_d,E_s);E_m;s) --> void  | def of ++, def of R2, 12

13. P;(.;S(G);S(seq(E_d,E_s));S(E_m);S(s))|- ld_G r_d, r_s              | 10, 11, substitution, 6
      ==>(.;S(G2);S(seq(E_d,E_s));S(E_m);S(s))

14. . |- S(G)(r_s) <= <G,b refh,E_s'>                                   | Inversion of (ld_G-t), 13
15. E = sel (sequpd S(E_m) S(seq(E_d,E_s))) E_s'                        | Inversion of (ld_G-t), 13


16. P; . |-Z R(r_s) : <G,b refh,E_s'>                                   | Inversion of (reg-file-t), 9, (val-subtp-t), 14



SUBCASE a: Z = G

17a. . |- S(E_m) : kmem                                                 | Exp Evaluation Lemma, Inversion on (heap-t), 8
18a. . |- S(seq(E_d,E_s)) : seq(kint,kint)                              | Inversion of (heap-t), 8, Inversion (Q-t) and (Q-zap-t)
```

```
19a. . | E_s' : kint                                                    | Canonical Forms, 1, 8, 3, 16
20a. . |- E : kint                                                      | By applying sequences of (E-upd-t) and (E-sel-t),
                                                                            15, 17a, 18a, 19a
21a. P;.|-Z n : <G,b,E>                                                 | (val-zap-t), assumption, 20a


SUBCASE b: Z =/= G

22b. P(R(r_s)) = b reff and f <= h                                      | Canonical Forms, 1, 8, 3, 16
23b. . |- R(R_s) in Dom(M)
24b. . |- E_s' = R(r_s)
25b. f<=h ==> Z=/=G ==> all (R(r_s),v) in Q. P |- v : b
26b. P |- n : b                                                         | 25b, assumption, 22b, (p1)

27b. . |- E = n                                                         | Find Lemma, (p1), (Inversion of (heap-t), 8), 15, 24b

28b. P;. |-Z n : <G,b,E>                                                | (val-t), 26b, 27b

MERGE:
22. P;.|-Z R2(r_d) : S(G2)(r_d)                                         | 21a/28b, def of R2, def of G2, def of ++
9'. P |-Z R2 : S(G2)                                                    | (reg-file-t), 9, def of R2, def of G2, def of ++, 22

23. |-Z (R2,C,M,Q,.)                                                    | (S-t), 1, 2, 3, ir=., 5', 6, 7, 8, 9'
* complete




CASE ld_G-mem:                                                       ** Change Summary:** complete -- add init flags
~~~~~~~~~~~~~~

(p1) find(Q,R(r_s)) = ()
(p2) R(r_s) in Dom(M)
(s1) R2 = R++[r_d -> M(R(r_s)) ]
-----------------------------------------------(ld_G-mem)
(R,C,M,Q, ld_G rd, rs )  -->_0  (R2,C,M,Q,.)

0. |-Z (R,C,M,Q,ld_G r_d, r_s)                                          | Given
1. Dom(P) = Dom(C) union Dom(M_m)                                       | Inversion of (S-t), 0
2. M = M_s #Dom(C) M_m                                                  | Inversion of (S-t), 0
3. P |- C                                                               | Inversion of (S-t), 0
4. Forall c=/=Z. C(R(pc_c)) = ld_G r_d, r_s                             | Inversion of (S-t), 0
5. Forall c=/=Z. P(R(pc_c)) = (D;G;seq(E_d,E_s);E_m;s)-->void           | Inversion of (S-t), 0
6. Exists S. . |- S : D                                                 | Inversion of (S-t), 0
7. P |-Z M_s : S(s)                                                     | Inversion of (S-t), 0
8. P |-Z (M_m,Q) : (S(E_m), S(seq(E_d,E_m)))                            | Inversion of (S-t), 0
9. P |-Z R : S(G)                                                       | Inversion of (S-t), 0

Let R2 = R++[r_d --> M(R(r_s)) ]
Let G2 = G++[r_d --> <G,b,E> ]

10. P;(D;G;seq(E_d,E_s);E_m) |- ld_G r_d, r_s ==> RT2                   | Inversion of (C-t), 3, 4
11. RT2 = (D;G2;seq(E_d,E_s);E_m;s)                                     | Inspection of (ld_G-t), def of G2, 10
12. Forall c=/=Z. P(R(pc_c)+1) = RT2 -> void                            | Inversion of (C-t), 3, 4, 11
5'. Forall c=/=Z. P(R2_val(pc_c)) = (D;G2;seq(E_d,E_s);E_m;s) --> void  | def of ++, def of R2, 12

13. P;(.;S(G);S(seq(E_d,E_s));S(E_m);S(s))|- ld_G r_d, r_s              | 10, 11, substitution, 6
      ==>(.;S(G2);S(seq(E_d,E_s));S(E_m);S(s))

14. . |- S(G)(r_s) <= <G,b refi,E_s'>                                   | Inversion of (ld_G-t), 13
15. E = sel (sequpd S(E_m) S(seq(E_d,E_s))) E_s'                        | Inversion of (ld_G-t), 13

16. P; . |-Z R(r_s) : <G,b refh,E_s'>                                   | Inversion of (reg-file-t), 9, (val-subtp-t), 14


SUBCASE a: Z = G

17a. . |- S(E_m) : kmem                                                 | Exp Evaluation Lemma, Inversion on (heap-t), 8
18a. . |- S(seq(E_d,E_s)) : seq(kint,kint)                              | Inversion of (heap-t), 8, Inversion (Q-t) and (Q-zap-t)
19a. . | E_s' : kint                                                    | Canonical Forms, 1, 8, 3, 16
20a. . |- E : kint                                                      | By applying sequences of (E-upd-t) and (E-sel-t),
                                                                            15, 17a, 18a, 19a
21a. P;.|-Z n : <G,b,E>                                                 | (val-zap-t), assumption, 20a


SUBCASE b: Z =/= G

22b. P(R(r_s)) = b reff and f <= h                                      | Canonical Forms, 1, 8, 3, 16
23b. . |- R(R_s) in Dom(M)
24b. . |- E_s' = R(r_s)
25b. f=h ==> Z=/=G ==> exists (n,v) in Q
26b. f=1 ==> P |- M(R(r_s)) : b
```

```
27b. f = h or f = 1                                                        | 22b, def of f <= f
subsubcase on 27b
subsubcase b1: f = h
28b1. exists (n,v) in Q                                                    | 25b, subcase assumption, subsubcase assumption
29b1. contradiction, subcase doesn't apply                                   | 28b1, (p1)
subsubcase b2: f = 1
28b2. P |- M(R(r_s)) : b                                                   | 26b, subsubcase assumption
Merge:
29b. P |- M(R(r_s)) : b                                                    | 29b1 / 28b2

30b. P;. |-Z M(R(r_s)) : <G,b,E>                                           | (val-t), 24b, 29b


MERGE:
32. P;.|-Z R2(r_d) : S(G2)(r_d)                                            | 21a/30b, def of R2, def of G2
9'. P |-Z R2 : S(G2)                                                       | (reg-file-t), 9, def of R2, def of G2, def of ++, 32

23. |-Z (R2,C,M,Q,.)                                                       | (S-t), 1, 2, 3, ir=., 5', 6, 7, 8, 9'
*
```

```
CASE ld_G-rand:                                                          ** Change Summary:** simple
~~~~~~~~~~~~~~~

find(Q,R(r_s)) = ()
R(r_s) not in Dom(M)
R2 = R++[r_d -> n ]
-------------------------------------------------(ld_G-rand)
(R,C,M,Q, ld_G r_d, r_s )  -->_0  (R2,C,M,Q,.)

0. |-Z (R,C,M,Q,ld_G r_d, r_s)                                             | Given
1. Dom(P) = Dom(C) union Dom(M_m)                                          | Inversion of (S-t), 0
2. M = M_s #Dom(C) M_m                                                     | Inversion of (S-t), 0
3. P |- C                                                                  | Inversion of (S-t), 0
4. Forall c=/=Z. C(R(pc_c)) = ld_G r_d, r_s                                | Inversion of (S-t), 0
5. Forall c=/=Z. P(R(pc_c)) = (D;G;seq(E_d,E_s);E_m;s)-->void              | Inversion of (S-t), 0
6. Exists S. . |- S : D                                                    | Inversion of (S-t), 0
7. P |-Z M_s : S(s)                                                        | Inversion of (S-t), 0
8. P |-Z (M_m,Q) : (S(E_m), S(seq(E_d,E_m)))                               | Inversion of (S-t), 0
9. P |-Z R : S(G)                                                          | Inversion of (S-t), 0

Let R2 = R++[r_d --> n]
Let G2 = G++[r_d --> < G,int,E_n>]                                         | where E_n is just n

10. P;(D;G;seq(E_d,E_s);E_m;s) |- ld_G r_d, r_s ==> RT2                    | Inversion of (C-t), 3, 4
11. RT2 = (D;G2;seq(E_d,E_s);E_m;s)                                        | Inspection of (ld_G-t), def of G2, 10
12. Forall c=/=Z. P(R(pc_c)+1) = RT2 --> void                              | Inversion of (C-t), 3, 4, 11
5'. Forall c=/=Z. P(R2_val(pc_c)) = (D;G2;seq(E_d,E_s);E_m;s) --> void     | def of ++, def of R2, 11, 12

13. P |- n : int                                                           | (int-t)
14. . |- E_n = n                                                           | def of G2
15. P;. |-Z n : <G,int,E_n>                                                | (val-t), 13, 14
16. P;. |-Z R2(r_d) : S(G2)(r_d)                                           | 15, def of R2, def of G2, Subst Closed Exp Lemma
9'. P |-Z R2 : S(G2)                                                       | (reg-file-t), 7, def of R2, def of G2, def of ++, 15, 16

17. |-Z (R2,C,M,Q,.)                                                       | (S-t), 1, 2, 3, ir=., 5', 6, 7, 8, 9'
*
```

```
CASE ld_G-fail:                                                            ** Change Summary:** none
~~~~~~~~~~~~~~~

find(Q,R(r_s)) = ()
R(r_s) not in Dom(M)
-------------------------------------(ld_G-fail)
(R,C,M,Q, ld_G r_d, r_s)  -->_0  fault


does not apply (fails second assumption)
*
```

```
CASE ld_B-mem:                                                              ** Change Summary:** spelled out instead of being
```

```
~~~~~~~~~~~~~~                                                                                                 similar to ld_G-mem, not too tricky

R(r_s) in Dom(M)
R' = R1++[ rd -> B M(R(r_s)) ]
-------------------------------------------------(ld_B-mem)
(R,C,M,Q, ld_B rd, rs )  -->_0  (R',C,M,Q,.)


0. |-Z (R,C,M,Q,ld_B r_d, r_s)                                          | Given
1. Dom(P) = Dom(C) union Dom(M_m)                                       | Inversion of (S-t), 0
2. M = M_s #Dom(C) M_m                                                  | Inversion of (S-t), 0
3. P |- C                                                               | Inversion of (S-t), 0
4. Forall c=/=Z. C(R(pc_c)) = ld_B r_d, r_s                             | Inversion of (S-t), 0
5. Forall c=/=Z. P(R(pc_c)) = (D;G;seq(E_d,E_s);E_m;s)-->void           | Inversion of (S-t), 0
6. Exists S. . |- S : D                                                 | Inversion of (S-t), 0
7. P |-Z M_s : S(s)                                                     | Inversion of (S-t), 0
8. P |-Z (M_m,Q) : (S(E_m), S(seq(E_d,E_m)))                            | Inversion of (S-t), 0
9. P |-Z R : S(G)                                                       | Inversion of (S-t), 0

Let R2 = R++[r_d --> M(R(r_s)) ]
Let G2 = G++[r_d --> <B,b,E> ]

10. P;(D;G;seq(E_d,E_s);E_m) |- ld_B r_d, r_s ==> RT2                   | Inversion of (C-t), 3, 4
11. RT2 = (D;G2;seq(E_d,E_s);E_m;s)                                     | Inspection of (ld_B-t), def of G2, 10
12. Forall c=/=Z. P(R(pc_c)+1) = RT2 -> void                            | Inversion of (C-t), 3, 4, 11
5'. Forall c=/=Z. P(R2_val(pc_c)) = (D;G2;seq(E_d,E_s);E_m;s) --> void  | def of ++, def of R2, 12

13. P;(.;S(G);S(seq(E_d,E_s));S(E_m);S(s))|- ld_G r_d, r_s              | 10, 11, substitution, 6
      ==>(.;S(G2);S(seq(E_d,E_s));S(E_m);S(s))

14. S(G)(r_s) = <G,b ref1,E_s'>                                         | Inversion of (ld_B-t), 13
15. E = sel S(E_m) E_s'                                                 | Inversion of (ld_B-t), 13

16. P; . |-Z R(r_s) : <B,b ref1,E_s'>                                   | Inversion of (reg-file-t), 9, (val-subtp-t), 14


SUBCASE a: Z = B

17a. . |- S(E_m) : kmem                                                 | Exp Evaluation Lemma, Inversion on (heap-t), 8
18a. . | E_s' : kint                                                    | Canonical Forms, 1, 8, 3, 16
19a. . |- E : kint                                                      | (E-sel-t), 17a, 18a
20a. P;.|-Z n : <B,b,E>                                                 | (val-zap-t), assumption, 19a


SUBCASE b: Z =/= B

22b. P(R(r_s)) = b ref1                                                 | Canonical Forms, 1, 8, 3, 16, def of f <= f
23b. . |- R(R_s) in Dom(M)
24b. . |- E_s' = R(r_s)
25b. P |- M(R(r_s)) : b
26b. P;. |-Z M(R(r_s)) : <G,b,E>                                        | (val-t), 24b, 25b


MERGE:
32. P;.|-Z R2(r_d) : S(G2)(r_d)                                         | 20a/26b, def of R2, def of G2
9'. P |-Z R2 : S(G2)                                                    | (reg-file-t), 9, def of R2, def of G2, def of ++, 32

23. |-Z (R2,C,M,Q,.)                                                    | (S-t), 1, 2, 3, ir=., 5', 6, 7, 8, 9'
*






CASE ld_B-rand:                                                         ** Change Summary:** none
~~~~~~~~~~~~~~~

R(r_s) not in Dom(M)
R' = R++[ r_d -> B n ]
---------------------------------------------------(ld_B-rand)
(R,C,M,Q, ld_B r_d, r_s )  -->_0  (R',C,M,Q,.)

Similar to ld_G-rand.
*






CASE ld_B-fail:                                                        ** Change Summary:** none
```

```
~~~~~~~~~~~~~~~

Rval(r_s) not in Dom(M)
-------------------------------------------(ld_B-fail)
(R,C,M,Q, ld_B r_d, r_s )  -->_0  fault

does not apply (fails second assumption)
*
```

```
CASE sld_c:                                                           ** Change Summary:** New Case -- requires stack
~~~~~~~~~~~                                                                                     update lemma, then easy

(p1) R(spc) + n in Dom(M)
-----------------------------------------------------------------(sld_c)
(R,C,M,Q, sld_c r_d n )  -->_0  (R++[r_d -> M(R(spc)+n)],C,M,Q,.)


0. |-Z (R,C,M,Q,sld_c r_d n)                                          | Given
1. Dom(P) = Dom(C) union Dom(M_m)                                     | Inversion of (S-t), 0
2. M = M_s #Dom(C) M_m                                                | Inversion of (S-t), 0
3. P |- C                                                             | Inversion of (S-t), 0
4. Forall c=/=Z. C(R(pc_c)) = sfree n                                 | Inversion of (S-t), 0
5. Forall c=/=Z. P(R(pc_c)) = (D;G;seq(E_d,E_s);E_m;s)-->void         | Inversion of (S-t), 0
6. Exists S. . |- S : D                                               | Inversion of (S-t), 0
7. P |-Z M_s : S(s)                                                   | Inversion of (S-t), 0
8. P |-Z (M_m,Q) : (S(E_m), S(seq(E_d,E_m)))                          | Inversion of (S-t), 0
9. P |-Z R : S(G)                                                     | Inversion of (S-t), 0

Let R2 = R++[r_d -> R(spc)+n]
Let G2 = G++[spc -> <G,b,E>]

10. P;(D;G;seq(E_d,E_s);E_m;s) |- sld_c r_d n ==> RT2                 | Inversion of (C-t), 3, 4
11. RT2 = (D;G2;seq(E_d,E_s);Em;s)                                    | Inspection of (op2r-t), def of G2
12. Forall c=/=Z. P(R(pc_c)+1) = RT2 --> void                         | Inversion of (C-t), 3, 4, 11

5'. Forall c=/=Z. P(R2_val(pc_c)) = (D;G2;seq(E_d,E_s);E_m;s) --> void  | def of ++, def of R2, 11, 12

13. P;(D;G;seq(E_d,E_s);E_m;s)|- sld_c r_d n                          | 10, 11
    ==>(D;G2;seq(E_d,E_s);E_m;s)
14. P;(.;S(G);S(seq(E_d,E_s));S(E_m);S(s))|- sld_c r_d n              | substitution, 6, 13
      ==> (.;S(G2);S(seq(E_d,E_s));S(E_m);S(s))

15. S(G)(spc) = <c,sptr,Ec>                                           | Inversion of (sld_c-t), 14
16. . |- Ec + n = En
17. .;s |- En : <c,b,E>

18. . |- Ec = R(spc)                                                  | Canonical Forms, 1, 3, 8, (Inversion of (reg-file-t),9,15)
19. . |- R(spc) + n = En                                              | Exp Eq Trans, 18, 16
20. P;. |-Z M_s(R(spc)+n) : <c,b,E>                                   | Stack Lookup Lemma, 7, 17, 19

9'. P |-Z R2 : S(G2)                                                  | (reg-file-t), 9, 20, def of ++

21. |-Z (R2,C,M,Q,.)                                                  | (S-t), 1, 2, 3, ir=., 5', 6, 7, 8, 9'
*
```

```
CASE sld_c-fail:                                                      ** Change Summary:** New Case -- trivial
~~~~~~~~~~~~~~~~

(p1) R(spc) + n not in Dom(M)
-----------------------------------------------------------------(sld_c-fail)
(R,C,M,Q, sld_c r_d n )  -->_0  fault

does not apply (fails second assumption)
*
```

```
CASE st_G-queue:                                                          ** Change Summary:** complete, have to modify P if
~~~~~~~~~~~~~~~~                                                                                Z=/=G, need to update all known aliases

 Q2 = ( (R(r_d), R(r_s)), Q )
---------------------------------------------------(st_G-queue)
(R,C,M,Q, st_G r_d, r_s )  -->_0  (R++,C,M,Q2,.)



0. |-Z (R,C,M,Q,st_G r_d, r_s)                                            | Given
1. Dom(P) = Dom(C) union Dom(M_m)                                         | Inversion of (S-t), 0
2. M = M_s #Dom(C) M_m                                                    | Inversion of (S-t), 0
3. P |- C                                                                 | Inversion of (S-t), 0
4. Forall c=/=Z. C(R(pc_c)) = st_G rd, rs                                 | Inversion of (S-t), 0
5. Forall c=/=Z. P(R(pc_c)) = (D;G;seq(E_d,E_s);E_m;s)-->void             | Inversion of (S-t), 0
6. Exists S. . |- S : D                                                   | Inversion of (S-t), 0
7. P |-Z M_s : S(s)                                                       | Inversion of (S-t), 0
8. P |-Z (M_m,Q) : (S(E_m), S(seq(E_d,E_m)))                              | Inversion of (S-t), 0
9. P |-Z R : S(G)                                                         | Inversion of (S-t), 0


Let p = increment_f(f)
let G2 = S(G++[r_d -> <G, b refp, E_d'>])               ** FJP needs to be updated to handle larger change to G as in other examples


10. P;(D;G;seq(E_d,E_s);E_m) |- st_G r_d, r_s ==> RT2                     | Inversion of (C-t), 3, 4
11. RT2 = (D;G++;((E_d',E_s'),(seq(E_d,E_s)));E_m;s)                      | Inspection of (st_G-t), def of G2, 11
12. Forall c=/=Z. P(R(pc_c)+1) = RT2 --> void                             | Inversion of (C-t), 3, 4, 11

13. P;(.;S(G);S(seq(E_d,E_s));S(E_m);S(s))|- st_G r_d, r_s                | 10, 11, substitution, 6
        ==>(.;S(G2);((E_d',E_s'),S(seq(E_d,E_s)));S(E_m);S(s))

14. S(G)(r_d) = <G,b reff,E_d'>                                           | Inversion of (st_G-t), 13
15. S(G)(r_s) = <G,b,E_s'>


16. [[S(E_m)]] = M_m                                                      | Inversion of (heap-t), 8
17. P |-Z Q : S(seq(E_d,E_s))
18. all l in Dom(M_m). exists f. P;M_m;Q |-Z l : b reff


SUBCASE a: Z = G  [ don't need to update P ]

5a'. Forall c=/=Z. P(R2_val(pc_c))                                        | def of ++, def of R2, 11, 12
    = (D;G2;((E_d',E_s'),(seq(E_d,E_s)));E_m;s) --> void

20a. . |- E_d' : kint  . |- E_s' : kint                                   | Canonical Forms,1,8,3, (Inversion of (reg-file-t), 9, 14, 15)
21a. P |-G ((R(r_d),R(r_s)), Q) : ((E_d',E_s'),S(seq(E_d,E_s)))           | (Q-zap-t), 17, 20a
22a. P |-G ((R(r_d),R(r_s)), Q) : (S((E_d',E_s'),seq(E_d,E_s)))           | Subst Closed Exp Lemma, 20a, 21a
23a. all l in Dom(M_m). exists f. P;M_m;Q2 |-G l : b reff                 | Inversion/reconstruction of (init-t), (uninit-t),
                                                                             (hinit-t) when Z=G
8a'. P |-G (M_m,Q2) : (S(E_m), S(E_d',E_s'),seq(E_d,E_m)))                | (heap-t), 15, 22a, 23a

24a. P;. |-G R(r_d) : <G, b refp, E_d'>                                   | (val-zap-t), 20a
9a'. P |- R++ : S(G2)                                                     | (reg-file-t), 9, def of ++, def of G2, 24a

25a. |-Z (R++,C,M,Q2,.)                                                   | (S-t), 1, 2, 3, ir=., 5a', 6, 7, 8a', 9a'



SUBCASE b: Z =/= G [  may need to update P to note initialization ]

20b. . |- E_d' = R(r_d) and P(R(r_d)) = b reff'  and f' <= f              | Canonical Forms, 1, 8, 3, (Inversion of (reg-file-t),9,14)
21b. . |- E_s' = R(r_s) and P(R(r_s)) = b                                 | Canonical Forms, 1, 8, 3, (Inversion of (reg-file-t),9,15)

22b. Q subset Q2                                                          | def of Q2
23b. all l  in Dom(M_m). exists f. P;M_m;Q2 |-Z l : b reff                | Inversion/reconstruction of (init-t), (uninit-t),
                                                                             (hinit-t), 22b
subsubcase on value of f'



subsubcase b1: f' = 0  [ do need to update P because change init flag from 0 to 1/2 ]

Let P2 = P[R(r_d) -> b refp]

24b1. f = 0                                                               | 20b, subsubcase assumption, def of f <= f
25b2. p = h                                                               | def of p, def of increment_f

26b1. p <= f                                                              | def of f <= f, 24b1, 25b1
27b1. b refp <= b reff                                                    | (subtp-b-ref), 26b1
```

```
28b1. all l not R(r_d) in Dom(M_m). Exists f. P2;M_m;Q2 |-Z l : b reff  | Psi Subtyping Lemma,  23b, 27b1

29b1. P2(R(r_d)) = b refh                                               | (addr-heap-t), def of P2,  25b2
30b1. (R(r_d),R(r_s)) in Q2                                             | def of Q2
31b1. P2;M_m;Q2 |-Z R(r_d) : b refh                                     | (hinit-t), 29b1, 30b1

32b1. all l in Dom(M_m). Exists f. P2;M_m;Q2 |-Z l : b reff             | 28b1, 31b1

33b1. P2 |-Z Q : S(seq(E_d,E_s))                                        | Inversion/Reconstruction of (Q-t) and (Q-emp-t),
                                                                           17, def of P2, 25b2
34b1. P2 |-Z Q2 : S((E_d',E_s'),seq(E_d,E_s))                           | (Q-t), 33b1, 20b, 21b, 21b1, Subst Closed Exp

8b1'. P |-Z (M_m,Q2) : ( S(E_m), S((E_d',E_s'),seq(E_d,E_s)) )          | (heap-t), 16, 34b1, 32b1

35b1. R(r_d) in Dom(P)                                                  | 20b
1b1'. Dom(P2) = Dom(C) union Dom(M_m)                                   | 1, 39b

40b1. R(r_d) not in Dom(C)                                              | Inversion of (C-t), 20b
3b1'. P2 |- C                                                           | (C-t), (Inversion of (C-t), 3), 40b1
5b1'. Forall c=/=Z. P2(R2_val(pc_c))                                    | def of ++, def of R2, 11, 12, def of P2, 40b1
    = (D;G2;((E_d',E_s'),(seq(E_d,E_s)));E_m;s) --> void

41b1: P2 |- R(r_d) : b refp                                             | Def of P2, (addr-heap-t)
42b1. P2;. |-Z R(r_d) : <G, b refp, E_d'>                               | (val-t), 41b1, 20b
43b1. all a. P2;. |- R(a) : S(G)(a)                                     | Inversion of (reg-file-t), 9, Psi Subtyping Lemma
9b1'. P2 |- R++ : S(G2)                                                 | (reg-file-t), 43b1, 42b1, def of ++

44b1. |-Z (R++,C,M,Q2,.)                                                | (S-t), 1b1', 2, 3b1', ir=., 5b1', 6, 7, 8b1', 9b1'



subsubcase b2: f' = h or f' = 1     [ do not need to update P - increment_f(f) = f ]

24b2. f <= h                                                            | subcase assumption
25b2. P |-Z Q2 : ((E_d',E_s'),S(seq(E_d,E_s)))                          | (Q-t), def of Q2, 20b, 21b, 24b2
25b2. P |-Z Q2 : S((E_d',E_s'),seq(E_d,E_s))                            | Subst Closed Expr, 25b1, 20b, 21b

8b2'. P |-Z (M_m,Q2) : ( S(E_m), S((E_d',E_s'),seq(E_d,E_s)) )          | (heap-t), 23b, 25b2, 16


5b2'. Forall c=/=Z. P(R2_val(pc_c))                                     | def of ++, def of R2, 11, 12
    = (D;G2;((E_d',E_s'),(seq(E_d,E_s)));E_m;s) --> void


26b2: p = f                                                             | def of increment_f, 24b2
27b2. P;. |-Z R(r_d) : <G, b refp, E_d'>                                | Inversion of (reg-file-t), 9, 14, 26b2
9b2'. P |- R++ : S(G2)                                                  | Inversion/reconstruction of (reg-file-t), 9, def of ++,
                                                                           27b2, def of G2

28b2. |-Z (R++,C,M,Q2,.)                                                | (S-t), 1, 2, 3, ir=., 5b2', 6, 7, 8b2', 9b2'



MERGE:
45. |-Z (R++,C,M,Q2,.)                                                  | 25a / 44b1 / 28b2
*








CASE st_B-mem:                                                          ** Change Summary:** complete - had to modify P
~~~~~~~~~~~~~~

(p1) R(r_d) = n1
(p2) R(r_s) = n1'
------------------------------------------------------------------(st_B-mem)
(R, C, M, ((seq(ns1,ns1'),(n1,n1')), st_B r_d, r_s)
      -->_0^(n1,n1')  (R++, C, M[n1->n1'], seq(ns1,ns1') , .)


0. |-Z (R,C,M,((seq(ns1,ns1'),(n1,n1')),st_G r_d, r_s)                      | Given
1. Dom(P) = Dom(C) union Dom(M_m)                                           | Inversion of (S-t), 0
2. M = M_s #Dom(C) M_m                                                      | Inversion of (S-t), 0
3. P |- C                                                                   | Inversion of (S-t), 0
4. Forall c=/=Z. C(R(pc_c)) = st_G rd, rs                                   | Inversion of (S-t), 0
5. Forall c=/=Z.P(R(pc_c))=(D;G;(seq(E_d,E_s),(E_d',E_s'));E_m;s)-->void    | Inversion of (S-t), 0
6. Exists S. . |- S : D                                                     | Inversion of (S-t), 0
7. P |-Z M_s : S(s)                                                         | Inversion of (S-t), 0
8. P |-Z (M_m,(seq(ns1,ns1'),(n1,n1')))                                     | Inversion of (S-t), 0
```

```
         : (S(E_m),S(seq(E_d,E_m),(E_d',E_s')))
9. P |-Z R : S(G)                                                            | Inversion of (S-t), 0


let P2 = P[n1 -> b ref1]
let G2 = S(G++[r_d -> <G, b ref1, E_d''>]

10. P;(D;G;(seq(E_d,E_s),(E_d',E_s'));E_m) |- st_B r_d, r_s ==> T2          | Inversion of (C-t), 3, 4
11. T2 = (D;G++;seq(E_d,E_s); upd E_m E_d' E_s'; s )                         | Inspection of (st_B-t), def of G2, 10
12. Forall c=/=Z. P(R(pc_c)+1) = T2                                          | Inversion of (C-t), 3, 4,11
13. Forall c=/=Z. P(R2_val(pc_c))                                            | def of ++, def of R2, 11, 12
    = (D;G++;seq(E_d,E_s); upd E_m E_d' E_s';s ) --> void

14. P;(.;S(G);S(seq(E_d,E_s),(E_d',E_s'));S(E_m); S(s))|- st_B r_d, r_s     | 10, 11, substitution, 6
        ==> (.;S(G++);S(seq(E_d,E_s); upd S(E_m) S(E_d') S(E_s'); S(s) )


15. . |- S(G)(r_d) <= <B,b refh,E_d''>                                       | Inversion of (st_B-t), 13
16. S(G)(r_s) = <B,b,E_s''>
17. . |- S(E_d') = E_d''
18. . |- S(E_s') = E_s''


19. [[S(E_m)]] = M_m                                                         | Inversion of (heap-t), 8
20. P |-Z ((seq(ns1,ns1'),(n1,n1')) : S(seq(E_d,E_s),(E_d',E_s'))
21. all l in Dom(M_m). exists f.
    P;M_m;((seq(ns1,ns1'),(n1,n1')) |-Z l : b reff


22. P2(n) <= P(n)                                                            | def of P2, 21a
23. all l not n1 in Dom(M_m). exists f.                                      | Psi Subtyping Lemma, 21, 22
     P2;M_m;((seq(ns1,ns1'),(n1,n1'))  |-Z l : b reff
24. all l not n1 in Dom(M_m). exists f.                                      | Inversion/Reconstruction of (init-t), (hinit-t),
     P2;M_m;(seq(ns1,ns1') |-Z l : b reff                                      (uninit-t), 23



SUBCASE a. Z = B (queue is guaranteed correct)
25a. . |- S(E_d') = n1                                                       | Inversion of (Q-t), assumption, 20
26a. . |- S(E_s') = n1'
27a. P |- n1 : b reff  and  f <= h
28a. P |- n1' : b
29a. n1 in Dom(M_m)                                                          | similar to Cannonical Forms, 1, 2, 21a


SUBCASE b. Z = G (r_s/r_d are guaranteed correct)
25b. P;. |-G R(r_d) : <B,b ref, E_d''>                                       | Inversion on (reg-file-t), 9, 14, (val-subtp-t)
26b. P;. |-G n1 : <B,b ref, E_d'>                                            | 25b, (p1), Exp Eq Transitivity, 16
27b. n1 in Dom(M_m)   and  P |- n1 : b ref  and  . |- E_d' = n1              | Cannonical Forms 1, 3, 8, 26b

28b. P;. |-G R(r_s) : <B,b,E_s''>                                            | Inversion on (reg-file-t), 9, 15
29b. P;. |-G n1' : <B,b,E_s'>                                                | 28b, (p2), Exp Eq Transitivity, 17
30b. P |- n1' : b  and  . |- E_s' = n1                                       | Canonical Forms, 1, 3, 8, 29b



MERGE:
31. n1 in Dom(M_m)                                                           | 29a / 27b
32. P |- n1' : b                                                             | 28a / 30b
33. . |- S(E_d') = n1  and . |- S(E_s') = n1'                                | (25a, 26a) / (27b, 30b)
34. P(n1) =/= P(n1')                                                         | (27a, 28a) / (27b, 30b)


35. P2 |- n1' : b                                                            | 32, def of P2,34
37. P2 |- M_m[n1 -> n1'](n1') : b                                            | 35
38. P2; M_m[n1 -> n1']; seq(ns1,ns1') |- n1 : b ref1                         | (init-t), def of P2, 37

39. all l =/= n1 in Dom(M_m). exists f.                                      | Inversion/Reconstruction of (init-t), (hinit-t),
     P2;M_m[n1 -> n1'];seq(ns1,ns1') |-Z l : b reff                            (uninit-t), 24, 31

40. all l in Dom(M_m). exists f.                                             | 39, 38
     P2;M_m[n1 -> n1'];seq(ns1,ns1') |-Z l : b reff


41. P |-Z seq(ns1,ns1') : S(seq(E_d,E_s))                                    | Inversion of both (Q-t) and (Q-zap-t), 20


42. [[upd S(E_m) S(E_d') S(E_s')]]                                           | def of [[ ]]
    = [[S(E_m)]] [ [[S(E_d')]] -> [[S(E_s')]] ]
43. [[S(E_d')]] = n1  and [[S(E_s')]] = n1'                                  | Inversion of (E-eq), 33
44. [[upd S(E_m) S(E_d') S(E_s')]] = M [ n1 -> n1' ]                         | 42, 43, 19
45. [[S(upd E_m E_d' E_s')]] = M [ n1 -> n1' ]                               | 44
```

```
8'. P |-Z (M_m, seq(ns1,ns1')) : (S(upd E_m E_d' E_s'), S(seq(E_d,E_m)))    | (heap-t), 40, 41, 45

1'. Dom(P) = Dom(C) union Dom(M[n1->n1'])                                   | 1, 31
2'. M = M_s #Dom(C) M_m                                                     | 2, def of #, 31

46. n1 not in Dom(C)                                                        | Inversion of (C-t), Inversion of (heap-t), 31
3'. P2 |- C                                                                 | (C-t), (Inversion of (C-t), 3), 46, Psi Subtyping Lemma
5'. Forall c=/=Z. P2(R2_val(pc_c))                                          | 13, 46, def of P2
    = (D;G++;seq(E_d,E_s); upd E_m E_d' E_s';s ) --> void

47. |-Z (R++, C, M[n1->n1'], seq(ns1,ns1') , .)                             | 1', 2', 3', ir=., 5', 6, 7', 8', 9'
*




CASE st_B-mem-fail:                                                         ** Change Summary:** none
~~~~~~~~~~~~~~~~~~~

Q = (seq(n,n'),(n1,n1'))
R(r_d) =/= n1  or R(rs) =/= n1'
------------------------------------------(st_B-mem-fail)
(R,C,M,Q, ld_B r_d, r_s )  -->_0  fault

does not apply (fails second assumption)
*




CASE sst:                                                                   ** Change Summary:** New Case -- requires Stack
~~~~~~~~~                                                                                        Update Lemma, then easy

(p1) R(spg) = R(spb)
(p2) R(spg) + n in Dom(M)
-----------------------------------------------------------------(sst)
(R,C,M,Q, sst n r_v )  -->_0  (R++,C, M[R(spg)+n -> R(r_v)], Q,.)



0. |-Z (R,C,M,Q,sst n r_v)                                              | Given
1. Dom(P) = Dom(C) union Dom(M_m)                                       | Inversion of (S-t), 0
2. M = M_s #Dom(C) M_m                                                  | Inversion of (S-t), 0
3. P |- C                                                               | Inversion of (S-t), 0
4. Forall c=/=Z. C(R(pc_c)) = sst n r_v                                 | Inversion of (S-t), 0
5. Forall c=/=Z. P(R(pc_c)) = (D;G;seq(E_d,E_s);E_m;s)-->void           | Inversion of (S-t), 0
6. Exists S. . |- S : D                                                 | Inversion of (S-t), 0
7. P |-Z M_s : S(s)                                                     | Inversion of (S-t), 0
8. P |-Z (M_m,Q) : (S(E_m), S(seq(E_d,E_m)))                            | Inversion of (S-t), 0
9. P |-Z R : S(G)                                                       | Inversion of (S-t), 0


10. P;(D;G;seq(E_d,E_s);E_m;s) |- sst n r_v ==> RT2                     | Inversion of (C-t), 3, 4
11. RT2 = (D;G++;seq(E_d,E_s);Em;s2)                                    | Inspection of (op2r-t), def of G2
12. Forall c=/=Z. P(R(pc_c)+1) = RT2 --> void                           | Inversion of (C-t), 3, 4, 11

5'. Forall c=/=Z. P(R2_val(pc_c)) = (D;G++;seq(E_d,E_s);E_m;s2) -->void | def of ++, def of R2, 11, 12

13. P;(D;G;seq(E_d,E_s);E_m;s)|- sst n r_v                              | 10, 11
    ==>(D;G++;seq(E_d,E_s);E_m;s2)
14. P;(.;S(G);S(seq(E_d,E_s));S(E_m);S(s))|- sst n r_v                  | substitution, 6, 13
      ==> (.;S(G++);S(seq(E_d,E_s));S(E_m);S(s2))

15. S(G)(spg) = <c,sptr,Eg>                                             | Inversion of (sld_c-t), 14
16. S(G)(spb) = <c,sptr,Eb>
17. . |- Eg = Eb
18. . |- Eg + n = En
19. S(G)(r_v) = <c,b,Ev>
20. . |- S(s)[En -> <c,b,Ev>] = s'


21. . |- R(spg) = Eg  or  . |- R(spb) = Eb                              | Canonical Forms, 1, 3, 8, 15 & 16, case on Z
22. . |- R(spg) = Eg  or  . |- R(spb) = Eg                              | Exp Eq Trans, 21, 17
23. . |- R(spg) = Eg                                                    | (p1), 22
24. . |- R(spg) + n = En                                                | exp Eq Trans Lemma, 21, 23

7'. P |-Z M_s[ R(spg)+n -> R(r_v)] : s'                                 | Stack Update Lemma, 7, 20, 24, (Inversion of (reg-file-t),19)

25. R(spg)+n in Dom(M_s)                                                | Valid Stack Loc Lemma, 7, 20, 24
```

```
2'. M[R(spg)+n -> R(r_v)] = M_s[R(spg)+n -> R(r_v)] #Dom(C) M_m         | 2, def of #

9'. P |- R++ : S(G++)                                                    | 9, def of ++


26. |-Z (R++,C, M[R(spg)+n -> R(spg)+n], Q,.)                            | (S-t), 1, 2', 3, ir=., 5', 6, 7', 8, 9'
*
```

```
CASE sst-fail:                                                              ** Change Summary:** New Case -- trivial
~~~~~~~~~~~~~~~

(p1) R(spg) =/= R(spb)  or   R(spg) + n not in Dom(M)
-----------------------------------------------------------------(sst)
(R,C,M,Q, sst n r_v )  -->_0  fault

does not apply (fails second assumption)
*
```

```
CASE bz_G-untaken:                                                          ** Change Summary:**  trivial
~~~~~~~~~~~~~~~~~~

(p1) R(d) = 0      (p2) R(r_z) =/= 0
-----------------------------------------------------(bz-untaken)
(R,C,M,Q, bz_G r_z, r_d )  -->_0  (R++,C,M,Q,.)

0. |-Z (R,C,M,Q,bz_G r_z, r_d)                                           | Given
1. Dom(P) = Dom(C) union Dom(M_m)                                        | Inversion of (S-t), 0
2. M = M_s #Dom(C) M_m                                                   | Inversion of (S-t), 0
3. P |- C                                                                | Inversion of (S-t), 0
4. Forall c=/=Z. C(R(pc_c)) = bz_G r_z, r_d                              | Inversion of (S-t), 0
5. Forall c=/=Z. P(R(pc_c)) = (D;G;seq(E_d,E_s);E_m;s)-->void            | Inversion of (S-t), 0
6. Exists S. . |- S : D                                                  | Inversion of (S-t), 0
7. P |-Z M_s : S(s)                                                      | Inversion of (S-t), 0
8. P |-Z (M_m,Q) : (S(E_m), S(seq(E_d,E_m)))                             | Inversion of (S-t), 0
9. P |-Z R : S(G)                                                        | Inversion of (S-t), 0

Let R2 = R++
Let T' = (D',G',seq(E_d',E_s'),E_m';s')
Let G2 = G++[ d -> (E_z = 0 ==> <G,T'->void,E_d'>) ]

10. P;(D;G;seq(E_d,E_s);E_m) |- bz_G r_z, r_d ==> RT2                    | Inversion of (C-t), 3, 4
11. RT2 = (D;G2;(seq(E_d,E_s));E_m;s)                                    | Inspection of (bz_G-t), def of G2, 10
12. Forall c=/=Z. P(R(pc_c)+1) = RT2                                     | Inversion of (C-t), 3, 4, 11
5'. Forall c=/=Z. P(R2_val(pc_c)) = (D;G2;(seq(E_d,E_s));E_m;s)-->void   | def of ++, def of R2, 11, 12

13. P;(.;S(G);S(seq(E_d,E_s));S(E_m))|- bz_G r_z, r_d                    | 10, 11, substitution, 6
        ==>(.;S(G2);S(seq(E_d,E_s));S(E_m);S(s))
14. S(G)(d)= <G,int,0>                                                   | Inversion of (bz_G-t), 13
15. S(G)(r_z)= <G,int,E_z>                                               | Inversion of (bz_G-t), 13
16. S(G)(r_d)= <G,T'->void,E_d'>                                         | Inversion of (bz_G-t), 13
17. G'(r_d)= <G,T'->void,E_d'>                                           | Inversion of (bz_G-t), 13

SUBCASE a: Z = G
18a. . |- E_z : kint                                                     | Inversion of (reg-file-t), 9, 15, Int Kinding Lemma
19a. . |- E_d' : kint                                                    | Inversion of (reg-file-t), 9, 16, Int Kinding Lemma
20a. P;. |-Z R(d) : (E_z = 0 ==> <G,T'->void,E_d'>)                      | (val-zap-cond), 18a, 19a, assumption, 14

SUBCASE b: Z =/= G
18b. P |- R(r_z) : <G, int, E_z>                                         | Inversion of (reg-file-t), 9
19b. . |- E_z = R(r_z)                                                   | Inversion of (val-t), assumption, 18b
20b. . |- E_z =/= 0                                                      | 19b, (p2), transitivity
21b. P;. |- R(d) : (E_z = 0 ==> <G,T'->void,E_d'>)                       | (cond-t-n0), (p1), 20b

MERGE:
22. P;. |- R(d) : (E_z = 0 ==> <G,T'->void,E_d'>)                        | 20a/21b
23. P |-Z R++ : S(G++)                                                   | 9, def of ++
9'. P |-Z R2 : S(G2)                                                     | (reg-file-t), 23, 22, def of R2, def of G2

24. |-Z (R++,C,M,Q,.)                                                    | (S-t), 1, 2, 3, ir=., 5', 6, 7, 8, 9'
*
```

```
CASE bz_B-untaken:                                                         ** Change Summary:**  trivial
~~~~~~~~~~~~~~~~~~

(p1) R(d) = 0      (p2) R(r_z) = 0
-----------------------------------------------------(bz-untaken)
(R,C,M,Q, bz_B r_z, r_d )  -->_0  (R++,C,M,Q,.)

0. |-Z (R,C,M,Q,bz_B r_z, r_d)                                          | Given
1. Dom(P) = Dom(C) union Dom(M_m)                                       | Inversion of (S-t), 0
2. M = M_s #Dom(C) M_m                                                  | Inversion of (S-t), 0
3. P |- C                                                               | Inversion of (S-t), 0
4. Forall c=/=Z. C(R(pc_c)) = bz_B r_z, r_d                             | Inversion of (S-t), 0
5. Forall c=/=Z. P(R(pc_c)) = (D;G;seq(E_d,E_s);E_m;s)-->void           | Inversion of (S-t), 0
6. Exists S. . |- S : D                                                 | Inversion of (S-t), 0
7. P |-Z M_s : S(s)                                                     | Inversion of (S-t), 0
8. P |-Z (M_m,Q) : (S(E_m), S(seq(E_d,E_m)))                            | Inversion of (S-t), 0
9. P |-Z R : S(G)                                                       | Inversion of (S-t), 0

10. P;(D;G;seq(E_d,E_s);E_m;s) |- bz_G r_z, r_d ==> RT2                 | Inversion of (C-t), 3, 4
11. RT2 = (D;G++;(seq(E_d,E_s));E_m;s)                                  | Inspection of (bz_B-t), def of G2, 10
12. Forall c=/=Z. P(R(pc_c)+1) = RT2                                    | Inversion of (C-t), 3, 4, 11
5'. Forall c=/=Z. P(R2_val(pc_c)) = (D;G++;(seq(E_d,E_s));E_m;s)-->void | def of ++, def of R2, 11, 12

9'. P |-Z R++ : S(G++)                                                  | (reg-file-t), 9, def of ++

13. |-Z (R++,C,M,Q,.)                                                   | (S-t), 1, 2, 3, ir=., 5', 6, 7, 8, 9'
*





CASE bz-untaken-fail:                                                   ** Change Summary:**  none
~~~~~~~~~~~~~~~~~~~~~

R(rz) =/= 0    R(d) =/= 0
-------------------------------------------(bz-untaken-fail)
(R,C,M,Q, bz_c rz, rd )  -->_0  fault

does not apply (fails second assumption)
*





CASE bz_G-taken:                                                         ** Change Summary:**  trivial
~~~~~~~~~~~~~~~~

R(d) = 0    R(r_z) = 0    R2 = R++[d -> R(r_d)]
-------------------------------------------------------------(bz_G-taken)
(R,C,M,Q, bz_G r_z, r_d )  -->_0  (R2,C,M,Q,.)


0. |-Z (R,C,M,Q,bz_G r_z, r_d)                                          | Given
1. Dom(P) = Dom(C) union Dom(M_m)                                       | Inversion of (S-t), 0
2. M = M_s #Dom(C) M_m                                                  | Inversion of (S-t), 0
3. P |- C                                                               | Inversion of (S-t), 0
4. Forall c=/=Z. C(R(pc_c)) = bz_G r_z, r_d                             | Inversion of (S-t), 0
5. Forall c=/=Z. P(R(pc_c)) = (D;G;seq(E_d,E_s);E_m;s)-->void           | Inversion of (S-t), 0
6. Exists S. . |- S : D                                                 | Inversion of (S-t), 0
7. P |-Z M_s : S(s)                                                     | Inversion of (S-t), 0
8. P |-Z (M_m,Q) : (S(E_m), S(seq(E_d,E_m)))                            | Inversion of (S-t), 0
9. P |-Z R : S(G)                                                       | Inversion of (S-t), 0

Let R2 = R++[d -> R(r_d)]
Let T' = (D',G',seq(E_d',E_s'),E_m';s')
Let G2 = G++[ d -> (E_z = 0 ==> <G,T'->void,E_d'>) ]

10. P;(D;G;seq(E_d,E_s);E_m) |- bz_G r_z, r_d ==> RT2                   | Inversion of (C-t), 3, 4
11. RT2 = (D;G2;(seq(E_d,E_s));E_m;s)                                   | Inspection of (bz_G-t), def of G2, 10
12. Forall c=/=Z. P(R(pc_c)+1) = RT2                                    | Inversion of (C-t), 3, 4
5'. Forall c=/=Z. P(R2_val(pc_c)) = (D;G2;(seq(E_d,E_s));E_m;s)-->void  | def of ++, def of R2, 11, 12

13. P;(.;S(G);S(seq(E_d,E_s));S(E_m))|- bz_G r_z, r_d                   | 10, 11, substitution, 6
        ==>(.;S(G2);S(seq(E_d,E_s));S(E_m))
14. S(G)(d)= <G,int,0>                                                  | Inversion of (bz_G-t), 13
15. S(G)(r_z)= <G,int,E_z>                                              | Inversion of (bz_G-t), 13
16. S(G)(r_d)= <G,T'->void,E_d'>                                        | Inversion of (bz_G-t), 13
```

```
17. G'(r_d)= <G,T'->void,E_d'>                                          | Inversion of (bz_G-t), 13

SUBCASE a: Z = G
18a. . |- E_z : kint                                                    | Inversion of (reg-file-t), 9, 15, Int Kinding Lemma
19a. . |- E_d' : kint                                                   | Inversion of (reg-file-t), 9, 16, Int Kinding Lemma
20a. P;. |-Z R(r_d) : (E_z = 0 ==> <G,T'->void,E_d'>)                   | (val-zap-cond), 18a, 19a, assumption, 14

SUBCASE b: Z =/= G
18b. P |- R(r_z) : <G, int, E_z>                                        | Inversion of (reg-file-t), 9, 15
19b. . |- E_z = R(r_z)                                                  | Inversion of (val-t), assumption, 18b
20b. . |- E_z = 0                                                       | 19b, (p2), transitivity

21b. P;. |-Z R(r_d) : <G,T'->void,E_d'>                                 | Inversion of (reg-file-t), 9, 16
22b. R(r_d) =/= 0                                                       | Cannonical Forms 3, 7, 3, 21b

23b. P;. |-Z R(r_d) : (E_z = 0 ==> <G,T'->void,E_d'>)                   | (cond-t), 22b, 21b, 20b

MERGE:
22. P;. |- R(r_d) : (E_z = 0 ==> <G,T'->void,E_d'>)                     | 20a/23b
23. P |-Z R++ : S(G++)                                                  | 9, def of ++
9'. P |-Z R2 : S(G2)                                                    | (reg-file-t), 23, 22, def of R2, def of G2

24. |-Z (R2,C,M,Q,.)                                                    | (S-t), 1, 2, 3, ir=., 5', 6, 7, 8, 9'
*




CASE bz_G-taken-fail:                                                  ** Change Summary:**  none
~~~~~~~~~~~~~~~~~~~~~

Rval(r_z) = 0    Rval(d) =/= 0
--------------------------------------------(bz_G-taken-fail)
(R,C,M,Q, bz_G r_z, r_d )  -->_0  fault

does not apply (fails second assumption)
*




CASE bz_B-taken:                                                        ** Change Summary:**  requires extending Subtyping Lemma
~~~~~~~~~~~~~~~~                                                                                to Stacks, otherwise trivial

(p1) R(d) =/= 0
(p2) R(r_z) = 0
(p3) R(r_d) = R(d)
R2 = R[pc_G -> R(d)][pc_B -> R(r_d)][d -> G 0]
--------------------------------------------------(bz_B-taken)
(R,C,M,Q, bz_B r_z, r_d )  -->_0  (R2,C,M,Q,.)

0. |-Z (R,C,M,Q,z_B r_z, r_d)                                           | Given
1. Dom(P) = Dom(C) union Dom(M_m)                                       | Inversion of (S-t), 0
2. M = M_s #Dom(C) M_m                                                  | Inversion of (S-t), 0
3. P |- C                                                               | Inversion of (S-t), 0
4. Forall c=/=Z. C(R(pc_c)) = bz_B r_z, r_d                             | Inversion of (S-t), 0
5. Forall c=/=Z. P(R(pc_c)) = (D;G;seq(E_d,E_s);E_m;s)-->void           | Inversion of (S-t), 0
6. Exists S. . |- S : D                                                 | Inversion of (S-t), 0
7. P |-Z M_s : S(s)                                                     | Inversion of (S-t), 0
8. P |-Z (M_m,Q) : (S(E_m), S(seq(E_d,E_m)))                            | Inversion of (S-t), 0
9. P |-Z R : S(G)                                                       | Inversion of (S-t), 0

Let R2 = R[pc_G -> R(d)][pc_B -> R(r_d)][d -> G 0]
Let T' = (D',G',seq(E_d',E_s'),E_m';s')

10. P;(D;G;seq(E_d,E_s);E_m) |- bz_B r_z, r_d ==> RT2                   | Inversion of (C-t), 3, 4
11. RT2 = (D;G++;(seq(E_d,E_s));E_m;s)                                  | Inspection of (bz_B-t), def of G2, 10
12. Forall c=/=Z. P(R(pc_c)+1) = RT2                                    | Inversion of (C-t), 3, 4,11

13. P;(.;S(G);S(seq(E_d,E_s));S(E_m);S(s))|- bz_B r_z, r_d              | 10, 11, substitution, 6
       ==>(.;S(G++);S(seq(E_d,E_s));S(E_m);S(s))
14. S(G)(d)= (E_z'=0 ==> <G,T'->void,E_r'>                              | Inversion of (bz_B-t), 13
15. S(G)(r_z)= <B,int,E_z>                                              | Inversion of (bz_B-t), 13
16. S(G)(r_d)= <B,T'->void,E_r>                                         | Inversion of (bz_B-t), 13
17. . |- E_z = E_z'                                                     | Inversion of (bz_B-t), 13
18. . |- E_r = E_r'                                                     | Inversion of (bz_B-t), 13
19. Exists S'. . |- S': D'                                              | Inversion of (bz_B-t), 13
20. . |- S(G) <= S'(G')                                                 | Inversion of (bz_G-t), 13
21. S'(G')(d) = <G,int,0>                                               | Inversion of (bz_G-t), 13
22. S'(G')(pc_G) = <G,int,E_r'>                                         | Inversion of (bz_G-t), 13
21. S'(G')(pc_B) = <B,int,E_r>                                          | Inversion of (bz_G-t), 13
```

```
22. . |- S(seq(E_d,E_s)) = S'(seq(E_d',E_s'))                          | Inversion of (bz_G-t), 13
23. . |- S(E_m) = S'(E_m;)                                             | Inversion of (bz_G-t), 13
23'. . |- S(s) <= S(s')                                                | Inversion of (bz_G-t), 13

24. R2_val(pc_G) = R2_val(pc_B) = R(r_d) = R(d)                        | def of R2, (p3)

SUBCASE a: Z = G
25a. P;. |-Z R(d) : <B,T'->void,E_r'>                                  | Inversion of (reg-file-t), 7, 16
26a. P |- R(d) : T'->void                                              | Inversion of (val-t), assumption, 25a
27a. P |- R2_val(pc_B) : T'->void                                      | 26a, 24

SUBCASE b: Z =/= G
25b. P;. |-Z R(d): (E_z'=0 ==> <G,T'->void,E_r'>                       | Inversion of (reg-file-t), 7, 16
26b. P |- R(d) : T'->void                                              | Inversion of (cond-t), 25b, assumption, (p1)
27b. Forall c=/=Z. P(R2_val(pc_c)) = T' -> void                        | 26b, 24

MERGE:
5'. Forall c=/=Z. P(R2_val(pc_c)) = (D',G',seq(E_d',E_s'),E_m';s) ->void | 27a/27b

6'. Exists S'. . |- S' : D'                                            | 19

7'. P |- M_s : S'(s')                                                  | Subtyping Lemma, 7, 23'

27i. all l in Dom(M_m). exists f. P;M_m;Q |-Z l : b reff               | Inversion of (heap-t), 8
28i. [[S(E_m)]] = M
29i. P |-Z Q : S(seq(E_d,E_s))

30i. [[S(E_m)]] = [[S'(E_m')]]                                         | Inversion of (E-mem-eq), 23
31i. . |- [[S'(E_m)]] = M_m                                            | Exp Eq Trans Lemma, 281, 30i
32i. P |-Z Q : S'(seq(E_d',E_s'))                                      | Inversion/Reconstruction of (Q-t), (Q-zap-t), (Q-emp-t)
                                                                         using Exp Eq Trans Lemma, 29i, 22
8'. P |-Z (M_m,Q) : S'(E_m', seq(E_d',E_m'))                           | (heap-t), 27i, 31i, 32i

28. P |-Z R : S'(G')                                                   | Subtyping Lemma, 9, 20
29. P;. |-Z 0 : <G,int,0>                                              | (val-t)
30. P;. |-Z 0 : S'(G)(d)                                               | 21, 29

SUBCASE a: Z = G
30a.  . |- E_r' : kint                                                 | Inversion of (E-eq), 18
31a. P;. |-Z R(d) : <G,int,E_r'>                                       | (val-zap-t), assumption, 30a
32a. P;. |-Z R(d) : S'(G')(pc_G)                                       | 31a, 22
33a. P;. |-Z R(r_d) : <B,T'->void,E_r>                                 | Inversion of (reg-file-t), 9, 16
34a. P;. |-Z R(r_d) : <B,int,E_r>                                      | Subtyping Lemma, 33a, (subtp-int)
35a. P;. |-Z R(r_d) : S'(G')(pc_B)                                     | 34a, 21

SUBCASE b: Z = B
30b. . |- E_r : kint                                                   | Inversion of (E-eq), 18
31b. P;. |-Z R(r_d) : <B,int,E_r>                                      | (val-zap-t), assumption, 30b
32b. P;. |-Z R(r_d) : S'(G')(pc_B)                                     | 31b, 21
33b. P;. |-Z R(d) : (E_z'=0 ==> <G,T'->void,E_r'>)                     | Inversion of (reg-file-t), 9, 14
34b. <deleted>
35b. P;. |-Z R(d) : <G,T'->void,E_r'>                                  | Inversion of (cond-t), assumption, 33b
36b. P;. |-Z R(d) : <G,int, E_r'>                                      | Subtyping Lemma, 35b, (subtp-int)
37b. P;. |-Z R(d) : S'(G')(pc_G)                                       | 36b, 22

MERGE:
38. P;. |-Z R(d)   : S'(G')(pc_G)                                      | 32a / 37b
39. P;. |-Z R(r_d) : S'(G')(pc_B)                                      | 35a / 32b
9'. P |-Z R2 : S'(G')                                                  | 28, def of R2,, 30, 38, 39

41. |-Z (R2,C,M,Q,.)                                                   | (S-t), 1,2,3,ir=.,5',6',7',8',9'
*




CASE bz_b-taken-fail:                                                  ** Change Summary:**  none
~~~~~~~~~~~~~~~~~~~~~

R(r_z) = 0
R(r_d) =/= R(d)  or  Rval(d) = 0
---------------------------------------------(bz_B-taken-fail)
(R,C,M,Q, bz_B r_z, r_d )  -->_0  fault

does not apply (fails second assumption)
*




CASE jmp_G:                                                            ** Change Summary:**  none
```

```
~~~~~~~~~~~

(p1) R(d) = 0     R2 = R++[d -> R(r_d)]
------------------------------------------------(jmp_G)
(R,C,M,Q, jmp_G rd )  -->_0  (R2,C,M,Q,.)

Similar to bz_G-taken.
*




CASE jmp_G-fail:                                                         ** Change Summary:**  none
~~~~~~~~~~~~~~~~

Rval(d) =/= 0
-------------------------------------------(jmp_G-fail)
(R,C,M,Q, jmp_G rd )  -->_0  fault

does not apply (fails second assumption)
*




CASE jmp_B:                                                               ** Change Summary:**  none
~~~~~~~~~~~

(p1) R(d) =/= 0
(p2) R(r_d) = R(d)
R2 = R[pc_G -> R(d)][pc_B -> R(r_d)][d -> G 0]
---------------------------------------------------(jmp_B)
(R1,C,M,Q1, jmp_B rd )  -->_0  (R2,C,M,Q1,.)

Similar to bz_B-taken.
*



CASE jmp_B-fail:                                                         ** Change Summary:**  none
~~~~~~~~~~~~~~~~

R(r_d) =/= R(d)  or  R(d) = 0
--------------------------------------------(jmp_B-fail)
(R,C,M,Q, jmp_B rd )  -->_0  fault

does not apply (fails second assumption)
*
```

Preservation Part 2

```
2. If |- (R,C,M,Q,ir)
   and (R,C,M,Q,ir)  -->_1^s  (R',C',M',Q',ir')
   then Exists c . |-c (R',C',M',Q',ir')

Proof by induction on the structure of the derivation of (R,C,M,Q,ir)  -->_1^s  (R',C',M',Q',ir').

***CHANGE SUMMARY:*** COMPLETE 4/11/08


CASE reg-zap:
R(a) = n
--------------------------------------------- (reg-zap)
(R,C,M,Q,ir) -->_1 ( R[ a-> n' ], C,M,Q,ir)


0. |- (R,C,M,Q,.)                                                        | Given
1. Dom(P) = Dom(C) union Dom(M_m)                                        | Inversion of (S-t), 0
2. M = M_s #Dom(C) M_m                                                   | Inversion of (S-t), 0
3. P |- C                                                                | Inversion of (S-t), 0
4. Forall c. C(R_val(pc_c)) = ir                                         | Inversion of (S-t), 0
5. Forall c. P(R(pc_c)) = (D;G;seq(E_d,E_s);E_m;s)-->void                | Inversion of (S-t), 0
6. Exists S. .|- S : D                                                   | Inversion of (S-t), 0
7. P |- M_s : S(s)                                                       | Inversion of (S-t), 0
8. P |- (M_m,Q) : (S(E_m), S(seq(E_d,E_m)))                              | Inversion of (S-t), 0
9. P |- R : S(G)                                                         | Inversion of (S-t), 0

subcase on structure of S(G)(a)

SUBCASE a: S(G)(a) = <c,b,E>
10a. . |- E : kint                                                       | Inversion of (reg-file-t), subcase assumption
11a. P;. |-c n' : <c,b,E>                                                | (val-zap-t), 10a

SUBCASE b: S(G)(a) = ( Ez = 0 => <c,b,E> )
10b. . |- Ez : kint and . |- E : kint                                    | Inversion of (reg-file-t), subcase assumption
11b. P;. |-c n' : ( Ez = 0 => <c,b,E> )                                  | (val-zap-cond-t), 10b

SUBCASE c: S(G)(a) = ns
10c. all Z. P;. |-c n' : ns                                              | (ns-t)

MERGE:
10abc. Exists Z. P;. |-Z R[a -> n'](a) : S(G)(a)                         | 11a / 11 / 10c
11abc. Forall c. P;. |-c R : S(G)                                        | Color Weakening Lemma, 9
9abc'. Exists Z. P |-Z R[a -> n'] : S(G)                                 | (reg-file-t), 11, 10

SUBCASE d: a not in dom(S(G))
9d'. Exists Z. P |-Z R[a -> n'] : S(G)                                   | (reg-file-t), 11, 10

7'. P |-Z M_s : S(s)                                                     | Color Weakening Lemma, 7
8'. P |-Z (M_m,Q) : (S(E_m), S(seq(E_d,E_m)))                            | Color Weakening Lemma, 8

4'. Forall c=/=Z. C(R_val(pc_c)) = ir                                    | 4
5'. Forall c=/=Z. P(R(pc_c)) = (D;G;seq(E_d,E_s);E_m;s)-->void           | 5

12. |-c (R[a -> c n'],C,M,Q,ir)                                          | (heap-t), 1, 2, 3, 4', 5', 6, 7', 8', 9abc'/9d'
*



CASE Q1-zap:

Q1 = (seq(n1,n1'),(m1,m'),seq(n2,n2'))
Q2 = (seq(n1,n1'),(m2,m'),seq(n2,n2'))
---------------------------------------------- (Q1-zap)
(R,C,M,Q1,ir) -->_1 (R,C,M,Q2,ir)


0. |- (R,C,M,Q,.)                                                        | Given
```

```
1. Dom(P) = Dom(C) union Dom(M_m)                                          | Inversion of (S-t), 0
2. M = M_s #Dom(C) M_m                                                     | Inversion of (S-t), 0
3. P |- C                                                                  | Inversion of (S-t), 0
4. Forall c. C(R_val(pc_c)) = ir                                           | Inversion of (S-t), 0
5. Forall c. P(R(pc_c)) = (D;G;seq(E_d,E_s);E_m;s)-->void                  | Inversion of (S-t), 0
6. Exists S. .|- S : D                                                     | Inversion of (S-t), 0
7. P |-Z M_s : S(s)                                                        | Inversion of (S-t), 0
8. P |-Z (M_m,Q) : (S(E_m), S(seq(E_d,E_m)))                               | Inversion of (S-t), 0
9. P |-Z R : S(G)                                                          | Inversion of (S-t), 0

let Z = G

10. [[S(E_m)]] = M_m                                                       | Inversion of (heap-t), 8
11. P |- Q : S(seq(E_d,E_s))
12. all l in Dom(M_m). exists f. P;M;Q |- l : b reff


13. all l in Dom(M_m). exists f. P;M;Q2 |-G l : b reff                     | Inversion/Reconstruction of (init-t), (hinit-t),
                                                                                (uninit-t), 12
14. P |-G Q2 : S(seq(E_d,E_s))                                             | Inversion of (Q-t), 11, def of Q2, Reconstruction
                                                                                 with (Q-zap-t)
8'. P |-G (M_m,Q2) : (S(E_m), S(seq(E_d,E_m)))                             | (heap-t), 10, 13, 14

4'. Forall c=/=G. C(R_val(pc_c)) = ir                                      | 4
5'. Forall c=/=G. P(R(pc_c)) = (D;G;seq(E_d,E_s);E_m;s)-->void             | 5
7'. P |-G M_s : S(s)                                                       | Color Weakening Lemma, 7
9'. P |-G R[a -> n'] : S(G)                                                | (reg-file-t), 11, 10


11. |-G (R,C,M,Q2,ir)                                                      | (heap-t), 1, 2, 3, 4', 5', 6, 7', 8', 9'
*



CASE Q2-zap:

Q1 = (seq(n1,n1'),(m,m1),seq(n2,n2'))
Q2 = (seq(n1,n1'),(m,m2),seq(n2,n2'))
---------------------------------------------- (Q1-zap)
(R,C,M,Q1,ir) -->_1 (R,C,M,Q2,ir)

Same as CASE Q1-zap.
*
```

No False Positives Lemma

```
If |- (R,C,M,Q,ir) then Forall n. (R,C,M,Q,ir) -n->_0 (R',C',M',Q',ir') and |- (R',C',M',Q',ir')

Proof: By induction over the multistep definition and use of part (1) of Progress and part (1) of Preservation (with z = .)
```

Definition of Multistep Judgment and Associated Lemmas

```
S -n->_k^ss S'  is a sequence of n steps with k faults resulting in an output sequence ss.


--------------(multi-single)
S -0->_0^() S



S -->_k1^s1 S''    S'' -(n-1)->_k2^ss2  S'
---------------------------------------------(multi-compose)
S -n->_(k1+k2)^(s1,ss2) S'




OUTPUT EQUIVALENCE

a1 = a2                  // addresses are always the same
a1 > l  ==> v1 = v2      // if it's in the heap, then values are also the same
------------------- (out-eq-pair)
(a1,v1) ~l~ (a2,v2)      // l is max of Dom(C)


--------- (out-eq-none)
() ~l~ ()


------- (out-eq-empty)
. ~l~ .


s1 ~l~ s2   ss1 ~l~ ss2
-------------------------- (out-eq-cons)
s1, ss1  ~l~  s2, ss2




------------------- (out-prefix-discard)
.  prefix(l)  s2

s1 ~l~ s2   ss1 prefix(l) ss2
---------------------------- (out-prefix-consume)
s1, ss1  prefix(l)  s2, ss2






Multistep Split Lemma
---------------------

If   S -n->_0^ss S'
then Exists n1,n2,S'',ss1,ss2.
     n = n1 + n2  and  S -n1->_0^ss1 S''  and  S'' -n2->_0^ss2 S'  and  ss = (ss1,ss2)


Proof: by induction on S -n->_0^ss S' (omitted)





Mulitstep Faulty Combine Lemma
------------------------------
```

```
If    S -n1->_0^ss1 S'  and  S' -->_1 Sf'  and  Sf' -n2->_0^ss2 S''
then  S -(n1+1+n2)->_1^(ss1,ss2) S''

Proof: by induction on S -n->_k^ss S' (omitted)




Multistep Fault Detection Lemma:
--------------------------------

If  S1 sim_c S2  and  S1 -n->_0^ss1 S1'  (and let l = max(Dom(S1.C)))
then either  S2 -n->_0^ss2 S2'  and  S1' sim_c S2'  and ss1 ~l~ ss2
         or  Exists m <= n. S2 -m->_0^ss2 fault and ss2 prefix(l) ss1


Proof: By induction on the structure of S1 -n->_0^ss1 S1'



CASE 1: multi-single

------------------(multi-single)
S1 -0->_0^() S1

a1. S1 sim_c S2
a2. S1 -0->_0^() S1

1. S2 -0->_0^() S2                                  | (multi-single)
2. S2 -0->_0^() S2 and S1 sim_c S2 and () ~l~ ()    | 1, a2, (out-eq-none)
*




CASE 2: multi-compose

(p1) S1 -->_0^s1 S1''     (p2) S1'' -(n-1)->_0^ss1  S1'
-----------------------------------------------------------(multi-compose)
S1 -n->_0^(s1,ss1) S1'


a1. S1 sim_c S2
a2. S1 -n->_0^ss1 S1'

1. either                                               | Singlestep Fault Detection Lemma, (a1), (a2), (p1) ]
   S2 -->_0^s2 S2'' and  S1'' sim_c S2'' and s1 ~l~ s2
   or  S2 -->_0^() fault


SUBCASE 2.1: fault does not occur in first step

a3. S2 -->_0^s2 S2''
a4. S1'' sim_c S2''
a5. s1 ~l~ s2

2. either                                               | IH, a3, a4
    S2'' -(n-1)->_0^ss2 S2' and S1' sim_c S2' and ss1 ~l~ss2
    or
    Exists m2 <= (n-1). S2'' -m2->_0^ss2 fault and ss2 prefix(l) ss1


SUBSUBCASE 2.1.1: fault never occurs
a6. S2'' -(n-1)->_0^ss2 S2'
a7. S1' sim_c S2'
a8. ss1 ~l~ss2

3. S2 -n->_0^(s2,ss2) S2'                               | (multi-compose), a3, a6
4. (s2,ss2) ~l~ (s1,ss1)                                | (out-eq-cons), a5, a8
5. S2 -n->_0^(s2,ss2) S2' and S1' sim_c S2'             | 3, a7, 4
   and (s1,ss1) ~l~ (s2,ss2)
*


SUBSUBCASE 2.1.2: fault occurs during remainder of execution

a6. Exists m2 <= (n-1). S2'' -m2->_0^ss2 fault
a7. ss2 prefix(l) ss1

3. S2 -(m2+1)->_0^(s2,ss2) fault                        | (multi-compose), a3, a6
4. m2 + 1 <= n                                          | a6
5. (s2,ss2) prefix(l) (s1,ss1)                          | (out-prefix-consume), a5, a7
```

```
6. S2 -(m2+1)->_0^(s2,ss2) fault                        | 3, 5
   and (s2,ss2) prefix(l) (s1,ss1)
*



SUBCASE 2.2: fault occurs during first step
a3. S2 -->_0^() fault

2. fault -0->_0^() fault                                | (multi-single)
3. S2 -1->_0^(s2) fault                                 | (multi-compose), a3, 2
4. 1 <= n                                               | p2, def of (multi-compose)
6. () ~l~ ()                                            | (out-eq-none)
5. () prefix(l) ((),ss1)                                | (out-prefix-consume), , 6, (out-prefix-discard)
6. S2 -1->_0^(s2) fault and () prefix ((),ss1)          | 3, 5

 *
```

Lemmas for the Fault Detection Theorem

```
Coloring Preservation Lemma - can deduce the coloring changes of each step by the instruction that is executed
----------------------------

1.If  |- (R,C,M,Q,ir) : K    and (R,C,M,Q,ir) -->_0 (R',C',M',Q',ir')
  then  |- (R',C',M',Q',ir') : K'
  and
    ir = .              ==>     K' = K
    ir = op rd,rs,rt    ==>     K' = K[ rd -> K(rs) ]
    ir = op rd rs n     ==>     K' = K[ rd -> K(rs) ]
    ir = mov rd, n      ==>     all c. K' = K[ rd -> c ]
    ir = ld_G rd rs     ==>     K' = K[ rd -> G ]
    ir = ld_B rd rs     ==>     K' = K[ rd -> B ]
    ir = sld_c rd n     ==>     K' = K[ rd -> K(R(spc)+n) ]
    ir = st_G rd rs     ==>     K' = K
    ir = st_B rd rs     ==>     K' = K
    ir = sst n rv       ==>     K' = K[ R(spc)+n -> K(rv) ]
    ir = malloc rg rb   ==>     K' = K[ rg -> G ][ rb -> B ][ R(rg) -> O ]
    ir = salloc n       ==>     K' = (R(spg)-1 -> ns),..., (R(spg)-n -> ns), K
    ir = sfree n        ==>     K = R(spg) -> k, R(spg)+1 -> k1, ..., R(spg)+n-1 -> kn, K'
    ir = bz_G rz rd     ==>     K' = K[ d -> G ]
    ir = bz_B rz rd     ==>     K' = K[ d -> G ]
    ir = jmp_G rt       ==>     K' = K
    ir = jmp_B rt       ==>     K' = K

2. If  |- (R,C,M,Q,ir) : K    and (R,C,M,Q,ir) -->_1 (R[a -> n'],C,M,Q,ir)  then  |-^K(a) (R[a -> n'],C,M,Q,ir) : K
   If  |- (R,C,M,Q,ir) : K    and (R,C,M,Q,ir) -->_1 (R,C,M,Q',ir)                then  |-G     (R,C,M,Q',ir) : K

Proof:

1. Similar to the proof of Preservation Part 1. Inspection of appropriate instruction typing rule
gives changes to G;s, which translate into changes in K.
2. Similar to the Proof of Preservation Part 2. Color of zapped value is used to type check resulting state.




K Lemma - can determine things about the structure of K given typing information
-------

If  P |- R : G  and  P |- Ms : s  and  P |- (M,Q) : (Em,seq(Ed,Es))  and  M = Ms # Mh
and K = color(R,G),color(Ms,s),color(M_m)
then
1. G(r) =< <c,b,E>       ==>  K(r) = c
2. G(r) =< Ez=0=><c,b,E> ==>  K(r) = c
3. G(r) =< <c,b reff,E>  ==>  K(R(r)) = none
4. .;s |- E : <c,b,E'>   ==>  K(E) = c
5. K(pcg) = G
6. K(pcb) = B

Proof:

1/2. Inspection of K-def, P |- R : G and subtyping definitions
3. By Canonical Forms,  R(r) in Dom(M_m). by K-def, K(R(r)) = none
4. by inspection of K-def, Inversion of P |- Ms: s
5/6. Inversion of P |- R : G, k-def




Colored Pairs Lemma - info about pairs of similar values
--------------------

1. If  k n1 sim_c k n2  and  k =/= c            then n1 = n2
2. If  G n1 sim_c G n2  and  B n1' sim_c B n2'  then either n1 = n2 or n1' = n2'
3. If  k n1 sim_c k n2  and  k n1' sim_c k n2'  then  k (n1 op n1') sim_c  k (n2 op n2')
4. If  k n1 sim_c k n2  and                     then  k (n1 op n) sim_c  k (n2 op n)
```

```
5. If  Q1 sim_B Q2                              then Q1 = Q2

6. If k n1 sim_c k n2  and  n1 in Dom(M1)  and  n2 in Dom(M2)  then  k M2(n1) sim_c k M2(n2)

Proof:
1-4. by (sim-val) snd/or (sim-val-zap)
5. by the (sim-Q) rules and 1.
6. by sim-val and/or (sim-val-zap)




Coloring Update Lemma - updating one piece of state does not affect coloring of another piece of state
---------------------

1. If  K |- M1 sim_c M2  then forall a. K[a -> c] |- M1 sim_c M2
2. If  K |- R1 sim_c R2  then forall a. K[l -> c] |- R1 sim_c R2

proof: by (sim-R) and (sim-M), set of registers a does not intersect set of locations l





Fault Similarity Lemma:
-----------------------

If  |- S : K  and  S -->_1 Sf
then Exists c. S sim_c Sf


Proof: By case analysis on the definition of S -->_1 Sf


CASE 1: reg-zap

R(a) = n
------------------------------------------------ (reg-zap)
(R,C,M,Q,ir) -->_1 (R[a -> n'],C,M,Q,ir)

1. |-K(a) (R[a -> n'],C,M,Q,ir) : K             | Coloring Preservation Part 2, assumptions
2. K(a) n sim_c K(a) n'                            | (sim-val-zap)
3. K |- R sim_c R[a -> n']                      | (sim-R), (sim-val), 2
4. Q sim_c Q                                       | (sim-Q)
5. K |- M sim_c M                                  | (sim-M)
6. (R,C,M,Q,ir) sim_c (R[a -> n'],C,M,Q,ir)     | (sim-S), assumption, 1, 3, 4, 5
*


CASE 2: Q1-zap

(p1) Q1 = ( seq(n1,n1'), (m1, m2), seq(n2,n2') )
(p2) Q2 = ( seq(n1,n1'), (m1',m2), seq(n2,n2') )
------------------------------------------------ (Q1-zap)
(R,C,M,Q1,ir) -->_1 (R,C,M,Q2,ir)

1. K |- R sim_G R                               | (sim-R), (sim-val)
2. K |- M sim_G M                               | (sim-M)
3. Q1 sim_G Q2                                  | (sim-Q-zap), p1, p2
4. |-G (R,C,M,Q2,ir) : K                        | Coloring Preservation Part 2, assumptions
5. (R,C,M,Q1,ir) sim_G (R,C,M,Q2,ir)            | (sim-S), assumption, 4, 1, 2, 3
*


CASE 3: Q2-zap

similar to CASE 2.
*
```

Singlestep Fault Detection

```
Singlestep Fault Detection Lemma:
---------------------------------

If    S1 sim_c S2  and  S1 -->_0^s1 S1'
then  S2 -->_0^s2 S2'
and Forall S2'. S2 -->_0^s2 S2'
    either  S2 -->_0^s2 S2'  and  S1' sim_c S2'  and  s1 ~max(Dom(S1.C))~ s2
    or      S2 -->_0^() fault




Proof:  By case analysis of ir in S1


a1. S1 sim_c S2
a2. S1 -->_0^s1 S1'

1. S = (R1,C,M1,Q1,ir) and S2 = (R2,C,M,Q2,ir)              | Inspection of (sim-S), a1

2. |- (R1,C,M1,Q1,ir) : K                                   | Inversion of (sim-S), a1
3. |-c (R2,C,M2,Q2,ir) : K
4. K |- R1 sim_c R2
5. K |- M1 sim_c M2
6. Q1 sim_c Q2

7. all a. K(a) R1(a) sim_c K(a) R2(a)                       | Inversion of (sim-R), 4

8. Dom(M1) = Dom(M2)                                        | Inversion of (sim-M), 5
9. all l in Dom(M1). K(l) M1(l) sim_c K(l) M2(l)


10. P;(D;G;seq(E_d,E_s);E_m;s) |- ir ==> RT                 | Inversion of (S-t), 2, Inversion of (C-t)
12. P;(.;S(G);S(seq(E_d,E_s));S(E_m);S(s))|- ir ==> S(RT)   | substitution, 6, 13

13. K |- R1++ sim_c R2++                                    | def of ++, Color Pairs Lemma, (k-def)




CASE FETCH :
~~~~~~~~~~~

By Preservation of |- S1 and  S1 -->_0 S1', and inspection of rules, following must apply:

(p1) R1(pc_G) = R1(pc_B)
(p2) R1(pc_G) in Dom(C)
------------------------------------------------(fetch)
(R1,C,M1,Q1,.)  -->_0  (R1,C,M1,Q1,C(R1(pc_G)))


Case on R2(pc_G) =? R2(pc_B)


SUBCASE FETCH 1.1: One of the pc's was zapped
a3. R2(pc_G) =/= R2(pc_B)

20. (R2,C,M2,Q2,.) -->_0 fault                              | (fetch-fail), a3
21. S2 -->_0^() fault                                       | 4
*


SUBCASE FETCH 1.2: Neither pc was zapped
a3. R2(pc_G) = R2(pc_B)

20. K(pc_G) = G, K(pc_B) = B                                | (K-def)
21. either R1(pc_G) = R2(pc_G) or R1(pc_B) = R2(pc_B)       | Colored Pairs Lemma, 7, 20
22. R2(pc_G) in Dom(C)                                      | Transitivity, p1, a3, 21, p2
```

```
23. (R2,C,M2,Q2,ir) -->_0 (R2,C,M2,Q2,C(R2(pc_G)))          | (fetch), a3, 22

24. |-  (R1,C,M1,Q1,C(R1(pc_G))) : K                        | Coloring Preservation Lemma, 2, a2, ir = .
25. |-c (R2,C,M2,Q2,C(R2(pc_G))) : K                        | Coloring Preservation Lemma, 3, 23, ir = .

26. R1(pc_G) = R2(pc_G)                                     | Transitivity, p1, a3, 21
27. (R1,C,M1,Q1,C(R1(pc_G))) sim_c (R2,C,M2,Q2,C(R2(pc_G))) | (sim-S), 24, (25,26), 4, 5, 6

28. S2 -->_0 S2' and  S1' = S2' and () = ()                 | 27
*







CASE op2r:
~~~~~~~~~~

By inspection of S1 -->_0 S1', S1 must step as follows:

(d1) R1' = R1++[ rd -> R1(rs) op R1(rt) ]
------------------------------------------------------- (op2r)
(R1,C,M1,Q1, op rd, rs, rt )  -->_0  (R1',C,M1,Q1,.)


(d2) let R2' = R2++[ rd -> R2(rs) op R2(rt) ]

20. (R2,C,M2,Q2,.) -->_0 (R2',C,M2,Q2,.)                    | (op2r), d2

21. |-  (R1',C,M1,Q1,.) : K[ rd -> K(rs) ]                  | Coloring Preservation Lemma, 2, a2, ir = op2
22. |-c (R2',C,M2,Q2,.) : K[ rd -> K(rs) ]                  | Coloring Preservation Lemma, 3, 23, ir = op2

23. K(rs) = K(rt)                                           | Inspection of (op2r-t), 13, def of K
24. K(rs) (R1(rs) op R1(rt)) sim_c K(rs) (R2(rs) op R2(rt)) | Color Pairs Lemma, 7, 23
25. K[ rd -> K(rs) ] |- R1' sim_c R2'                       | (sim-R), 7, 14, 24

26. K[ rd -> K(rs) ] |- M1' sim_c M2'                       | Coloring Update Lemma, 5

27. (R1',C,M1,Q1,.) sim_c (R2',C,M2,Q2,.)                   | 21, 22, 25, 26, 6
28. S2 -->_0 S2' and  S1' sim_c S2'  and  () = ()           | 20, 28
*




CASE  op1r :
~~~~~~~~~~~~~

By inspection of S1 -->_0 S1', S1 must step as follows:

(d1) R1' = R1++[ rd -> R1(rs) op n ]
---------------------------------------------------------(op1r)
(R1,C,M1,Q1, op rd, rs, n )  -->_0  (R1',C,M1,Q1,.)


(d2) let R2' = R2++[ rd -> R2(rs) op n ]

20. (R2,C,M,Q2,.) -->_0 (R2',C,M,Q2,.)                      | (op1r), d2

21. |-  (R1',C,M1,Q1,.) : K[ rd -> K(rs) ]                  | Coloring Preservation Lemma, 2, a2, ir = op1
22. |-c (R2',C,M2,Q2,.) : K[ rd -> K(rs) ]                  | Coloring Preservation Lemma, 3, 23, ir = op1

23. K(rs) (R1(rs) op n) sim_c K(rs) (R2(rs) op n)           | Color Pairs Lemma, 7
25. K[ rd -> K(rs) ] |- R1' sim_c R2'                       | (sim-R), 7, 14, 23

26. K[ rd -> K(rs) ] |- M1' sim_c M2'                       | Coloring Update Lemma, 5

27. (R1',C,M1,Q1,.) sim_c (R2',C,M2,Q2,.)                   | 21, 22, 25, 26, 6
28. S2 -->_0 S2' and  S1' sim_c S2'  and  () = ()           | 20, 27
*






CASE mov-n:
~~~~~~~~~~

By inspection of S1 -->_0 S1', S1 must step as follows:
```

```
------------------------------------------------------------(mov)
(R1,C,M1,Q1, mv rd, n )  -->_0  (R1++[rd -> n] ,C,M1,Q1,.)


20. (R2,C,M,Q2,.) -->_0 (R2++[rd -> n],C,M,Q2,.)            | (mov)

21. |-  (R1++[rd -> n],C,M1,Q1,.) : all c. K[ rd -> c' ]    | Coloring Preservation Lemma, 2, a2, ir = mov
22. |-c (R2++[rd -> n],C,M2,Q2,.) : all c. K[ rd -> c' ]    | Coloring Preservation Lemma, 3, 23, ir = mov

23. all c'. c' n sim_c c' n                                 | (sim-val)
24. all c'. K[rd -> c'] |- R1++[rd -> n] sim_c R2++[rd -> n]| (sim-R), 7, 14, 23
25. all c'. K[rd -> c'] |- M1' sim_c M2'                    | Coloring Update Lemma, 5

26. (R1',C,M1,Q1,.) sim_c (R2',C,M2,Q2,.)                   | 21, 22, 24, 25, 6
27. S2 -->_0 S2' and  S1' sim_c S2'  and  () = ()           | 20, 26
*




CASE mov-r:
~~~~~~~~~~

Similar to mov-n.
*




CASE LD_G:
~~~~~~~~~~

By inspection of |- S1, S1 -->_0 S1' must be one of the 4 ld_G rules:

(ld_G-rand) does not apply by the Well-typed Domain Lemma
(ld_G-fail) does not apply by Preservation Part 1

subcase A: S1 steps using (ld_G-queue)
20a1. (R1,C,M1,Q1, ld_G rd, rs )  -->_0  (R1++[rd -> n],C,M1,Q1,.)
20a2. find(Q1,R1(rs)) = (R1(rs),n)

subcase B: S1 steps using (ld_G-mem)
20b1. (R1,C,M1,Q1, ld_G rd, rs )  -->_0  (R1++[rd -> M1(R1(rs))],C,M1,Q1,.)
20b2. find(Q1,R1(rs)) = ()
20b3. R1(rs) in Dom(M1)


20. S(G)(rs) =< <G,b refh, Es'>                                   | Inversion of (ld_G-t), 13
21. K(rs) = G                                                     | K Lemma, (Inversion of (S-t),2), 20
22. K(R1(rs)) = none                                              | K Lemma, (Inversion of (S-t),2), 20

23. |- S1' : K[rd -> G]                                           | Coloring Preservation Lemma, 2, a2, ir = ld_G


SUBCASE 1: c = B  --  S2 must step using same rule

231. Q1 = Q2                                                      | 6, def of (sim-Q)
241. R1(rs) = R2(rs)                                              | Colored Pairs Lemma, 21, assumption, 7
251. M1(R1(rs)) = M2(R2(rs))                                      | Colored Pairs Lemma, 9, 22, 24


261. a: find(Q2,R2(rs)) = (R2(rs),n)                              | subcase A/B applies, 22, 23
     b: find(Q2,R2(rs)) = () and  R2(rs) in Dom(M2)

271. a: S2 -->_0 (R2++[rd -> n],C,M2,Q2,.)                        | (ld_G-queue), (20a2, 241, 231), no other ld_G rule applies
     b: S2 -->_0 (R2++[rd -> M2(R1(rs))],C,M2,Q2,.)               | (ld_G-mem), (20b2, 20b3, 231, 241, 251), no other ld_G rule applies

281. a: |-c (R2++[rd -> n],C,M2,Q2,.) : K[rd -> G]                | Coloring Preservation Lemma, 3, 271, ir = ld_G
     b: |-c (R2++[rd -> M2(R1(rs))],C,M2,Q2,.) : K[rd -> G]

291. a: G n sim_B G n                                             | (sim-val)
     b: G M1(R1(rs)) sim_B G M2(R2(rs))                           | (sim-val), 25c

301. a: R1++[rd -> n] sim_B R2++[r1 -> n]                         | (sim-R), 7, 13, 291
     b: R1++[rd -> M1(R1(rs))] sim_B R2++[r1 -> M2(R2(rs))]

311. a: S1' sim_c (R2++[rd -> n],C,M2,Q2,.)                       | (sim-S), 301, 5, 6
     b: S1' sim_c (R2++[rd -> M2(R1(rs))],C,M2,Q2,.)

321. a/b: S2 -->_0 S2'  and  S1' sim_B S2'  and () = ()           | 281, 311



SUBCASE 2: c = G  --  S2 may step using any of the 4 applicable rules
```

```
232. S2 -->_0 S2' where S2' is either                          | Progress Part 2, 3, and inspection of rules for ld_G
     c: (R2++[rd ->n],C,M2,Q2,.)                                   (ld_G-queue)
     d: (R2++[rd ->M1(R1(rs))],C,M2,Q2,.)                          (ld_G-mem)
     e: (R2++[rd ->n'],C,M2,Q2,.)                                  (ld_G-rand)
     f: fault                                                      (ld_G-fail)

242. for all possible pairings,                                | (sim_R), (sim-val-zap), 13
     K[rd->G] |-
     {{ R1++[rd -> n],  R1++[rd -> M1(R1(rs))] }}                  a/b: possible values for S1'
     sim_G
     {{ R2++[rd ->n], R2++[rd ->M1(R1(rs))],  R2++[rd ->n'] }}     c/d/e: possible values for S2'

252. if S2 -->_0 (R2',C,M2',Q2',.) then                        | c/d/e: Coloring Preservation Lemma, 3, 232, ir = ld_G
     |-c (R2',C,M2',Q2',.) : K[rd -> G]

262. if S2 -->_0 (R2',C,M2',Q2',.) then                        | a/b and c/d/e, (sim-S), 23, 252, 242, 5, 6
     S1' sim_G (R2',C,M2',Q2',.)

26d. either
     S2 -->_0 S2'  and  S1' sim_G S2' and () = ()              | a/b and c/d/e
     or
     S2 -->_0^() fault                                         | a/b and f
*




CASE LD_B:
~~~~~~~~~~

By inspection of |- S1, S1 -->_0 S1' must be one of the 3 ld_B rules:

(ld_B-rand) does not apply by the Well-typed Domain Lemma
(ld_B-fail) does not apply by Preservation Part 1

therefore S1 must step using (ld_B-mem)
20. (R1,C,M1,Q1, ld_B rd, rs )  -->_0  (R1++[rd -> M1(R1(rs))],C,M1,Q1,.)
21. R1(rs) in Dom(M1)


23. S(G)(rs) =< <B,b ref1, Es'>                                | Inversion of (ld_B-t), 13
24. K(rs) = B                                                  | K Lemma, (Inversion of (S-t),2), 23
25. K(R1(rs)) = none                                           | K Lemma, (Inversion of (S-t),2), 23

26. |- S1' : K[rd -> B]                                        | Coloring Preservation Lemma, 2, a2, ir = ld_B



SUBCASE 1: c = B  --  S2 may step with one of three rules

231. Q1 = Q2                                                   | 6, def of (sim-Q), assumption

241. S2 -->_0^() S2' where S2' is either                       | Progress Part 2, 3, and inspection of rules for ld_G
     a: (R2++[rd -> n'),C,M2,Q2,.)                                   (ld_B-mem)
     b: (R2++[rd -> M2(R2(rs)),C,M2,Q2,.)                            (ld_B-rand)
     c: fault                                                        (ld_B-fail)

251. for all possible pairings,                                | a/b: (sim_R), (sim-val-zap), 13
     K[rd->B] |-
     {{ R1++[rd -> M1(R1(rs))] }}                                  possible value for S1'
     sim_B
     {{ R2++[rd ->n'], R2++[rd ->M2(R2(rs))] }}                    a/b: possible values for S2'

261. if S2 -->_0 (R2',C,M2',Q2',.) then                        | a/b: Coloring Preservation Lemma, 3, 241, ir = ld_B
     |-c (R2',C,M2',Q2',.) : K[rd -> B]

262. if S2 -->_0 (R2',C,M2',Q2',.) then                        | a/b: (sim-S), 261, 26, 251, 5, 6
     S1' sim_G (R2',C,M2',Q2',.)

26d. either
     S2 -->_0 S2'  and  S1' sim_G S2' and () = ()              | a/b, 241, 262
     or
     S2 -->_0^() fault                                         | c
*


SUBCASE 2: c = G  --  S2 must step with same rule (ld_B-mem)

232. Q1 = Q2                                                   | 6, def of (sim-Q)
242. R1(rs) = R2(rs)                                           | Colored Pairs Lemma, 24, assumption, 7
```

```
252. R2(rs) in Dom(M2)                                          | 8, 242
262. S2 -->_0 (R2++[rd --> M2(R2(rs))],C,M2,Q2,.)               | (ld_B-mem), 252

272. M1(R1(rs)) = M2(R2(rs))                                    | Colored Pairs Lemma, 9, 25, 242
282. B M1(R1(rs)) sim_B B M2(R2(rs))                            | (sim-val), 272
292. K[rd ->B] |-                                               | (sim-R), 7, 13, 282
     R1++[rd --> M1(R1(rs))] sim_B R2++[rd --> M2(R2(rs))]

302. |-c (R2++[rd --> M2(R2(rs))],C,M2,Q2,.) : K[rd->B]         | Coloring Preservation Lemma, 3, 262, ir = ld_B
312. S1' sim_c (R2++[rd --> M2(R2(rs))],C,M2,Q2,.)              | (sim-S), 26, 302, 292, 5, 6

*




CASE SLD:
~~~~~~~~~

20. S1 -->_0 (R1++[rd -> M1(R1(spc)+n)],C,M1,Q1,.)              | a2, Preservation Part 1, inspection of sld_c rules
21. R(spc) + n in Dom(M1)                                       | Inversion of (sld_c), 20

23. S(G)(spc) = <c',sptr, Ec>                                   | Inversion of (ld_B-t), 13
24. .;s |- Ec + n : <c,b,E>

25. R1(spc) = Ec                                                | Canonical Forms, Inversion of (S-t), a1, 23
26. K(R1(spc)+n) = c'                                           | K Lemma, (Inversion of (S-t),2), 24, 25
27. K(spc) = c'                                                 | K Lemma, (Inversion of (S-t),2), 23

subcase on  R2(spc) in Dom(M2)

SUBCASE A: R2(spc) in Dom (M2)
27a. S2 -->_0(R2++[rd -> M2(R2(spc)+n)],C,M2,Q2,.)              | (sld_c), assumption

28a. c' M1(R(spc)+n) sim_c c' M2(R(spc)+n)                      | Colored Pairs Lemma, 21, assumption, 26, (Colored Pairs Lemma, 7, 27)
29a. K[rd -> c'] |-                                             | (sim-R), 7, 13, 28a
    R1++[rd -> M1(R1(spc)+n)] sim_c R2++[rd -> M2(R2(spc)+n)]

30a. |- S1' : K [ rd -> c']                                     | Coloring Preservation Lemma, 2, 26, ir = sld
31a. |- S2' : K [ rd -> c']                                     | COloring Preservation Lemma, 3, 26, assumption, ir = sld

32a. K[rd -> c'] |- M1 sim_c M2                                 | Coloring Update Lemma, 5

33a. S1' sim_c (R2++[rd -> M2(R2(spc)+n)],C,M2,Q2,.)            | (sim-S), 30a, 31a, 29a, 32a, 6

34a. S2 -->_0 S2'  and  S1' sim_c S2'  and () = ()                27a, 33a


SUBCASE B: R2(spc) not in Dom (M2)
27b. S2 -->_0^() fault                                          | (sld_c-fail), assumption
*



CASE ST_G:
~~~~~~~~~~

20. S1 -->_0 (R1++,C,M1,((R1(rd),R1(rs)),Q1),.)                 | a2, Preservation Part 1, inspection of st_G rules
21. S2 -->_0 (R2++,C,M2,((R2(rd),R2(rs)),Q2),.)                 | (st_G-queue)

22. |-  S1' : K                                                 | Coloring Preservation Lemma, 2, 20, ir = st_G
23. |-c S2' : K                                                 | Coloring Preservation Lemma, 3, 21, ir = st_G

24. S(G)(rd) = <G, b reff, Ed'>                                 | Inversion of (st_G), 13
25. S(G)(rs) = <G,b,Es'>
26. K(rd) = K(rs) = G                                           | K Lemma, (Inversion of (S-t), 2), 24, 25

27. G R1(rd) sim_c G R2(rd)  and G R1(rs) sim_c G R2(rs)        | 7, 26
28. ((R1(rd),R1(rs)),Q1) sim_c ((R2(rd),R2(rs)),Q2)             | (sim-Q), 6, 27

29. S1' sim_c S2'                                               | (sim-S), 22, 23, 13, 5, 28
*




CASE ST_B:
~~~~~~~~~~

20.(R1,C,M1,(Q1',(n1,n1'),.)                                    | a2, Preservation Part 1, inspection of st_B rules
   -->_0^(n1,n1') (R1++,C,M1[n1 -> n1'],Q1',.)
21. R1(rd) = n1                                                 | Inversion of (sst), 20
```

```
22. R1(rs) = n1'

23. S(G)(rs) = <B,b, Es>                                        | Inversion of (st_B), 13
24. . |- S(G)(rd) <= <B,b refh, Ed'>

25. K(rs) = B  and K(rd) = B                                    | K Lemma, (Inversion of (S-t),2), 23, 24
26. B R1(rs) sim_c B R2(rs)   and  B R1(rd) sim_c B R2(rd)      | 7, 25

27. Q2 = (Q2', (n2,n2'))                                        | Inspection/Inversion of (sim-Q), 6
28. G n1 sim_c G n2  and  G n1' sim_c G n2'
29. Q1' sim_c Q2'

subcase on (R2(rd) = n2  and  R2(rs) = n2')

SUBCASE A: (R2(rd) = n2  and  R2(rs) = n2')
30a. S2 -->_0^(n2,n2') (R2++,C,M2[n2 -> n2'],Q2',.)             | (st_B), assumption

31a. c=G ==> R1(rd) = R2(rd)  and  R1(rs) = R2(rs)              | Colored Pairs, 26
32a. c=G ==>     n1 = n2      and     n1' = n2'                 | 31a, 21, 22, assumption

33a. c=B ==>     n1 = n2      and     n1' = n2'                 | Colored Pairs, 28

34a. n1 = n2, n1' = n2'                                         | 32a / 33a

35a. K |- M1[n1 -> n1'] sim_c M2[n1 -> n2']                     | (sim-val), 9, 34a

36a. |- S1' : K                                                 | Coloring Preservation Lemma, 2, 20, ir = st_B
37a. |- S2' : K                                                 | Coloring Preservation Lemma, 3, 30a, ir = st_B

38a. S1' sim_c (R2++C,M2[n2 -> n2'],Q2,.)                       | (sim-S), 36a, 37a, 13, 35a, 29

35a. S2 -->_0 S2' and S1' sim_c S2' and (n1,n1') ~l~ (n2,n2')   | 30a, 38a, 34a


SUBCASE B: (R2(rd) =/= n2  or  R2(rs) =/= n2')
27b. S2 -->_0^() fault                                          | (sst-fail), assumption
*




CASE SST:
~~~~~~~~~

20. S1 -->_0^(R1(spg)+n,R1(rv))                                 | a2, Preservation Part 1, inspection of sst rules
           (R1++,C,M1[R1(spg)+n ->R1(rv)],Q1,.)
21. R(spg) + n in Dom(M1)                                       | Inversion of (sst), 20
22. R(spb) = R(spg)

23. S(G)(spg) = <G,sptr, Eg>  and S(G)(spb) = <B,sptr,Eb>       | Inversion of (ld_B-t), 13
24. . |- s [Eg + n -> <c,b,Ev>] = s'
25. S(G)(rv) = <c,b,Ev>

26. R1(spg) = Eg  and R2(spb) = Eb                              | Canonical Forms, Inversion of (S-t), a1, 23
27. K(rv) = c'                                                  | K Lemma, (Inversion of (S-t),2), 25
28. K(spg) = G  and K(spb) = B                                  | K Lemma, (Inversion of (S-t),2), 23
29. either R1(spg) = R2(spg) or R1(spb) = R2(spb)               | Colored Pairs Lemma, 7, 28, 29

subcase on  ( R2(spg)=R2(spb) and R2(spg)+n in Dom(M2) )

SUBCASE A: R2(spg)=R2(spb) and R2(spg)+n in Dom(M2)
27a. S2 -->_0^(R2(spg)+n,R2(rv))                                | (sst), assumption
            (R2++C,M2[R2(spg)+n ->R2(rv)],Q2,.)

28a. R1(spg)+n = R2(spg)+n                                      | Transitivity, 22, assumption, 29
29a. R1(rv) R1(rv) sim_c R1(rv) R2(rv)                          | 7, 27
30a. K[ R1(spg)+n -> K(rv)]                                     | (sim-M), 8, 9, 28a, 29a
     |- M1[R1(spg)+n ->R1(rv) sim_c M2[R2(spg)+n ->R12(rv)]

31a. |-  S1' : K [ R1(spg)+n -> K(rv)]                          | Coloring Preservation Lemma, 2, 20, ir = sld
32a. |-c S2' : K [ R2(spg)+n -> K(rv)]                          | COloring Preservation Lemma, 3, 27a, ir = sld

33a. K[ R1(spg)+n -> K(rv)] |- R1++ sim_c R2++                  | Coloring Update Lemma, 13

34a. S1' sim_c (R2++C,M2[R2(spg)+n ->R2(rv)],Q2,.)              | (sim-S), 31a, 32a, 30a, 33a, 6

35a. R1(spg) + n in Dom(Ms) and R1(spg) + n < max(Dom(c))       | 24, 26, Inversion of (S-t), 2
36a. (R1(spg)+n,R1(rv)) ~(max(Dom(C))~ (R2(spg)+n,R2(rv))       | 28a,

37a. S2 -->_0 S2'  and  S1' sim_c S2'  and s1 ~l~ s2             | 27a, 34a, 36a


SUBCASE B: ( R2(spg)=/= R2(spb) or R2(spg)+n not in Dom(M2) )
```

```
27b. S2 -->_0^() fault                                          | (sst-fail), assumption
*
```

```
CASE MALLOC:
~~~~~~~~~~~~

d1. let n = max(Dom(M1)) + 1

20. S1 -->_0 (R1++[rg -> n][rb -> n],C,(M1,n->0),Q1,.)          | a2, Preservation Part 1, inspection of malloc rule

21. n = max(Dom(M2)) + 1                                        | 8, d1
22. S2 -->_0 (R2++[rg -> n][rb -> n],C,(M2,n->0),Q2,.)          | (malloc), 21

23. all k'. all n'. k' n' sim_c k' n'                           | (sim-val)
24. K[rg -> G][rb -> B][R(rg) -> none]                          | (sim-R), 13, 23, Coloring Update Lemma
    |- R1++[rg -> n][rb -> n] sim_c R2++[rg -> n][rb -> n]

25. K[rg -> none][rb -> none] |- (M1,n->0) sim_c (M2, n->0)     | (sim-M), 8, 9, 23, Coloring Update Lemma

31. |-  S1' : Krg -> none][rb -> none][R(rg) -> none]           | Coloring Preservation Lemma, 2, 20, ir = malloc
32. |-c S2' : K[rg -> none][rb -> none][R(rg) -> none]          | COloring Preservation Lemma, 3, 22, ir = malloc


33. (R1++[rg -> n][rb -> n],C,(M1,n->0),Q1,.)                   | (sim-S), 31, 32, 24, 25, 6
    sim_c
    (R2++[rg -> n][rb -> n],C,(M2,n->0),Q2,.)
*
```

```
CASE SALLOC:
~~~~~~~~~~~~

d1. let n = min(Dom(M1))
d2. let M1' = M1, m-1 -> 0, ..., m-n -> 0

20. S1 -->_0 (R1++[spg->R1(spg)-n][spb->R1(spb)-n],C,M1',Q1,.)  | a2, Preservation Part 1, inspection of salloc rule


21. n = min(Dom(M1)=  min(Dom(M2)                               | 8, d1
d3. let M2' = M2, m-1 -> 0, ..., m-n -> 0
22. S2 -->_0 (R2++[spg->R2(spg)-n][spb->R2(spb)-n],C,M2',Q2,.)  | (salloc), 21, d3

d4. let K' = (R(spg)-1 -> none),..., (R(spg)-n -> none, K

23. K(spg) R1(spg)-n sim_c K(spg) R2(spg)-n                     | Colored Pairs Lemma, 7
24. K(spb) R1(spb)-n sim_c K(spb) R2(spb)-n                     | Colored Pairs Lemma, 7
25. K' |- R1++[spg -> R1(spg)-n][spb -> R1(spb)-n]              | (sim-R), 13, 23, 24, Coloring Update Lemma, d4
          sim_c R2++[spg -> R2(spg)-n][spb -> R2(spb)-n]

26. none 0 sim_c none 0                                         | (sim-val)
27. K' |- M1' sim_c M2'                                         | (sim-M), d2, d3, 8, 9, 26


28. |-  S1' : K'                                                | Coloring Preservation Lemma, 2, 20, ir = salloc
29. |-c S2' : K'                                                | Coloring Preservation Lemma, 3, 22, ir = salloc


40. ((R1++[spg->R1(spg)-n][spb->R1(spb)-n],C,M1',Q1,.)          | (sim-S), 28, 29, 25, 27, 6
    sim_c
    (R2++[spg->R2(spg)-n][spb->R2(spb)-n],C,M2',Q2,.)
*
```

```
CASE SFREE:
~~~~~~~~~~~

d1. let n = min(Dom(M1))

20. S1 -->_0 (R1++[spg->R1(spg)+n][spb->R1(spb)+n],C,M1',Q1,.)  | a2, Preservation Part 1, inspection of salloc rule
21. M1 = M1', m -> v1, ..., m-n+1 -> vn                         | Inversion of (sfree), 20


22. n = min(Dom(M2)                                             | 8, d1
```

```
23. M2 = M2', m-1 -> 0, ..., m-n -> 0                             | 8, 21
24. S2 -->_0 (R2++[spg->R2(spg)-n][spb->R2(spb)-n],C,M2',Q2,.)  | (sfree), 22, 23

25. |-  S1' : K'                                                  | Coloring Preservation Lemma, 2, 20, ir = sfree
26. K = R(spg) -> k, R(spg)+1 -> k1, ..., R(spg)+n-1 -> kn, K'

27. K(spg) R1(spg)+n sim_c K(spg) R2(spg)+n                       | Colored Pairs Lemma, 7
28. K(spb) R1(spb)+n sim_c K(spb) R2(spb)+n                       | Colored Pairs Lemma, 7
29. K' |- R1++[spg -> R1(spg)+n][spb -> R1(spb)+n]                | (sim-R), 13, 27,28, Coloring Update Lemma, d4
          sim_c R2++[spg -> R2(spg)+n][spb -> R2(spb)+n]

30. K' |- M1' sim_c M2'                                           | (sim-M), 8, 9, 21, 23, 26

31. |-c S2' : K'                                                  | Coloring Preservation Lemma, 3, 24, ir = sfree

43. ((R1++[spg->R1(spg)+n][spb->R1(spb)+n],C,M1',Q1,.)           | (sim-S), 25, 31, 29, 30, 6
    sim_c
    (R2++[spg->R2(spg)+n][spb->R2(spb)+n],C,M2',Q2,.)
*





CASE BZ_G:
~~~~~~~~~~

By Inspection of S1 -->_0 S1', one of the four bz_G rules may apply.
By Preservation, (bz-untaken-fail) and (bz_G-taken-fail) do not apply.

subcase A: S1 steps using (bz-untaken)
20a. (R1,C,M1,Q1,bz_G rz, r) -->_0 (R1++,C,M1,Q1,.)
21a. R1(d) = 0
22a. R1(rz) =/= 0


subcase B: S1 steps using (bz_G-taken)
20b. (R1,C,M1,Q1,bz_G rz, r) -->_0 (R1++[d->R1(rd),C,M1,Q1,.)
21b. R1(d) = 0
22b. R1(rz) = 0


23. S(G)(d) = <G,int,0>                                           | Inversion of (bz_G-t), 12
24. S(G)(rz) = <G,int,Ez>
25. S(G)(rd) = <G,T->void,Ed'>

26. K(d) = K(rz) = K(rd) = G                                      | K Lemma, (Inversion of (S-t), 2), 23, 24, 25

27. a: |- (R1++,C,M1,Q1,.) : K [ d -> G ]                         | Coloring Preservation Lemma, 2, 20a/20b, ir = bz_G
    b: |- (R1++[d->R1(rd)],C,M1,Q1,.) : K [ d -> G]

28. K[ d -> G ] |- M1 sim_c M2                                    | Coloring Update Lemma, 5

SUBCASE 1: c = B  --  S2 must step using the same rule

271. Q1 = Q2                                                      | 6, def of (sim-Q)
281. R1(d) = R2(d)                                                | Colored Pairs Lemma, 26, assumption, 7
291. R1(rz) = R2(rz) and R1(rd) = R2(rd)                          | Colored Pairs Lemma, 26, assumption, 7

301. a: R2(d) = 0 and R2(rz) =/= 0                                | 281, 291, (21a,22a) / (21b,22b)
     b: R2(d) = 0 and R2(rz) = 0

311. a: S2 -->_0 (R2++,C,M2,Q2,.)                                 | (bz-untaken), 301a
     b: S2 -->_0 (R2++[d->R2(rd),C,M2,Q2,.)                       | (bz-taken), 301b


331. a: |-c (R2++,C,M2,Q2,.) : K [ d -> G ]                       | Coloring Preservation Lemma, 3, 311, ir = bz_G
     b: |-c (R2++[d->R2(rd),C,M12,Q2,.) : K [ d -> G]


341. a: K[ d -> G ] |-  R1++ sim_c R2++                           | 13, 26
     b: K[ d -> G ] |-  R1++[d->R1(rd)] sim_c R3++[d->R2(rd)]     | 13, 26, 7

361. a: S1' sim_c (R2++,C,M2,Q2,.)                                | (sim-S), 27, 331, 341, 28, 6
     b: S1' sim_c (R2++[d->R2(rd),C,M12,Q2,.)

371. S2 -->_0 S2'  and  S1' sim_c S2'  and () = ()                | 311, 361


SUBCASE 2: c = G  --  S2 may step using any of the 4 applicable rules

272. S2 -->_0 S2' where S2' is either
     c: (R2++,C,M2,Q2,.)                                          | (bz-untaken)
```

```
     d: fault                                                      | (bz-untaken-fail)
     e: (R2++[d->R2(rd),C,M12,Q2,.)                                | (bz_G-taken)
     f: fault                                                      | (bz_G-taken-fail)

282. for all possible pairings                                     | (sim-R), (sim-val-zap), 13,
     K[ d-> G ] |-
     {{ R1++, R1++[d-> R1(rd)] }}                                      a/b: possible values for S1'
     sim_G
     {{ R2++, R2++[d-> R2(rd)] }}                                      c/e: possible values for S2'

292. if S2 -->_0 (R2',C,M2',Q2',.) then                            | c/e: Coloring Preservation Lemma, 3, 272, ir = bz_G
     |-c (R2',C,M2',Q2',.) : K[d -> G]

262. if S2 -->_0 (R2',C,M2',Q2',.) then                            | a/b and c/e, (sim-S), 27, 292, 272, 28, 6
     S1' sim_G (R2',C,M2',Q2',.)

26d. either
     S2 -->_0 S2'  and  S1' sim_G S2' and () = ()                  | a/b and c/e
     or
     S2 -->_0^() fault                                             | a/b and d/f
*








CASE BRZ_B:
~~~~~~~~~~~
By Inspection of S1 -->_0 S1', one of the four bz_B rules may apply.
By Preservation, (bz-untaken-fail) and (bz_B-taken-fail) do not apply.


20. S(G)(d) = E=0 => <G,b,Er'>                                             | Inversion of (bz_B-t), 12
21. S(G)(rz) = <B,int,Ez>
22. S(G)(rd) = <B,T->void,Er>

23. K(d) = G                                                               | K Lemma, (Inversion of (S-t), 2), 20, 21, 22
24. K(rz) = K(rd) = B

25. |- S1' : K[ d -> G]                                                    | Coloring Preservation Lemma, 2, a2, ir = bz_B
26. K[d -> G] |- M1 sim_C M2                                               | Coloring Update Lemma, 5


subcase A: S1 steps using (bz-untaken)

25a. (R1,C,M1,Q1,bz_B rz, r) -->_0 (R1++,C,M1,Q1,.)
26a. R1(d) = 0
27a. R1(rz) =/= 0

subsubcase A1: ( R2(d) = 0 and  R2(rz) =/= 0 )  --  S2 also steps with bz-untaken
29a1. S2 -->_0( R1++,C,M1,Q1,.)                                            | (bz_B-untaken), assumption
30a1. |-c ( R2++,C,M2,Q2,.) : K[ d -> G]                                   | Coloring Preservation Lemma, 3, 29a1, ir = bz_B
31a1. ( R1++,C,M1,Q1,.) sim_c ( R1++,C,M1,Q1,.)                            | (sim-S), 25, 30a1, 13, 26, 6
32a2. S2 -->_0 S2'  and  S1' sim_c S2' and () = ()                         | 29a1, 31a1

subsubcase A2: ( R2(d) =/= 0 or  R2(rz) = 0 )   -- S2 steps with a bz-fail
29a2. c = B ==> R1(d) = R2(d)                                              | Colored Pairs Lemma, 7, 23, 24
      c = G ==> R1(rz) = R2(rz)

30a2. c = B ==> R2(d) = 0                                                  | 29a2, 26a, 27a
      c = G ==> R2(rz) =/= 0

31a2. c = B ==> R2(d) = 0 and R2(rz) = 0                                   | 30a2, assumption
      c = G ==> R2(rz) =/= 0  and R2(d) =/=0

32a2. c = B ==>  S2 -->_0^() fault                                         | (bz_G-taken-fail), 31a2
      c = G ==>  S2 -->_0^() fault                                         | (bz-untaken-fail), 21a2

33a2. S2 -->_0^() fault                                                    | 32a2



subcase B: S1 steps using (bz_B-taken)

25b. (R1,C,M1,Q1,bz_G rz, r)
     -->_0 (R1++[pcg -> R1(rd)][pcb -> R1(rd)][d->0],C,M1,Q1,.)
26b. R1(d) =/= 0
27b. R1(rz) = 0
28b. R1(rd) = R1(d)
```

```
subsubcase B1: ( R2(d) =/= 0  and  R2(rz) = 0  and R2(rd) = R2(d) )
29b1. S2 -->_0 (R2++[pcg -> R2(rd)][pcb -> R2(rd)][d->0],C,M2,Q2,.)     | (bz_B-taken), assumptions
30b1. |-c (R2++[pcg->R2(rd)][pcb -> R2(rd)][d->0],C,M2,Q2,.):K[d->G]    | Coloring Preservation Lemma, 3, 29b1, ir = bz_B

31b1. c = G ==> R1(rd) = R2(rd)                                         | Colored Pairs Lemma, 7, 23, 24
      c = B ==> R1(d) = R2(d)

32b1. c = G ==> R1(rd) = R2(rd)                                         | 31b1, 28b, assumption
      c = B ==> R1(rd) = R2(rd)

33b1. all k. k R1(rd) sim_c k R2(rd)

34b1. K[d->G] |-                                                        | (sim-R), 13, 33b1
      R1++[pcg->R1(rd)][pcb -> R1(rd)][d->0]
      sim_c
      R2++[pcg->R2(rd)][pcb -> R2(rd)][d->0]

35b1. S1' sim_c (R2++[pcg -> R2(rd)][pcb -> R2(rd)][d->0],C,M2,Q2,.)    | (sim-S), 25, 30b1, 34b1, 26, 6
32a2. S2 -->_0 S2'  and  S1' sim_c S2' and () = ()                      | 29b1, 35b1


subsubcase B2: R2(d) = 0  or  R2(rz) =/= 0  or  R2(rd) =/= R2(d)
29b2. c = G ==> R1(rz) = R2(rz)  and  R1(rd) = R2(rd)                   | Colored Pairs Lemma, 7, 23, 24
      c = B ==> R1(d) = R2(d)

30b2. c = G ==> R2(rz) = 0  and R2(rd) =/= 0                            | 29b2, 26b, 27b, 28b
      c = B ==> R2(d) =/= 0

31b2. c = G ==> R2(rz) = 0 and (R2(d) = 0 or R2(rd) =/= R2(d))          | 30b2, assumptions
      c = B ==> R2(d) =/= 0 and (R2(rz) =/= 0 or R2(rd) =/= R2(d))

32b2. c = G ==> R2(rz) = 0 and (R2(d) = 0 or R2(rd) =/= R2(d))          | 31b2
      c = B ==> either R2(d)=/=0 and R2(rz)=/=0
                or     R2(d)=/=0 and R2(rz) = 0 and R2(rd) =/= R2(d)

33b2. c = G ==> S2 -->_0^() fault                                       | (bz_B-taken-fail), 32b2
      c = B ==> either  S2 -->_0^() fault                               | (bz_B-untaken-fail), 32b2
                or      S2 -->_0^() fault                               | (bz_B-taken-fail), 32b2

34b2. S2 -->_0^() fault                                                 | 33b2
*




CASE JMP_G:
~~~~~~~~~~~

similar to BZ_G
*



CASE JMP_B:
~~~~~~~~~~~

similar to BZ_B
*
```

Fault Tolerance Theorem

```
Fault Tolerance Theorem:
*************************

If  |- S and  S -n->_0^ss S'   (and let l = max(Dom(C)))
then  either  Exists m <= n+1.  S -m->_1^ssf fault  and  ssf prefix(l) ss
          or  S -(n+1)->_1^ssf Sf and Exists c. S sim_c Sf and ss ~l~ ssf


Proof: By case analysis on the definition of S -n->_k^ss S'


CASE: multi-single

--------------(multi-single)
S -0->_0^() S


a1. |- S
a2. S -0->_0^() S'


1. S -->_1 Sf                                          | (reg-zap), (Q1-zap), or (Q2-zap)
2. Exists c. S sim_c Sf                                | Fault Similarity Lemma, 1
3. () ~l~ ()                                           | (out-eq-none)
4. S -1->_1^() Sf                                      | 1, 2, 3
   and Exists c. S sim_c Sf and () ~l~ ()
*


CASE: multi-compose
(p1) S -->_0^s1 S''    (p2) S'' -(n-1)->_0^ss2  S'    (s1) ss = (s1,ss2)
--------------------------------------------------------------------------(multi-compose)
S -n->_0^(ss) S'


a1. |- S
a2. S -n->_0^ss S'


1. n = n1 + n2                                         | Multistep Split Lemma, a2
2. S -n1->_0^ssa Sn1
3. Sn1 -n2->_0^ssb S'
4. ss = (ssa,ssb)


5. Sn1 -->_1^() Sn1f                                   | (reg-zap), (Q1-zap), or (Q2-zap)
6. Exists c. Sn1 sim_c Sn1f                            | Fault Similarity Lemma, 5

7. either                                              | Lemma Multistep-Fault-Detection, 3, 6
    a: Sn1f -n2->_0^ssbf Sf'  and  S' sim_c Sf'
       and ssbf ~l~ ssb
    or
    b: Exists m2<=n2.  Sn1f -m2->_0^ssbf fault
       and  ssbf prefix(l) ssb


SUBCASE: fault not reached yet

a3. Sn1f -n2->_0^ssbf Sf'
a4. S' sim_c Sf'
a5. ssbf ~l~ ssb

8. let ssf = (ssa,ssbf)
9. S -(n+1)->_1^ssf Sf'                                | Multistep Combine Lemma, 2, 5, a3, 8
10. ss ~l~ ssf                                         | repeated applications of (out-eq-cons), a5, 8, 4

11. S -(n+1)->_1^ssf Sf' and S' sim_c Sf'              |9, A4, 10
```

```
    and ss ~l~ ssf
*


SUBCASE: fault reached
a3. Exists m2 <= n2
a4. Sn1f -m2->_0^ssbf fault
a5. ssbf prefix(l) ssb

8. let m = n1 + 1 + m2
9. m <= n+1                                          | 1, a3, 8
10. let ssf = (ssa,ssbf)
11. S -m->_1^ssf  fault                              | Multistep Combine Lemma, 2, 5, a4
12. ssf prefix(l) ss                                 | repeated applications of (out-prefix-consume), 10, a5, 4

13. m <= n+1 and S -m->_1^ssf fault                  | 9, 11, 12
    and ssf prefix of ss
*
```

MiniC Syntax and Typing

```
types        tau     ::=     int  |  tau ref  |  A -> tau
var contxt  X       ::=     .  |  x:tau, X             // specific contexts: F functions, A arguments, L local variables

values      v       ::=     n  |  x  |  ref v
value list  vs      ::=     .  |  v, vs


statement   s       ::=     x = v
                            |  x = v op v
                            |  x = !v
                            |  v := v
                            |  if v then s else s
                            |  while v do s
                            |  x = x(vs)
                            |  s; s


funct decls  fds    ::=     .  |  tau f(A) {lds; s; return v;}  fds

local decls  lds    ::=     .  |  tau x = v; lds


program     p       ::=     fds; lds; s;
```

```
 --------------
| X |- v : tau |
 --------------
                                                         X |- v : tau
 ------------ (n-t)         --------------- (var-t)     --------------------- (ref-t)
 X |- n : int                X |- x : X(x)              X |- ref v : tau ref
```

```
 --------------
| X |- vs : ps |
 --------------
                            X |- v : tau   X |- vs : ps
------------ (vs-.-t)       ------------------------------ (vs-t)
X |- . : .                  X |- v, vs : x:tau, ps
```

```
 ---------------
| F;A;L |- s ok |
 ---------------

L |- x : tau    (A union L) |- v : tau
----------------------------------------(s-assign-t)
F;A;L |- x = v ok


L |- x : int    (A union L) |- v1 : int   (A union L) |- v2 : int
------------------------------------------------------------------ (s-op-t)
F;A;L |- x = v1 op v2 ok
```

```
(A union L) |- v : tau ref    L |- x : tau
----------------------------------------------- (s-deref-t)
F;A;L |- x = !v ok


(A union L) |- v1 : tau ref   (A union L) |- v2 : tau
------------------------------------------------------- (s-update-t)
F;A;L |- v1 := v2 ok


(A union L) |- v : int    F;A;L |- s1 ok    F;A;L |- s2 ok
------------------------------------------------------------- (s-if-t)
F;A;L |- if v then s1 else s2 ok


(A union L) |- v : int    F;A;L |- s ok
----------------------------------------- (s-while-ok)
F;A;L |- while v do s ok


F |- f : ps -> tau    (A union L) |- vs : ps    L |- x : tau
--------------------------------------------------------------- (s-call-t)
F;A;L |- x = f(vs) ok


F;A;L |- s1 ok    F;A;L |- s2 ok
--------------------------------- (s-seq-ok)
F;A;L |- s1;s2 ok





 --------------------
| F;A;L |- lds : L'  |
 --------------------

------------------- (lds-.-t)
F;A;L |- . : L

x notin Dom(F union A union L)
(A union L) |- v : tau
F;A;L[x:tau] |- lds : L'
------------------------------------- (lds-val-t)
F;A;L |- tau x = v; lds : L'




 ---------------
| F |- fds : F' |
 ---------------

----------- (fds-.-t)
F |- . : F


f notint Dom(F)
F;A;. |- lds : L    F;A;L |- s ok    (A union L) |- v : tau
F[f : A -> tau] |- fds : F'
----------------------------------------------- (lds-val-t)
F |- tau x(A) {lds; s; return v;} fds : F'




 ---------
| |- p ok |
 ---------

    . |- fds : F
  F;. |- lds : L
F;.;L |- s ok
    L |- v : int
-------------------------- (p-ok)
|- fds; lds; s; return v;
```

Translation from MiniC to ETAL_FT

```
REGISTERS
*********

t0 ... tt    // temporaries. number required calculated during translation
pcg pcb      // program counters
spg spb      //stack pointers



CALLING CONVENTION
*******************

Many are possible, but here's one:

-- function entry: load args into registers
-- before call: spill all temps corresponding to lds, push args vnB vnG ... v0B v0G, push ret address raB raG
-- after call: restore result, restore all lds into temps
-- function exit: pop args, push return values



NOTATION
********

tcount                                                        // number of temp registers -- always even. even are green, odd are blue

(tcount',r1,r2) = freshReg(tcount)                            // generate 2 new temp registers and return new tcount and registers
(tcount',r1,...,rn) = freshRegs(tcount)                       // multiple applications of freshReg

V   ::=  .  |   x -> (r,r), V                                 // maps source level variables to assembly level registers
Vg(x) refers to first component
Vb(x) refers to second

C @ i  =def=  let l = max(dom(C)) in  C[l+1 -> i]             // appending an instruction to code memory

B   ::=  .  |  x -> n                                         // contains the addresses for the function entries



DEFINITIONS
***********
let ha be some address in memory



TYPE TRANSLATIONS
*****************


 ----------------------------------
| [[ tau ]] = b                     |                         // type translation
 ----------------------------------


------------------------------ (trans-int)
[[ int ]] = int



[[ tau ]] = b
------------------------------------------ (trans-ref)     // note ref is initialized
[[ tau ref ]] = b ref1
```

```
[[ A ]] = Es:<G,b1,aa1> ::...:: al-1:<B,bn,aan> :: al:as    // translate arguments into a stack type
[[ tau ]] = bt                                              // translate return type

amem, apc, at, amemr, at fresh

Dr = amemr : kmem, at:kint                                  //      fresh vars for return
Gr = spg -> <G,sptr,al-2>, spb -> <B,sptr,al-2>             //      stack pointer to top of stack
     pcg -> <G,int,ar>, pcb -> <G,int,ar>                   //      program counters are return address
     d -> <G,int,0>                                         //      no jumps in progress
E_mr = amemr                                                //      memory contents may change during call
sr = al-2 : <G,bt,at> :: al -1 : <B,bt,at> :: al : as       //      stack has popped args and ret addr and pushed ret val
Tr = (Dr, Gr, (), E_mr, sr)                                 //

D = (amem:kmem, apc: kint, al:kint, as:kstack, aa1:kint, ..., aan:kint)
G = spg -> <G,sptr,Es-2>, spb -> <B,sptr,Es-2>,
     pcg -> <G,int,apc>, pcb -> <G,int,apc>, d -> <G,int,0>
E_m = amem
s = Es-2 : <G,Tr->void,ar> :: Es-1 : <B,Tr->void,ar>        // add return addresses to stack
    :: Es:<G,t1,aa1> ::...:: al-1:<B,tn,aan> :: al:as
--------------------------------------------------------------- (trans-functtp)
[[ A -> tau ]] =  (D, G,(),E_m, s) -> void




 -----------------------
| [[ A ]] = s          |                                    // translating parameters into stack type
 -----------------------

al,as fresh
---------------------------------------------------------- (gen-s-empty)
[[ . ]]   = al:as


[[ tau ]] = bt
[[ A ]]   = Es:us
at fresh
----------------------------------------------------------------- (gen-s)
[[ x:tau, A ]] = Es-2 : <G,bt,at> :: Es-1 : <B,bt,at> :: Es : us




 ----------------------------------      // used inside a function to get registers corresponding to arguments
| [[ A,s ]]_V = G                   |
 ----------------------------------


---------------------------------------------------------- (gen-G-args-empty)
[[ ., al:as ]]_V   = .


[[ A, Es:us ]]_V = G
G' = G, Vg(x) -> <G,bt,at>, Vb(x) -> <B,bt,at>
----------------------------------------------------------------- (gen-G-args)
[[ x:tau,A,  Es-2 : <G,bt,at> :: Es-1 : <B,bt,at> :: Es : us ]]_V = G'




 ---------------
| [[ X ]]_V = G |                       // generate types for registers corresponding to source variables (local or arguments)
 ---------------

-------------------
[[ . ]]_V = .


[[ L ]]_V = GL
[[ tau ]] = bt
at fresh
G' =  GL, Vg(x) -> <G,bt,at>, Vb(x) -> <B,bt,at>
---------------------------------------------------------- (gen-G-lds)
[[ x:tau, L ]]_V = G'
```

```
 ------------------------------------------------
| [[ A -> tau; L ]]_V = (D,G,seq(E_d,E_s),E_m,s) |           // generates Theta for a points in a function between statements
 ------------------------------------------------            //


[[A -> tau]] = (D, G,(),E_m, s) -> void                     // type at function entry
G  = G', [[ L ]]_V, [[ A ]]_V                               // add on the local variable info and argument info
--------------------------------------- (gen-Theta)
[[ A->tau; L ]]_V = (D',G,(),E_m',s')




VALUE TRANSLATIONS
******************


 ----------------------------------------------------
| [[ X |- v : tau]] C tcount V = C' tcount' rg rb    |             // move v into fresh temporary registers rg and rb
 ----------------------------------------------------

(tcount',rg, rb) = freshReg(tcount)
C' = C @ mov rg n @ mov rb n
------------------------------------------------------ (trans-n)
[[ X |- n : int]] C tcount V = C' tcount' rg rb



--------------------------------------------------------- (trans-v)
[[ X |- x : X(x) ]] C tcount V =  C  tcount  Vg(x) Vb(x)



[[ X |- v : tau ]] C tcount V = Cv tcountv rgv rbv
[[tau]] = bt
(tcount',rga,rba) = freshReg(tcountv)
C' =  Cv @ malloc[ bt ] rga rba;
         @ st_G rga rgv
         @ st_B rba rbv
----------------------------------------------------------- (trans-ref)
[[ X |- ref v : tau ref ]] C tcount V = C' tcount' rga rgb





 ------------------------------------------------
| [[ X |- vs : ps ]] C tcount V Pnum = C' tcount' |         // generates code to push arguments on to stack, assumes allocation done
 ------------------------------------------------           // Pnum is an integer representing the arg number of the 1st  in list


------------------------------------------------ (trans-vs-.)
[[ X |- . : . ]] C tcount V Pnum = C tcount



[[ X |- v : tau ]] C tcount V = Cv tcountv rvg rvb
Cv' = Cv @ sst (Pnum*2) rvg; sst (Pnum*2+1) rvb
[[ X |- vs : ps ]] Cv' tcountv V (Pnum+1) = Cvs tcountvs
------------------------------------------------------------- (vs-t)
[[ X |- v, vs : x:tau, ps ]] C tcount V Pnum = Cvs tcountvs


if calling f(v0,...,vn), layout is first arg lowest on stack, each green lower than corresponding blue

|  .  |
|__.__|
| vnb |
| vng |
|  .  |
|  .  |
|  .  |
| v0b |
| v0g |  <-- spg, spb
```

PROLOGUE / EPILOGUE GENERATION
****

```
 ----------------------------------------------------
| [[ A]]_prologue C tcount V Pnum = C' tcount' V'    |             // generates function prologue from argument list
 ----------------------------------------------------

-------------------------------------------- (trans-prologue-empty)
[[ A]]_prologue C tcount V Pnum = C tcount V


(tcountx,rg, rb) = freshRegs(tcount)
Cx = C @ sld_G (Pnum*2+2) rg                                    // load arg into registers
       @ sld_B (Pnum*2+3) rb
[[ A]]_prologue Cx tcountx (Pnum+1) V[x->(rg,rb)] = C' tcount' V'  // extend V with the new registers
------------------------------------------------------------------ (trans-prologue)
[[(x:tau,A)]_prologue C tcount V Pnum = C' tcount' V'
```

```
 ----------------------------------------------------------------------
| [[ (A union L) |- v : tau]]_epilogue Cs tcounts Vlds ps = C' tcount' | // generates function epilogue from argument list
 ----------------------------------------------------------------------

[[ (A union L) |- v : tau]] Cs tounts Vlds = Cv tcountv rvg rvb
(tcount', rag, rab) = freshRegs(tcountv)
argspace = sizeof(ps)*2 + 2
C' = C @ sld_G 0 rag    @ sld_B 1 rab
       @ sfree argspace @ salloc 2
       @ sst 0 rgv      @ sst 1 rbv
       @ jmp_G rag      @ jmp_B rab
-------------------------------------------------------------------
[[ (A union L) |- v : tau]]_epilogue Cs tcounts Vlds = C' tcount'
```

STATEMENT TRANSLATIONS
****

```
 ------------------------------------------------
| [[ F;A;L |- s ok ]]_f C tcount V B = C' tcount' | // translating statement s in function f
 ------------------------------------------------


[[ (A union L) |- v : tau ]] C tcount V = Cv tcountv rgv rbv
C' = Cv @ mov Vg(x) rgv
        @ mov Vb(x) rbv
-----------------------------------------------------------(trans-s-assign)
[[ F;A;L |- x = v ok ]]_f C tcount V B = C' tcountv



[[ (A union L) |- v1 : int ]] C  tcount  V  = C1 tcount1 rg1 rb1
[[ (A union L) |- v2 : int ]] C1 tcount1 V  = C2 tcount2 rg2 rb2
C' = C2 @ op Vg(x) rg1 rg2
        @ op Vb(x) rb1 rb2
------------------------------------------------------------- (trans-s-op)
[[ F;A;L |- x = v1 op v2 ok ]]_f C tcount V B = C' tcount2



[[ (A union L) |- v : tau ref ]] C tcount V = Cv tcountv rg rb
C' = Cv @ ld_G Vg(x) rg @ ld_B Vb(x) rb
```

```
---------------------------------------------------------------- (trans-s-deref)
[[ F;A;L |- x = !v ok ]]_f C tcount V B = C' tcountv



[[ (A union L) |- v1 : tau ref ]] C  tcount  V  = C1 tcount1 rg1 rb1
[[ (A union L) |- v2 : tau ]]     C1 tcount1 V  = C2 tcount2 rg2 rb2
C' = C2 @ st_G rg1 rg2 @ st_B rb1 rb2
---------------------------------------------------------------- (trans-s-update)
[[ F;A;L |- v1 := v2 ok ]]_f C tcount V B = C' tcount2




lc = max(dom(C)) + 1
[[ A union L) |- v : int ]] C tcount V  = Cv tcountv rgv rbv    // get the conditional registers

(tcountc, rtg, rfg, rjtg, rjfg, rtb, rfb, rjtb, rjfb, reg, reb)
 = freshReg(tcountv) // fresh registers for the block labels

lfixtrue = max(Dom(Cv)+1)                                       // finish of the first block beginning at lfixme
Cc = Cv @ mov rtg 1 @ mov rtb 1                                 // temporarily set to branch to ending code
        @ bz_G rgv rtg @ bz_B rbv rtb

[[ F;A;L |- s2 ok ]]_f Cc tcountc V = Cf tcountf                // translate false (fallthru) block
lfixjoin = max(Dom(Cf)+1)
Cf' = Cf @ mov rjfg 1 @ mov rjfb 1                              // temporarily set to branch to ending code
         @ jmp_G rjfg @ jmp_B rjfg

lbztarget = max(Dom(Cf')+1)
[[ F;A;L |- s2 ok ]]_f Cf' tcountf V = Ct tcountt               // translate true branch
ljoin = max(Dom(Ct)+1)
Ct' =  Ct @ mov reg ljoin @ mov reb ljoin
              @ jmp_G reg @ jmp_B reb

C' = Ct'[lfixjoin   -> mov rjfg ljoin]                           // patch end of false to jump to join
        [lfixjoin+1 -> mov rjfb ljoin]
        [lfixtrue -> mov rtg lbztarget]                          // patch up branch target to true block
        [lfixtrue+1 -> mov rtb lbztarget]
tcount' = tcount

-------------------------------------------------------------------------------- (trans-s-if)
[[ F;A;L |- if v then s1 else s2 ok ]]_f C tcount V B = C' tcount'




lstart = max(Dom(C)+1)
[[ (A union L) |- v : int ]]_f C tcount V = Cv tcountv rgv rbv  // get the conditional registers

(tcountc, reg, reb, rsg, rsb) = freshReg(tcountv)
lfixend = max(Dom(Cv)+1)
Cc = Cv @ mov reg linfloop @ mov reb linfloop                   // temporarily set to branch to ending code
        @ bz_G rgv reg @ bz_B rbv reb

[[ F;A;L |- s ok ]]_f Cc tcountc V = Cs tcounts

Cs' = Cs @ mov rsg lstart @ mov rsb lstart
         @ jmp_G rsg      @ jmp_B rsb

lend = max(Dom(Cs')+1)
C' = Cs'[ lfixend   -> mov reg lend ]
        [ lfixend+1 -> mov reb lend ]
tcount' = tcounts
------------------------------------------------------- (trans-s-while)
[[ X |- while v do s ok ]]_f C tcount V B = C' tcount'




argspace = (size(vs) * 2) + 2
vspace = size(Dom(V)) * 2
spillspace = vspace + argspace

Ctemps = C @ salloc spillspace                // alloc space for temps, args, return address
           @ sst (argspace+0)      Vx(x1)     // spill all temps in Range of V
           @ ...
           @ sst (argspace+vspace-1) Vb(xn)

[[ (A union L) |- vs : ps ]] Ctemps tcount V 1 = Cvs tcountvs // get each arg and store onto stack
```

```
(tcountcall, rfg, rfb, rag, rab)
  = freshRegs(tcountvs)

retaddr = max(Dom(Cvs)) + 9

Ccall = Cvs @ mov rag retaddr @ sst 0 rag     // store return addresses
            @ mov rab retaddr @ sst 1 rab

            @ mov rfg B(g)    @ mov rfb B(f)  // call g
            @ jmp_G rfg       @ jmp_B rfb

            @ sld_G (2+0)      Vx(x1)         // restore temps (alternate load color)
            @ ...
            @ sld_B (2+vspace-1) Vb(xn)

            @ sld_G Vg(x) 0   @ sld_B Vb(x) 1 // x <- return values

            @ sfree (vspace+2)                // remove space for return values and spilled temps

------------------------------------------------------------------- (trans-s-call)
[[ F;A;L |- x = g(vs) ok ]]_f C tcount V B = Ccall tcountcall




[[ F;A;L |- s1 ok ]]_f C  tcount  V B = C1 tcount1
[[ F;A;L |- s2 ok ]]_f C1 tcount1 V B = C2 tcount2
---------------------------------------------------- (trans-s-seq)
[[ F;A;L |- s1;s2 ok ]]_f C tcount V B = C2 tcount2





LOCAL DECLARATION TRANSLATION
*****************************

 ----------------------------------------------------
| [[ F;A;L |- lds : L' ]] C tcount V = C' tcount' V' |
 ----------------------------------------------------


---------------------------------------------- (trans-lds-empty)
[[ F;A;L |- . : L ]] C tcount V = C tcount V


[[ (A union L) |- v : tau ]] C tcount V = Cv tcountv rvg rvb
(tcountld, rg, rb) = freshReg(tcountv)
Vld = V[x -> (rg,rb)]
Cld = Cv @ mov rg rvg @ mov rb rvb
[[ F;A;L[x:tau] |- lds : L' ]] Cld tcountld Vld = C' tcount' V'
------------------------------------------------------------------ (trans-lds)
[[ F;A;L |- tau x = v; lds : L' ]] C tcount V = C' tcount' V'






FUNCTION TRANSLATION
*********************

 ------------------------------------
| [[ F |- fds : F' ]] C B = C' B' n |       // extends C with new function, adds address of new function to B, n is max num
 ------------------------------------             of temp registers used by any one function


------------------------------------ (trans-fds-empty)
[[ F |- . : F ]] C B = C B



faddr = max(Dom(C)+1))
prologue(ps) C 0 V 0 = Cp tcountp Vp
[[ F[f : ps -> tau];ps;. |- lds : L ]] Cp tcountp Vp = Clds tcountlds Vlds
[[ F[f : ps -> tau];ps;L |- s ok ]]_f Clds tcountlds Vlds B = Cs tcounts
[[ (A union L) |- v : tau]]_epilogue Cs tcounts Vlds ps = C' tcountr
tcount = max(tcountr, tcountfds)
```

```
------------------------------------------------------------------------------- (trans-fds)
[[ F |- tau f(ps) {lds; s; return v;} fds : F']] C B = C' B' tcount
```

```
PROGRAM TRANSLATION
*******************

 ------------------------------------
| [[ |- p ok ]] = C tcount startaddr | // returns code mem and max number of temp registers used
 ------------------------------------

Cstart =  ha   -> mov t0 ha                                          // code mem starts with infinite loop
          ha+1 -> mov t1 ha
          ha+2 -> jmp_G t0
          ha+3 -> jmp_G t1

[[ . |- fds : F ]]   Cstart 0 . = Cfds Bfds tcountfds
startaddr = max(Dom(Cfds)+1)
[[ F;.;. |- lds : L ]] Cfds 0 . = Clds tcountlds Vlds
[[ F;.;L |- s ok ]]  Clds tcountlds Vlds Bfds = Cs tcounts
[[ L |- v : tau]]_epilogue Cs tcounts Vlds () = C' tcountr

tcount' = max(tcountr, tcountfds)
--------------------------------------------------------- (trans-p)
[[ |- fds; lds; s; return v ok ]] = C' tcount' startaddr
```

Translation Theorem

```
 ----------------
| B;F |- C : P   |                                   // code memory contains wf functions
 ----------------

P |- C                                               // code memory is well formed

all f in F. P(B(f)) = [[F(f)]]                       // functions starts are correct

Dom(P) = Dom(C)

P(ha) = ( ( ap:kint,am:kmem,al:kint,as:kstack ),     // code mem starts with infinite loop
         ( pcg -> <G,int,1>,   pcb -> <B,int,1>,
           spg -> <G,sptr,al>, spb -> <B,sptr,al>,
           d   -> <G,int,0> ),
         (), am, al:as )
------------------------------------------ (wf-F)
B;F |- C : P




 ---------------------
| T <= T'             |     // current state T a subtype of another
 ---------------------

Exists S. D' |- S : D
S(G') <= G
D |- S(Em') = Em
D |- S(s) = s
----------------------------------- (T-subtp)
(D',G',(),Em',s') <= (D,G,(),Em,s)



 --------------
|  P |- C : T  |                              // code memory is well formed, and the next location
will have type T
 --------------

0 not in Dom(C)
lm = max(Dom(C))
Dom(P) = Dom(C) union (lm+1)
all l in Dom(C).
    P(l) = T -> void
    and P;T |- C(l) => RT
    and (RT = T' ==> P(l+1) = T' -> void)
P(lm+1) = T -> void
------------------------------------------------ (C-partial-t)
P |- C : T
```

```
 -------------
| X;V |- T wf |
 -------------

T = (D,G,seq,Em,s)
all x:tau in X.
 V(x) = (rg,rb)
 G(rg) = <G,[[tau]],Eg>
 G(rb) = <B,[[tau]],Eb>
 D |- Eg = Eb
------------------------ (T-V-wf)
X;V |- T wf
```

```
 ----------------------
| f;V;B;F;A;L |- C : P |                          // code memory contains wf functions and one
function f in progress that is
 ----------------------                                 consistent with [[A;L]]_V

P |- C : T
(A union L);V |- T wf
T <= [[ F(f);L ]]_V                               // translation in progress is in a constistent
state
all f' in F. P(B(f)) = [[F(f')]]                  // functions starts are correct
P(ha) = ( ( ap:kint,am:kmem,al:kint,as:kstack ),
         ( pcg -> <G,int,1>,   pcb -> <B,int,1>,
           spg -> <G,sptr,al>, spb -> <B,sptr,al>,
           d   -> <G,int,0> ),
         (), am, al:as )
----------------------------------------------------- (wf-s)
f;V;B;F;A;L |- C : P
```

```
Code Addition Lemma
********************

If P |- C : T1
   P; T1 |- i1 : T2
   P; T2 |- i2 : T3
   ...
   P; Tn |- in : T'

then P' |- C @ i1 @ ... @ in : T'
where P' = P,(max(Dom(P)+1) -> (T2->void), (lm+3) -> (T3->void), ..., (max(Dom(P)+n) -> (T'->void)

Proof: by inversion/reconstruction of (C-partial-t)
*
```

```
Block Extension Lemma
**********************
If P |- C
and P;T1 |- i1 : T2
```

```
and P;T2 |- i2 : T3
and ...
and P;Tn |- in : T'

then P' |- C @ i1 @ ... @ in : T'
where P' = P, (max(Dom(P)+1) -> (T1->void), (max(Dom(P)+n) -> (Tn->void), (max(Dom(P)+n+1) -> (T'-
>void)

Proof: similar to Code Addition Lemma. deconstruction (C-t) to build first (C-partial-t)



Value Translation Lemma
***********************

If [[ X |- v : tau]] C tcount V = C' tcount' rg rb
and P |- C : (D,G,seq,Em,s)
and X;V |- T wf
then
Exists Em', P'. P' |- C' : (D,G[rg -> <G,[[tau]],Eg>][rb -> <B,[[tau]],Eb>],seq,Em',s)
and  D |- Eg = Eb

Proof: By case analysis on [[ X |- v : tau]] C tcount V = C' tcount' rg rb

case (trans-n): immediate from (mov-t)

case (trans-v): immediate from (mov-t) and inversion of (T-V-wf)

case (trans-ref):
  let P' be P extended with the new location returned by malloc
  P' |- C : T by inversion/reconstruction of (C-partial-t) and Heap Extension Lemma
  then use (malloc-t), (st_G-t), and (st_B-t)
*




Local Decl Translation Lemma
*****************************

If  [[ F;A;L |- lds : L' ]] C n V = C' n' V'
and f;V;B;F;A;L |- C : P
then Exists P'. f;V';B;F;A;L' |- C': P'


Proof: by case analysis on [[ F;A;L |- lds : L' ]] C n V = C' n' V'


case (trans-lds-empty): immediate


case (trans-lds):

p1. [[ (A union L) |- v : tau ]] C R V = Cv Rv rvg rvb
p2. (tcountld, rg, rb) = freshReg(tcount)
p3. Vld = V[x -> (rg,rb)]
p4. Cld = Cv @ mov rg rvg @ mov rb rvb
p5. [[ F;A;L[x:tau] |- lds : L' ]] Cld tcountld Vld = C' tcount' V'
------------------------------------------------------------------ (trans-lds)
[[ F;A;L |- tau x = v; lds : L' ]] C R V = C' tcount' V'
```

```
1. P |- C : T                                       | Inversion of (wf-s), a2
2. (A union L);V |- T wf
3. T <= [[ F(f);L ]]_V
4. all f' in F. P(B(f)) = [[F(f')]] and P(ha) = Thalt

5. T = (D,G,(),Em,s)                                | 2, inspection of (T-subtp)
6. Exists P', Em'                                   | Value Trans Lemma, p1, 1, 2, 5
   P' |- Cv : (D,G[rvg -> <G,[[tau]],Eg>]
                  [rvb -> <b,[[tau]],Eb>],
               (),Em',s)
   and D |- Eg = Eb

d1. T2 = (D, G[rvg -> <G,[[tau]],Eg>][rvb -> <b,[[tau]],Eb>]
             [rg  -> <G,[[tau]],Eg>][rb  -> <b,[[tau]],Eb>],
          (),Em',s)
7. P' |- Cv @ mov rg rvg @ mov rb rvb : T2           | Code addition Lemma, 6, (mov-t)
8. (A union L[x:tau]);(V,x->(rg,rb)) |- T2 wf        | Inversion/reconstruction of (T-V-wf), 2, 5,
d1
9. T2 <= [[ F(f);L ]]_(V,x->(rg,rb))                 | Inversion/reconstruction of (T-subtp), 3,
(gen-Theta)
10. f;(V,x->(rg,rb));B;F;A;L[x:tau] |- Cv : P'       | (wf-s), 7, 8, 9, 4

11. Exists P''. f;V';B;F;A;L' |- C' : P''            | I.H. p5, 10
*
```

```
Argument Translation Lemma
**************************

If [[ X |- vs : ps ]] C tcount V Pnum = C' tcount'
and vs = x1:tau1, ..., xn:taun
and P |- C : (D,G,seq,Em,s)
and s = El:t :: ... :: El+2*Pnum : t1 :: ... :: El+2*(Pnum+n-1) : tn :: s''
then P |- C' : (D,G',seq,Em',s')
where s' = El:t :: ... :: El+2*Pnum : <G,[[tau1]],E1> :: ... :: El+2*(Pnum+n-1) : <B,[[taun]],En>::
s''
and G' <= G

Proof: by case analysis of [[ X |- vs : ps ]] C tcount V Pnum = C' tcount'
 case trans-vs-.: trivial
 case trans-vs: uses Value Translation Lemma, Code Addition Lemma,  and (sst-t)
                G' <= G because only new temporary registers are modified
```

```
Prologue Lemma
****************

If  prologue(ps) C tcount V Pnum = C' tcount' V'
and F;B |- C
then Exists P'. f;V';B;F[f: ps->tau];ps;. |- C' : P'

Proof: by induction on  prologue(ps) C tcount V Pnum = C' tcount' V'
```

```
case (trans-prolog-empty): trivial

case (trans-prologue):
deconstruct (wf-f), use (gen-Theta) to get evidence to construct Exists Pp |- Cp : [[A->tau;.]]_Vp
then build  (wf-s)




Epilogue Lemma
****************

If [[ (A union L) |- v : tau]]_epilogue C tcount V = C' V' tcount'
and f;V;B;F;A;L |- C : P
then  Exists P'. B;F|- C : P'

Proof:
deconstruct (wf-s), show how additional code modifies s
now have type that satifies ret addr




Statement Translation Lemma
****************************

If  [[ F;A;L |- s ok ]]_f C tcount V B =  C' tcount'
    and f;V;B;F;A;L |- C : P
then Exists P'. f;V;B;F;A;L|- C' : P'

Proof: by case analysis on [[ F;A;L |- s ok ]]_f C tcount V B =  C' tcount'

1. P |- C : T                                         | Inversion of (wf-s), a2
2. (A union L);V |- T wf
3. T <= [[ F(f);L ]]_V
4. all f' in F. P(B(f')) = [[F(f')]] and P(ha) = Thalt

5. T = (D,G,(),Em,s)                                  | Inspection of (T-subtp), 3




CASE TRANS-S-ASSIGN:
~~~~~~~~~~~~~~~~~~~~

p1. [[ (A union L) |- v : tau ]] C tcount V = Cv tcountv rgv rbv
p2. C' = Cv @ mov Vg(x) rgv
            @ mov Vb(x) rbv
------------------------------------------------------------(s-assign-t)
[[ F;A;L |- x = v ok ]]_f C tcount V B = C' tcountv

6. Exists P',Em'.                                  | Value Trans Lemma, p1, 1, 5, 2
   P' |- Cv : (D,G[rvg -> <G,[[tau]],Eg>]
                  [rvb -> <b,[[tau]],Eb>],(),Em',s)
   and D |- Eg = Eb

d1. T2 = (D, G[rvg  -> <G,[[tau]],Eg>][rvb  -> <b,[[tau]],Eb>]
              [Vg(x)-> <G,[[tau]],Eg>][Vb(x)-> <b,[[tau]],Eb>],
           (),Em',s)
7. P' |- Cv @ mov rg rvg @ mov rb rvb : T2         | Code addition Lemma, 6, (mov-reg-t)
8. (A union L);V |- T2 wf                          | Inversion/reconstruction of (T-V-wf), 2, d1
```

```
9. T2 <= [[ F(f);L ]]_V                               | Inversion/reconstruction of (T-subtp), 3, d1
10. f;V;B;F;A;L|- C' : P'                             | (wf-s), 7, 9, 9, 4, p2
*
```

```
CASE TRANS-S-OP, TRANS-S-DEREF, TRANS-S-UPDATE:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

similar to trans-s-assign
apply appropriate typing rules to get ending Gamma
then need to show it's a subtype of G from [[ F(f);L ]]_V
all new temps can be dropped
two computations do equiv things, so can build new substitution using same variables for modified
registers
*
```

```
CASE TRANS-S-SEQ:
~~~~~~~~~~~~~~~~~
immediate from IH.
```

```
CASE TRANS-S-CALL:
~~~~~~~~~~~~~~~~~~

d1. argspace = (size(vs) * 2) + 2
d2. vspace = size(Dom(V)) * 2
d3. spillspace = vspace + argspace

d4. Ctemps = C @ salloc spillspace
           @ sst (argspace+0)      Vx(x1)
           @ ...
           @ sst (argspace+vspace-1)

p1. [[ (A union L) |- vs : ps ]] Ctemps tcount V 1 = Cvs tcountvs // get each arg and store onto
stack

p2. (tcountcall, rfg, rfb, rag, rab)
  = freshRegs(tcountvs)

d5. retaddr = max(Dom(Cvs)) + 9

d6. Ccall = Cvs @ mov rag retaddr @ sst 0 rag
            @ mov rab retaddr @ sst 1 rab
            @ mov rfg B(g)    @ mov rfb B(f)
            @ jmp_G rfg       @ jmp_B rfb
            @ sld_G (2+0)      Vx(x1)
            @ ...
            @ sld_B (2+vspace-1) Vb(xn)
            @ sld_G Vg(x) 0   @ sld_B Vb(x) 1
            @ sfree (vspace+2)

------------------------------------------------------------------- (trans-s-call)
[[ F;A;L |- x = g(vs) ok ]]_f C tcount V B = Ccall tcountcall
```

```
6. Exists El,us. s = El : us                            | def of s
7. Exists P',Em'. P' |- Ctemps : (D,g,(),Em',s')        | Code Addition Lemma, 1, 6, (salloc-t), (sst-
t), d4
   where s' =    El-spillspace : ns
              :: ...
              :: El-vspace-1: ns
              :: El-vspace : G(Vg(x1))                  | El - spillspace + argspace = vspace
              :: ...
              :: El-1 : G(Vb(xn))
              :: El : us


8. Exists Ps,Ems,Gs. Ps |- Cvs : (D,Gs,(),Ems,ss)      | Argument Translation Lemma, p1, def of vs, 1,
7, d1, d2, d3
   where ss =    El-spillspace : ns
              :: El-spillspace+1 : ns
              :: El-spillspace+2 : <G,[[tau1]],E1>
              :: ...
              :: El-vspace-1: <B,[[taun]],En>
              :: El-vspace : G(Vg(x1))                  | El - spillspace + argspace = vspace
              :: ...
              :: El-1 : G(Vb(xn))
              :: El : us
   and ps = xn:tau1, ..., xn:taun
   and Gs <= G


41. let Tret = (Dr,Gr,(),Emr,sr)                        | build type after call returns
    where
    Dr = amemr : kmem, at:kint
    Gr = spg -> <G,sptr,El-vspace-2>, spb -> <B,sptr,El-vspace-2>
         pcg -> <G,int,ar>, pcb -> <G,int,ar>
         d -> <G,int,0>
    E_mr = amemr
    sr =   El-vspace-2 : <G,[[tau]],at>
        :: El-vspace-1 : <B,[[tau]],at>
        :: El-vspace   : G(Vg(x1))
        :: ...
        :: El-1        : G(Vb(xn))
        :: El          : us
    Tr = (Dr, Gr, (), E_mr, sr)

42. P',retaddr ->(Tret->void) - retaddr : Tret->void| (heap-addr-t)

10. Exists Pg,Emg. Pg |- Cvs @ mov rag retaddr      | Code Addition Lemma, 8, (mov-n-t), (sst-t),
(jmp_G-t), 42
                             @ sst 0 rag
                             @ mov rab retaddr
                             @ sst 1 rab
                             @ mov rfg B(g)
                             @ mov rfb B(f)
                             @ jmp_G rfg
                         : (D,Gg,(),Emg,sg)
   where sg =    El-spillspace   : <G,[[Tret->void]],retaddr>
              :: El-spillspace+1 : <B,[[Tret->void]],retaddr>
              :: El-spillspace+2 : <G,[[tau1]],E1>
              :: ...
              :: El-vspace-1: <B,[[taun]],En>
              :: El-vspace : G(Vg(x1))                  | El - spillspace + argspace = vspace
```

```
               :: ...
               :: El-1 : G(Vb(xn))
               :: El : us
         Gg <= Gs                                  | only fresh temp regs are modified

11. equalities hold for all pairs in current state  | Inversion of (T-V-wf), 2
12. equalities hold for all pairs in target state   | 4, (trans-functtp)
    using fresh exp vars
13. target stack is abstracted above args           | 4, (trans-functtp)
14. target mem is a fresh exp var                   | 4, (trans-functtp)
141. target's ret type can be satified by Tret      | 41, 4, (trans-functtp), g:ps->tau
15. build S                                         | 11, 12, 13, 14

16. target d, pcG, pcB have appropriate types       | 4, (trans-functtp)
17. S(Gg) <= G_target                               | 15, 4, (trans-functtp), 10, (gen-Theta), 3
18. S(sg) <= s_target                               | 15, 4, (trans-functtp), 10, (gen-Theta), 3
19. S(Emg) <= Em_target                             | 15, 14

20. Pg |- Cvs @ ... @ jmp_B rfg                     | (C-t), 10, (jmp-t), 15-19

21. Exists Pc.
    Pc;Tret |- sld_G (2+0) Vx(x1)                   | Code Addition Lemma, 41, (sld_G-t), (sld_B-
t), (sfree-t), (gen-Theta)
            @ ...
            @ sld_B (2+vspace-1) Vb(xn)
            @ sld_G Vg(x) 0   @ sld_B Vb(x) 1
            @ sfree (vspace+2)
            : Tcall
     and Tcall <= [[ F(f);L ]]_V
     and (A union L);V |- Tcall wf

23. f;V;B;F;A;L |- Ccall : Pc                       | (wf-s), 21, 4
*
```

```
CASE TRANS-S-IF:
~~~~~~~~~~~~~~~

d1. lc = max(dom(C)) + 1
p1. [[ A union L) |- v : int ]] C tcount V  = Cv tcountv rgv rbv

p2. (tcountc, rtg, rfg, rjtg, rjfg, rtb, rfb, rjtb, rjfb, reg, reb)
 = freshReg(tcountv) // fresh registers for the block labels

d2. lfixtrue = max(Dom(Cv)+1)
d3. Cc = Cv @ mov rtg 1 @ mov rtb 1
            @ bz_G rgv rtg @ bz_B rbv rtb

p3. [[ F;A;L |- s2 ok ]]_f Cc tcountc V = Cf tcountf
d4. lfixjoin = max(Dom(Cf)+1)
d5. Cf' = Cf @ mov rjfg 1 @ mov rjfb 1
         @ jmp_G rjfg @ jmp_B rjfb

d6. lbztarget = max(Dom(Cf')+1)
p4. [[ F;A;L |- s2 ok ]]_f Cf' tcountf V = Ct tcountt
d7. ljoin = max(Dom(Ct)+5)

d8. Ct' =  Ct @ mov reg ljoin @ mov reb ljoin
               @ jmp_G reg @ jmp_B reb )
d9. C' =  Ct'[lfixjoin   -> mov rjfg ljoin]
```

```
              [lfixjoin+1 -> mov rjfb ljoin]
              [lfixtrue -> mov rtg lbztarget]
              [lfixtrue+1 -> mov rtb lbztarget]
d10. tcount' = tcount

-------------------------------------------------------------------------------- (trans-s-if)
[[ F;A;L |- if v then s1 else s2 ok ]]_f C tcount V B = C' tcount'



6. Exists Pc, Emc.
   Pc |- Cc : (D,Gc,(),Emc,s)               | Code Addition Lemma, 1, 6, Value Translation Lemma,
(mov-n-t),(bz_G-t),(bz_B-t), 4
   where Gc <= G
7. f;V;B;F;A;L |- Cc : Pc                   | (wf-s), (T-V-wf), 6, 4

8. Exists P1. f;V;B;F;A;L |- Cf : P1        | IH, p3, 7

9. Exists P1', Em1'.                        | Block Addition Lemma, 1, 6, Value Translation Lemma,
(mov-n-t),(jmp_G-t),(jmp_B-t), 4
   P1' |- Cf' : (D,Gc,(),Em1',s)
   where G1 <= G
   and P1'(lbztarget) = [[ F(f);L ]]_V

10. f;V;B;F;A;L |- Cf' : P1'                | (wf-s), (T-V-wf), 9, 4

11. Exists P2. f;V;B;F;A;L |- Ct : P2       | IH, p4, 10

12. Exists P2', Em2'.                       | Block Addition Lemma, 1, 10, Value Translation Lemma,
(mov-n-t),(jmp_G-t),(jmp_B-t), 4
   P2' |- Ct' : (D,G2,(),Em2',s)
   where G2 <= G
   and P2'(ljoin) = [[ F(f);L ]]_V

13. f;V;B;F;A;L |- C' : P2'                 | (wf-s), (T-V-wf), 12, 4
14. Exists Pfix. f;V;B;F;A;L |- C' : Pfix   | 13, 9, 12
*




CASE TRANS-S-WHILE:
~~~~~~~~~~~~~~~~~~~

similar to TRANS-S-IF.
*










Function Translation Lemma
**************************
```

```
If [[ F |- fds : F' ]] C B = C' B' n
    and B;F |- C : P
then exists P'. B';F' |- C' : P'

Proof: by induction on the structure of [[ F |- fds : F' ]] C B = C' B' n

CASE TRANS-FDS-EMPTY: trivial



CASE TRANS-FDS:

d1. faddr = max(Dom(C)+1))
p1. prologue(A) C 0 V 0 = Cp tcountp Vp
p2. [[ F[f : A -> tau];A;. |- lds : L ]] Cp tcountp Vp = Clds tcountlds Vlds
p3. [[ F[f : A -> tau];A;L |- s ok ]]_f Clds tcountlds Vlds B = Cs tcounts
p4. [[ (A union L) |- v : tau ]]_epilogue Cs tcounts Vlds = Cf tcountf
p5. [[ F[f : A -> tau] |- fds : F' ]] Cf B[f -> faddr] = C' B' tcountfds
d2. tcount = max(tcountf, tcountfds)
-------------------------------------------------------------------------------- (trans-fds)
[[ F |- tau f(ps) {lds; s; return v;} fds : F']] C B = C' B' tcount


1. 0 not in Dom(C)                                      | Inversion of (wf-F), assumption 2
2. Dom(P) = Dom(C)
3. all l in Dom(C).
    P(l) = T -> void
    and P;T |- C(l) => RT
    and (RT = T' ==> P(l+1) = T' -> void)
4. all f in F. P(B(f)) = [[F(f)]]
5. P(ha) = Thalt

6. Exists Pp.   f;Vp  ;B;F[f: A->tau];A;. |- Cp   : Pp   | Prologue Lemma, p1, assumption 2
7. Exists Plds. f;Vlds;B;F[f: A->tau];A;L |- Clds : Plds | Local Declaration Translation Lemma, p2,
6
8. Exists Ps.   f;Vlds;B;F[f: A->tau];A;L |- Cs   : Ps   | Statement Translation Lemma, p3, 7
9. Exists Pf.   F[f:A->tau]; B |- Cf : Pf                | Epilogue Lemma, p5

10. Exists P'.  F';B' |- C' : P'                         | I.H. p5, 9
*






Translation Theorem
********************

If [[ |- fds;lds;s ok ]] = C tcount startaddr
then
st = min(Dom(C)-3)
R = buildR(tcount)[pcg -> startaddr][pcb -> startaddr][spg -> st][spb -> st]
M = st+2 -> 0, st+1 -> ha ,  st -> ha
|- (R, C, M, (), .)




Proof:

1. Cstart =  ha->mov t0 1, ha+1->mov t1 1, ha+2->jmp_G t0, ha+3->jmp_G t1    | Inversion of (trans-
p), assumption
2. [[ . |- fds : F ]]   Cstart  0 . = Cfds Bfds tcountfds
```

```
3. startaddr = max(Dom(Cfds)+1)
4. [[ F;.;. |- lds : L ]] Cfds 0 . = Clds tcountlds Vlds
5. [[ F;.;L |- s ok ]]  Clds tcountlds Vlds Bfds = Cs tcounts
6. [[ L |- v : tau]]_epilogue Cs tcounts Vlds () = C tcountr
8. tcount = max(tcounthalt, tcountfds)

d1. Dhalt = ap:kint,am:kmem,al:kint,as:kstack
d2. Ghalt = pcg -> <G,int,1>,   pcb -> <B,int,1>,
           spg -> <G,sptr,al>, spb -> <B,sptr,al>,
           d   -> <G,int,0>
d3. Thalt = ( Dhalt, Ghalt, (), am, al:as )
d4. Pstart = 1 -> Thalt
             2 -> ( Dhalt, Ghalt[t0-><G,Thalt,1>], (), am, al:as )
             3 -> ( Dhalt, Ghalt[t0-><G,Thalt,1>][t1-><B,Thalt,1>], (), am, al:as )
             4 -> ( Dhalt, Ghalt[t0-><G,Thalt,1>][t1-><B,Thalt,1>][d-><G,Thalt,1>], (), am, al:as )

9.  Pstart |- Cstart                                                | (C-t), 1, d4, (mov-n-t),
(jmp_G-t), (jmp_B-t)
10. .;. |- C : P                                                    | deconstruction of (C-t), 9,
reconstruction of (wf-F), d4

11. exists Pfds. Bfds;F |- Cfds : Pfds                              | Function Translation Lemma,
2, 10

d5. F' = F[main : ()->int]                                          | main is a fresh function name
not in F
    B' = B[main : startaddr ]

d6. Tmain = ( (al:kint,as:kstack,am:kmem,apc:kint),
              (spg -> <G,sptr,al-2>, spb -> <B,sptr,al-2>,
               pcg -> <G,int,apc>, pcb -> <G,int,apc>, d -> <G,int,0>),
              (),
              am,
              al-2:<G,Tr->void,int> :: al-1: <B,Tr->void,int> :: al : as )
            where Tr = ( (amr:kmem,at:kint),
                            (spg -> <G,sptr,al-2>, spb -> <B,sptr,al-2>,
                             pcg -> <G,int,apc>, pcb -> <G,int,apc>, d -> <G,int,0>),
                            amr,
                            (),
                            al-2 : <G,int,at> :: al -1 : <B,intt,at> :: al : as )


d7. P' = Pfds, startaddr -> (Tmain -> void)
12. P' |- Cfds : Tmain                                              | (C-partial-t), inversion of
(wf-F), 11, d7
13. .;. |- Tmain wf                                                 | (T-V-wf)
14. all f' in F. P(B'(f)) = [[F(f')]]                               | inversion of (wf-F), 11, d5
15. P(ha) = Thalt                                                   | inversion of (wf-F), 11
16. Tmain <= [[ ()->int;. ]]_.                                      | d6, (gen-Theta)
17. main;.;B';F';.;. |- Cfds : P'                                   | (wf-s), 12, 13, 14, 15, 16
18. Exists Plds. main;Vlds;B';F';.;L |- Clds : Plds                 | Local Decl Translation Lemma,
4, 17, weakening of B/F
19. Exists Ps.   main;Vlds;B';F';.;L |- Cs : Ps                     | Statement Translation Lemma,
5, 18, weakening of B/F
20. Exists P. B';F' |- C : P                                        | Epilogue Lemma, 6, 19


d10. Ms = M
d11. Mm = .
```

```
1'. Dom(P) = Dom(C) union Dom(Mm)                                       | Inversion of (wf-F), 20, d11
2'. M = Ms #(dom(C)) Dom(Mm)                                            | (#-def), def of M, st
3'. P |- C                                                              | Inversion of (wf-F), 20
4'. ir = .
5'. P(startaddress) = (Dmain,Gmain,(),am,smain)                         | d7, code mem part of P never
removed
    where Dmain = (al:kint,as:kstack,am:kmem,apc:kint),
          Gmain = (spg -> <G,sptr,al-2>, spb -> <B,sptr,al-2>,
                   pcg -> <G,int,apc>, pcb -> <G,int,apc>, d -> <G,int,0>),
          smain = al-2:<G,Tr->void,int> :: al-1: <B,Tr->void,int> :: al : as )
          Tr = ( (amr:kmem,at:kint),
                 (spg -> <G,sptr,al-2>, spb -> <B,sptr,al-2>,
                   pcg -> <G,int,apc>, pcb -> <G,int,apc>, d -> <G,int,0>),
                  amr,
                  (),
                  al-2 : <G,int,at> :: al -1 : <B,intt,at> :: al : as )

6'. S = (st+2/al), (sbase/as), (emp/am), (startaddr/apc)

21. P |- (st+2 -> 0) : st+2 : sbase                                     | (s-t-base)
22. P;. |- ha : <B,Tr->void,int>                                        | (addr-heap-t), d4
23. P;. |- ha : <G,Tr->void,int>                                        | (addr-heap-t), d4
7'. P |- Ms : st:<G,Tr->void,int>                                       | (s-t-cons), 21, 22, 23
              :: st+1:<B,Tr->void,int> :: st+2:sbase

24. all l in Dom(Mm)....                                                | d11
25. [[emp]] = Mm                                                        | d11, [[ ]]
26. P |- ():()                                                          | (Q-emp-t)
8'. P |- (Mm,Q) : (S(am),S())                                          | (heap-t), 24, 25, 26

27. P;. |- startaddr : <c,int,startaddr>                                | (int-t)
28. P;. |- st : <c,sptr,st>                                             | (sptr-t)
9'. P |- R : S(G)                                                       | (reg-file-t), 27, 28, 6', 5'

30. |- (R, C, M, (), .)                                                 | (S-t), 1', 2' 3', 4', 5', 6',
7', 8', 9'
*
```

TAL_CF Syntax

```
MACHINE STATE SYNTAX
~~~~~~~~~~~~~~~~~~~~~

colors      c  ::=  B  |  G  |  R
values      v  ::=  c n

registers   r  ::=   ri  |  r1  |  ...  |  rn
reg file    R  ::=  {r->v,...,r->v}

instructions    i  ::=  movi rd v               // rd := v
            |  sub rd rs1 rs2       // rd := rs1 - rs2
            |  intend rt            // intend to jump to rt (ri := rt)
            |  intendz rz rt        // if rz = 0 then intend to jump to rt
            |  recovernz rz         // if rz != 0, branch to fault recovery code

block       b  ::=  i;b  | jmp rt  |  brz rz rt

Code Memory C  ::=  {n->b,...,n->b}

History         h  ::= l1 ... lk            // often written (h,lk) when only the last loc is needed


State          S ::= (C, h, R, b)
Machine States MS ::= S | recover(h) | hw-error(h)




TYPING SYNTAX
~~~~~~~~~~~~~~

Kinds       k  ::=  kint | kseq
Exp Contexts    D  ::=  .  |  D, x:k
Substitutions   S  ::=  .  |  S, E/x

ri types    rit::=  ok  |  go  |  goz  |  check
base types  t' ::=  int  |  All[D](G,A)  |  rit
static exps E  ::=  x  |  n  |  E - E  |  E ? E : E

types       t  ::=  <c,t',E>
reg file type   G  ::=  .  |  G, r->t

type option to ::=  t  |  undef
heap typing P  ::=  .  |  P, n->t    // P(n) = undef if n not in Dom(P)

location seq    seq ::=  empty  |  alpha  |  seq o E

fault tag   f  ::=  c  |  cf
zap tag     Z  ::=  .  |  f



   cf
  / \
  g  b  r
   \ | /
     .
```

Dynamic Semantics

```
 -----------
| S -->_0 S |
 -----------


---------------------------------------------------- (movi)
(C, h, R, movi rd v; b) -->_0 (C, h, R[rd -> v], b)



v' = R_col(rs1) (R_val(rs1) - R_val(rs2))
--------------------------------------------------------------- (sub)
(C, h, R, sub rd, rs1, rs2; b) -->_0 (C, h, R[ rd -> v' ],  b)



---------------------------------------------------------- (intend)
(C, h, R, intend rt; b) -->_0 (C, h, R[ri -> R(rt)],  b)


R_val(rz) = 0
-------------------------------------------------------------- (intendz-set)
(C, h, R, intendz rz rt; b) -->_0 (C, h, R[ri -> R(rt)],  b)


R_val(rz) =/= 0
------------------------------------------------- (intendz-unset)
(C, h, R, intendz rz rt; b) -->_0 (C, h, R, b)



R_val(rz) = 0
---------------------------------------------- (recovernz-ok)
(C, h, R, recovernz rz; b) -->_0 (C, h, R, b)


R_val(rz) =/= 0
-------------------------------------------- (recovernz-halt)
(C, h, R, recovernz rz; b) -->_0 recover(h)


R_val(rz) =/= 0     l+1 in Dom(C)
---------------------------------------------------------------------------- (brz-untaken)
(C, (h,l), R, brz rz rt) -->_0 (C, (h,l,l+1), R[ri -> R R_val(ri)], C(l+1))


R_val(rz) = 0   R_val(rt) in Dom(C)
------------------------------------------------------------------------------------------ (brz-taken)
(C, (h,l), R, brz rz rt) -->_0 (C, (h,l,R_val(rt)), R[ri -> R R_val(ri)], C(R_val(rt)))


R_val(rz) = 0    R_val(rt) not in Dom(C)
----------------------------------------- (brz-hw-error)
(C, h, R, brz rz rt) -->_0 hw-error (h)


R_val(rt) in Dom(C)
------------------------------------------------------------------------------ (jmp)
(C, h, R, jmp rt) -->_0 (C, (h,R_val(rt)), R[ri -> R R_val(ri)], C(R_val(rt)))


R_val(rt) not in Dom(C)
---------------------------------- (jmp-hw-error)
(C, h, R, jmp rt) -->_0 hw-error(h)
```

```
 -----------
| S -->_1 S |
 -----------

R(r) = c n
------------------------------------------- (zap-reg)
(C, h, R, b) -->_1 (C, h, R[r -> c n'], b)


R(rz) =/= 0     l in Dom(C)
--------------------------------------------------------------- (zap-recover-linC)
(C, h, R, recover rz; b) -->_1 (C, (h,l), R, C(l))

R(rz) =/= 0
--------------------------------------------------------------- (zap-recover-lnotinC)
(C, (h,l), R, recover rz; b) -->_1 hw-error(h)
```

TAL_CF Typing

```
 --------
| D |- E |        Static Expression Equality
 --------

x in Dom(D)
------------ (wf-var)
D |- x : D(x)


-------------- (wf-int)
D |- n : kint


D |- E1:kint   D |- E2:kint
--------------------------- (wf-sub)
D |- E1 - E2 : kint


D |- E1:kint   D |- E2:k   D |- E3:k
------------------------------------ (wf-ifexp)
D |- E1 ? E2 : E3 : k




 -------------
| D |- E = E  |       Static Expression Equality
 -------------

D |- E1:kint   D |- E2: kint
Forall S.  . |- S : D  ==>  [[S(E1)]] = [[S(E2)]]
--------------------------------------------------- (E-eq)
D |- E1 = E2


D |- E1:kint   D |- E2: kint
Forall S.  . |- S : D  ==>  [[S(E1)]] =/= [[S(E2)]]
--------------------------------------------------- (E-eq)
D |- E1 =/= E2



D |- E1 = E2   D |- seq1 = seq2
----------------------------------
D |- seq1 o E1 = seq2 o E2


----------------------
D |- empty = empty




 ---------
|  [[E]]  |
 ---------

[[n]]  =  n
[[E1 - E2]]  =  [[E1]] - [[E2]]
[[Eb?Et:Ef]] = if [[Eb]] then [[Et]] else [[Ef]]




 --------------
```

```
| D |- S : D'  |
 --------------

                D |- S : D'    D |- E:k    x not in (D union D')
----------- (subst-empt-t)  -----------------------------------------------  (subst-t)
D |- . : .          D |-  S,E/x  :  (D',x:k)





 -------------
| P |- n : t' |       Integer Typing Judgment
 -------------


--------------- (int-t)     --------------- (address-t) ------------- (rit-t)
P |- n : int            P |- n : P(n)           P |- n : rit





 ---------------
| D |-  t <= t' |    Subtyping Judgment
 ---------------

D |- E1 = E2
-------------------------- (subtp-reflex)
D |- <c,b,E1> <= <c,b,E2>


D |- E1 = E2
--------------------------- (subtp-int)
D |- <c,b,E1> <= <c,int,E2>


Forall r. G1(r) <= G2(r)
-------------------------------- (G-subtp)
D |- G1 <= G2





 -----------------
| D; P |-Z  v : t | Value Typing Judgment
 -----------------



P |- n : t'     D |- E = n
--------------------------- (val-t)
D;P |-Z  c n : <c,t',E>


D |- E : kint
--------------------------- (val-zap-c-t)
D;P |-c  c n : <c,t',E>


D |- E : kint
c' = B or c' = G                    <-- Note: value can be red
--------------------------- (val-zap-cf-t)
D;P |-cf  c n : <c',t',E>






 -----------------
| D;P;G |- i : G' |    Instruction Typing Judgment
 -----------------

rd =/= ri
--------------------------------------------------- (movi-t)
(D;P;G) |- movi rd c i : G[ rd -> <c,int,i> ]


rd =/= ri
```

```
G(rs1) = <c,int,Es1>   G(rs2) = <c,int,Es2>
----------------------------------------------------- (sub-t)
D;P;G |- sub rd rs1 rs2 : G[ rd -> <c,int,Es1-Es2> ]




G(ri) = <ci,ok,Ei>    G(rt) = <B,All[Dt](Gt,At),Et>
------------------------------------------------------ (intend-t)
D;P;G |- intend rt : G[ ri -> <B,go,Et> ]


G(ri) = <B,go,Ei>
G(rt) = <B,All[Dt](Gt,At),Et>
G(rz) = <B,int,Ez>
------------------------------------------------------ (intendz-t)
D;P;G |- intendz rz rt : G[ ri -> <B,goz,Ez?Ei:Et> ]





 --------------------
| D;P;G;A;Ei;to |- b |  Block Typing Judgment
 --------------------
// Ei is where we had intended to go, on block entry will always describe ri
// to is type of fallthru block


D;P;G |- i : G'     D;P;G';A;Ei;to |- b
-------------------------------------------------------- (sequence-t)
D; P; G; A; Ei; to |- i;b



G(rz) = <R,int,Ez>    D,x:kint |- Ez = El - x
G(ri) = <R,check,x>
D |- G/ri/rz wf      D |- seq wf    D |- El : kint
D;P;G[rz-><R,int,0>][ri-><B,ok,El>]; seq o El; El; to |- b
------------------------------------------------------------------- (recovernz-t)
(D,x:kint); P; G; seq o El; x; to |- recovernz rz; b



G(rz) = <R,int,Ez>
G(ri) = <R,check,Eli>
. |- Ez = Eli - Ela
. |- Eli = Ela
.; P; G[ri -> <R,ok,Eli>]; seq o Ela; Eli; to |- b
------------------------------------------------- (recovernz-eq-t)
.; P; G; seq o Ela; Eli; to |- recovernz rz ; b



G(rz) = <R,int,Ez>
G(ri) = <R,check,Eli>
. |- Ez = Eli - Ela
. |- Eli =/= Ela
------------------------------------------------- (recovernz-neq-t)
.; P; G; seq o Ela; Eli; to |- recovernz rz; b



G(ri)= <B,goz,Ez'?Ef':Et'>
D |- Ef' = Ela + 1

G(rz) = <G,int,Ez>
D |- Ez = Ez'

G(rt) = <G,All[Dt](Gt,At),Et>
D |- Et = Et'

Exists St. D |- St : Dt
D |- G[ri -> <R,check,Et'>] <= St(Gt)
D |- seq o Ela o Et' = St(At)

Exists Sf. D |- Sf : Df
D |- G[ri -> <R,check,Ef'>] <= Sf(Gf)
D |- seq o Ela o Ef' = Sf(Af)
--------------------------------------------------------------------------- (brz-t)
(D; P; G; seq o Ela; Ei; All[Df](Gf,Af) ) |- brz rz rt
```

```
G(ri)= <B,go,Et'>

G(rt) = <G,All[Dt](Gt,At),Et>
D |- Et = Et'

Exists S. D |- St : Dt
D |- G[ri -> <R,check,Et'>] <= St(Gt)
D |- seq o Ela o Et = St(At)
----------------------------------------- (jmp-t)
(D; P; G; seq o Ela; Ei; t) |- jmp rt
```

```
 ----------
| |- C : P |    Code Typing Judgment
 ----------
```

```
Forall n in Dom(C) union Dom(P).
P(n) = All[D,Y:kint,alpha:kseq]((G,ri-><R,check,Y>),alpha o n)
/\ D |- G wf  /\ Forall r' in Dom(G). G(r') =/= <R,t',E'>
/\ D |- P(n+1) wf
/\ (D,Y:kint,alpha:kseq); P; (G,ri-><R,check,Y>); alpha o n; Y; P(n+1) |- C(n)
-------------------------------------------------------------------------------- (C-t)
|- C : P
```

```
 ------------
| P |- R : G |  Register File Typing Judgment
 ------------
```

```
Forall r. P;. |-Z  R(r) : G(r)
------------------------------ (R-t)
P |-Z  R : G
```

```
 -------------
| |-  h : seq |   Sequence Typing Judgment
 -------------
```

```
------------------------------ (h-empty-t)
|- () : empty
```

```
|- E = n
|- h : seq
------------------------------ (h-append-t)
|- (h,n): seq o E
```

```
 ------------------
| |-Z (C, h, R, b) |    Machine Typing Judgment
 ------------------
```

```
|- C : P
P |-Z R : G
|- (h,l) : A
Z = cf  ? . |- Ei =/= l  : |- Ei = l
G(ri) =/= <R,check,Ei> ==> . |- Ei = l
.;P;G;A;Ei;P(l+1) |- b
------------------------------------ (S-t)
|-Z (C,(h,l),R,b)
```

Lemmas

```
Canonical Forms Lemma
~~~~~~~~~~~~~~~~~~~~~

If  .;P |-Z c n : <c',t',E'>  and  |- C : P


then
1. if Z = cf and (c' = B or c' = G), then . |- E' : kint

2. if Z = c' then c = c' and . |- E' : kint

3. if Z =/= c' and (Z =/= cf or c' =/= B and c' =/= G) then
    c = c' and
    t' = int  ==>  . |- E' = n
    t' = rit  ==>  . |- E' = n
    t' = All[D](G,A)  ==>  n in Dom(C)  and  . |- E' = n




Subtyping Lemma:
----------------
If D |- t <= t' and D;P |-Z v:t then D;P |-Z v:t'

Proof:
By induction on the derivation of D;P |-Z v:t.  Each case uses inversion
of the subtyping rules and transitivity of D |- E1 = E2.



Equal Code Labels Lemma:
------------------------

If .;P |-f c n1: <c,All[D1](G2,A2),Et>  and  . |- Et = n2
then P |- n2 : All[D1](G2,A2)

Proof:

For some f =/= c, inversion tells us that P |- All[D1](G2,A2) : Et and . |- Et = n1.
By transitivity, . |- n1 = n2.
By equality, P |- n2 : All[D1](G2,A2)




Exp Conditional Lemma
~~~~~~~~~~~~~~~~~~~~~
If D |- Ez = 0    then  D |- Ez?Ef:Et = Et.
If D |- Ez =/= 0  then  D |- Ez?Ef:Et = Ef.

Proof: By definition of D |- E = E and definition of [[E]]



Substitution Structure Lemma
~~~~~~~~~~~~~~~~~~~~~~~~~~~~

If D |- S : (D',x:k)
then Exists E, S'.  S = S', E/x  and  D |- E : k  and  D |- S' : D'

Proof: By inspection of (subst-t)
```

```
Exp Eq Trans Lemma
-------------------------

If  D |- E1 = E2   and  D |- E2 = E3  then  D |- E1 = E3
If  D |- seq1 = seq2  and  D |- seq2 = seq3  then  D |- seq1 = seq3

Proof: By definition of D |- E1 = E2 and definition of [[E]].




Color Weakening Lemma
~~~~~~~~~~~~~~~~~~~~~

If D;P |- v : t and then D;P |-c v : t
If D;P |-c v: t and c = B or c = G  then D;P |-cf v:t

Proof: By case analysis of D;P |- v : t



Substitution Lemma
~~~~~~~~~~~~~~~~~~
1. If D,x:k |- E': k'  and  D |- E : k  then  D |- E'[E/x] : k'
2. If D,x:k |- E1 = E2  and  D |- E : k  then D |- E1[E/x] = E2[E/x]
3. If D,x:k;P |-Z v : t  and  D |- E : k  then  D;P |-Z v : t[E/x]
4. If D,x:k;P;G;A;Et;t |- b  and  D |- E : k  then  D;P;G[E/x];A[E/x];Et[E/x];t[E/x] |- b

5. If (D1,D2) |- E':k'  and  D1 |- S : D2  then D1 |- S(E') : k'
6. If (D1,D2) |- E1 = E2  and  D1 |- S : D2  then  D1 |- E1[E/x] = E2[E/x]
7. If (D1,D2);P |-Z v : t  and  D1 |- S : D2  then D1;P |-Z v : S(t)
8. If (D1,D2);P;G;I;A;Ei;to |- b  and D1 |- S : D2  then  D1;P;S(G);S(I);S(A);S(Ei);S(to) |- b


PROOF:
1. By induction on the structure of D,x:k |- E':k'.
2. By case analysis on the structure of D,x:k |- E1 = E2 using Part 1.
3. By case analysis on the structure of D,x:k;P |-Z v : t using Parts 1 and 2.
4. By induction on the structure of D,x:k;P;G;I;A |- b using Parts 1-3. The case for recovernz-t
   divides into two subcases depending on if Ez = 0 and uses recovernz-eq-t or recovernz-neq-t as appropriate.

5-8. By induction on the size of D, using Parts 1-4 respectively.
```

Progress

```
1. If |- S then S -->_0 S'
2. If |-Z S then S -->_0 FS


=========================================================================================================================

PART 1: If |- (C,h,R,b) then (C,h,R,b) -->_0 (C',h',R',b')



PROOF: By case analysis on b



CASE MOVI: b = movi rd v; b'
~~~~~~~~~~~~~~~~~~~~~~~~~~~~

1. (C, h, R, movi rd v; b') -->_0 (C, h, R[rd -> v], b')            [ (movi) ]
* MOVI complete



CASE SUB: b = sub rd rs rs; b'
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

d1. let v' = R_col(rs) (R_val(rs) - R_val(rt))                      [ definition ]
1. (C,h,R,sub rd, rs, rt; b') -->_0 (C,h,R[ rd -> v' ], b')         [ (sub) ]
* SUB complete



CASE INTEND: b = intend ri rt; b'
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

1. (C,h,R,intend rd rs; b') -->_0 (C,h,R[rd -> R(rs)], b')          [ (intend) ]
* INTEND complete



CASE INTENDZ: b = intendz rz rt; b'
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

subcase on R_val(rz) =?= 0:

SUBCASE INTENDZ.a:
aa1. R_val(rz) = 0                                                  [ subcase assumption ]
1a. (C,h,R,intendz rz rd rs; b') -->_0 (C,h,R[rd -> R(rs)],b')      [ (intendz-set), aa1 ]
* INTENDZ.a complete

SUBCASE INTENDZ.b:
ab1. R_val(rz) =/= 0                                                [ subcase assumption ]
1b. (C,h,R, intendz rz rd rs; b') -->_0 (C,h,R,b')                  [ (intendz-unset), ab1 ]
* INTENDZ.b complete




CASE RECOVERNZ: b = recovernz rz; b'
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

a1. |- (C,(h,l),R,recovernz rz; b')

1. |- C : P                                                         [ Inversion of (S-t), a1 ]
2. P |- R : G
3. . |- (h,l) : A
4. . |- Ei = l
5. .;P;G;A;Ei;P(l+1) |- b'
```

```
subcase on the structure of 5 (recovernz-t doesn't apply since D = .)

SUBCASE RECOVERNZ.A:

G(rz) = <R,int,Ez>
G(ri) = <R,check,Ei>
. |- Ez = Ei - Ela
. |- Ei = Ela
.; P; G[ri -> <R,ok,Ei>]; seq o Ela; Ei; t |- b
------------------------------------------------- (recovernz-eq-t)
.; P; G; seq o Ela; Ei; t |- recovernz rz ; b

6a. .;P |-Z R(rz) : <R,int,Ez>                                           [ Inversion of (R-t), 2, pa1 ]
7a. . |- R_val(rz) = Ez                                                  [ Canonical Forms, 6a, 2 ]
8a. . |- Ela - Ei = 0                                                    [ pa4 ]
9a. . |- R_val(rz) = 0                                                   [ Exp Eq Transitivity, 7a, pa3, 8a ]

10a. (C, (h,l), R, recovernz rz; b) -->_0 (C, (h,l), R, b)               [ recovernz-ok, 9a ]
* RECOVERNZ.A complete


SUBCASE RECOVERNZ.B

G(rz) = <R,int,Ez>
G(ri) = <R,check,Ei>
. |- Ez = Ei - Ela
. |- Ei =/= Ela
------------------------------------------------- (recovernz-neq-t)
.; P; G; seq o Ela; Ei; t |- recovernz rz; b

6b. . |- Ela = l                                                         [ Inversion of (h-append-t), 3 ]
7b. . |- Ela = Ei                                                        [ Exp Eq Transitivity, 4, 6b ]
8b. subcase does not apply                                               [ 7b contradicts pb4 ]
* RECOVERNZ.B complete




CASE BRZ: b = brz rz rt
~~~~~~~~~~~~~~~~~~~~~~~

a1. |- (C,h,R,brz rz rt)

subcase on R_val(rz) =?= 0:

1. |- C : P
2. P |- R : G
3. .;P;G;A;Ei;P(l+1) |- brz rz rt

subcase on R_val(rz) =?= 0

SUBCASE BRZ.a:
aa1. R_val(rz) = 0                                                           [ subcase assumption ]

4a. G(rt) = <G,All[Dt](Gt,It,Rt),Et>                                         [ Inversion of (brz-t), 3 ]
5a. .;P |- R(rt) : G(rt)                                                     [ Inversion of (R-t), 4a ]
6a. R_val(rt) in Dom(C)                                                      [ Canonical Forms, (4a, 5a), 1 ]

7a. (C,h,R,brz rz rt) -->_0 (C,(h,R_val(rt)),R[ri -> R R_val(ri)],C(R_val(rt))) [ brz-taken, aa1, 6a ]
* BRZ.a complete


SUBCASE BRZ.b:
ab1. R_val(rz) =/= 0                                                         [ subcase assumption ]

4b. P(l+1) = All[Df](Gf,Af)                                                  [ Inspection of (brz-t), 3 ]
5b. l+1 in Dom(P)                                                            [ 4b, def of P ]
6b. l+1 in Dom(C)                                                            [ Inversion of (C-t), 1, 5b ]
1b. (C,h,R,brz rz rt [ri]; b') -->_0 (C,h,R,b')                              [ brz-untaken, ab1, 6b ]
* BRZ.b complete




CASE JMP: b = jmp rt
~~~~~~~~~~~~~~~~~~~~
a1. |- (C,h,R,jmp rt)

1. |- C : P
2. P |- R : G
```

```
3. .;P;G;A;Ei;P(l+1) |- brz rz rt
4. G(rt) = <G,All[Dt](Gt,It,Rt),Et>                                             [ Inversion of (jmp-t), 3 ]
5. .;P |- R(rt) : G(rt)                                                         [ Inversion of (R-t), 2 ]
6a. R_val(rt) in Dom(C)                                                         [ Canonical Forms, (4, 5), 1 ]

7a. (C,h,R,jmp) -->_0 (C,(h,R_val(rt)),R[ri -> R R_val(ri)],C(R_val(rt)))       [ jmp, 6a ]
* JMP complete


** Progress Part 1 Complete.







=================================================================================================================

PART 2: If |-Z (C,h,R,b) then (C,h,R,b) -->_0 FS

PROOF: By case analysis on b




CASES MOVI, SUB, INTEND, INTENDZ:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

As in Progress Part 1. (Don't depend on typing info.)
* MOVI, SUB, INTEND, INTENDZ complete



CASE RECOVERNZ: b = recovernz rz; b'
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

subcase on R_val(rz) =?= 0:

SUBCASE RECOVERNZ.a:
aa1. R_val(rz) = 0
1a. (C,h,R,recovernz rz; b') -->_0 (C,h,R,b')                                   [ (recovernz-ok), aa1 ]
* RECOVERNZ.a complete

SUBCASE RECOVERNZ.b:
ab1. R_val(rz) =/= 0
1b. (C, h, R, recovernz rz; b) -->_0 recover(h)                                     [ (recovernz-halt), ab1 ]
* RECOVERNZ.b complete





CASE BRZ: b = brz rz rt
~~~~~~~~~~~~~~~~~~~~~~~~

a1. |-Z (C,h,R,brz rz rt)

subcase on R_val(rz) and then R_val(rd) in Dom(C).

SUBCASE BRZ.a:
aa1. R_val(rz) =/= 0

1a. |- C : P                                                                        [ Inversion of (S-t), a1 ]
2a. .;P;G;A;Ei;P(l+1) |- brz rz rt
3a. P(l+1) = All[Df](Gf,Af)                                                         [ Inspection of (brz-t), 2a ]
4a. l+1 in Dom(P)                                                                   [ 3a, def of P ]
5a. l+1 in Dom(C)                                                                   [ Inversion of (C-t), 1a, 4a ]
6a. (C,h,R,brz rz rt [ri]; b') -->_0 (C,h,R,b')                                     [ brz-untaken, aa1, 5a ]
* BRZ.a complete

SUBCASE BRZ.b:
ab1. R_val(rz) = 0
ab2. R_val(rd) in Dom(C)
1b. (C,h,R,brz rz rt)-->_0(C,(h,l,R_val(rt)),R[ri -> R R_val(ri)],C(R_val(rt))) [ (brz-taken), ab1, ab2 ]
* BRZ.b complete

SUBCASE BRZ.b:
ac1. R_val(rz) = 0
ac2. R_val(rd) not in Dom(C)
1c. (C,h,R,brz rz rd; b) -->_0 hw-error(h)                                          [ (brz-hw-error, ac1, ac2 ]
* BRZ.c complete
```

```
CASE JMP: b = jmp rt
~~~~~~~~~~~~~~~~~~~~

a1. |-Z (C,h,R,brz rz rt)

subcase on R_val(rd) in Dom(C).

SUBCASE JMP.a:
aa1. R_val(rd) in Dom(C)
1a. (C,h,R,jmp rt)-->_0(C,(h,R_val(rt)),R[ri -> R R_val(ri)],C(R_val(rt)))          [ jmp, aa1 ]
* JMP.a complete

SUBCASE JMP.b:
ab1. R_val(rd) not in Dom(C)
1a. (C,h,R,jmp rt) -->_0 hw-error(h)                                                [ jmp-hw-error, ab1 ]
* JMP.b complete


** Progress Part 2 complete
```

Preservation

```
1. If |- S  and S -->_0 S' then |- S'
2. If |-f S  and S -->_0 S' then Exists Z'.  |-Z' S'  and Z' >= f
3. If |- S and S -->_1 S' then Exists c. |-c S'



Corollary: Preservation-Fault-Elevation

If |-Z S  and S -->_k S' then Exists Z'.  |-Z' S'  and Z' >= Z




==============================================================================================================
==============================================================================================================


************************************
* Lemma Preservation-No-Elevation: *
************************************

If |-Z (C,h,R,b) and b =/= brz and b =/= jmp and (C,h,R,b) --> S' then |-Z  S'


PROOF: By case analysis of S -->_0 S'
******


CASE MOVI:
~~~~~~~~~~

------------------------------------------------------------ (movi)
(C, (h,l), R, movi rd v; b) -->_0 (C, (h,l), R[rd -> v], b)


a1. |-Z (C, h, R, movi rd v; b)

1. |- C : P
2. P |-Z R : G
3. |- (h,l) : A
4x. Z = cf  ? . |- Ei =/= l  : |- Ei = l
4y. G(ri) =/= <R,check,Ei> ==> . |- Ei = l
5. .;P;G;A;Ei;to |- movi rd v; b

6. .;P;G; |- movi rd (c i) : G[ rd <- <c,int,i> ]                    [ Inversion of (seqence-t), 5, Inspection of (movi-t) ]
5'. .;P;G[rd-><c,int,i>];A;Ei;t |- b                                 [ Inversion of (sequence-t), 5, 6 ]

7. P |- i : int                                                      [ (int-t) ]
8. . |- i = i                                                        [ (E-eq) ]
9. .;P |- c i : <c,int,i>                                            [ (val-t), 7, 8 ]
10. .;P |-Z c i : <c,int,i>                                          [ Color Weakening Lemma, 9 ]
11. Forall r'. .;P |-Z R(r') : G(r')                                 [ Inversion of (R-t), 2 ]
2'. P |- R[rd -> c i] : G[rd -><c,int,i>]                            [ (R-t), 11, 10 ]

12. rd =/= ri                                                        [ Inversion of (movi-t), 6 ]
4y'. G[rd -><c,int,i>](ri) =/= <R,check,Ei> ==> . |- Ei = l          [ 12, 4y ]


12. |-Z (C, (h,l), R[rd -> v], b)                                    [ (S-t), 1, 2', 3, 4x, 4y', 5' ]
* MOVI complete



CASE SUB:
~~~~~~~~~

v' = R_col(rs1) (R_val(rs1) - R_val(rs2))
---------------------------------------------------------------------- (sub)
(C, (h,l), R, sub rd, rs1, rs2; b) -->_0 (C,(h,l), R[ rd -> v' ],  b)


a1. |-Z (C, (h,l), R, sub rd, rs1, rs2; b)
```

```
1. |- C : P
2. P |-Z R : G
3. |- (h,l) : A
4x. Z = cf  ? . |- Ei =/= l  : |- Ei = l
4y. G(ri) =/= <R,check,Ei> ==> . |- Ei = l
5. .;P;G;A;Ei;to |- sub rd, rs1, rs2; b

6. .;P;G |- sub rd, rs1, rs2 : G[ rd <- <c,int,Es1-Es2> ]       [ Inversion of (seqence-t), 5, Inspection of (sub-t) ]
5'. .;P;G[rd<-<c,int,Es1-Es2>];A;Ei;t |- b                      [ Inversion of (sequence-t), 5, 6 ]

7. G(rs1) = <c,int,Es1>                                         [ Inversion of (sub-t), 6 ]
8. G(rs2) = <c,int,Es2>

9.  .;P |-Z R(rs1) : <c,int,Es1>                                [ Inversion of (R-t), 2, 7 ]
10. .;P |-Z R(rs2) : <c,int,Es2>                                [ Inversion of (R-t), 2, 8 ]

subcase Z =/= c and (Z =/= cf or c =/= B and c =/= G)
11a. R_col(rs1) = c  and  . |- Es1 = R_val(rs1)                 [ Canonical Forms, 9, 1 ]
12a. R_col(rs2) = c  and  . |- Es2 = R_val(rs2)                 [ Canonical Forms, 10, 1 ]
13a. [[Es1]] = R_val(rs1) and [[Es2]] = R_val(rs2)              [ Inversion of (E-eq), 11a, 12a ]
14a. [[Es1]] - [[Es2]] = R_val(rs1) - R_val(rs2)                [ 13a ]
15a. [[Es1-Es2]] = R_val(rs1) - R_val(rs2)                      [ def of [[E]], 14a ]
16a. . |- Es1-Es2 = R_val(rs1) - R_val(rs2)                     [ (E-eq), 15a ]
17a. P |- R_val(rs1) - R_val(rs2) : int                         [ (int-t) ]
18a. .;P |-Z R_col(rs) (R_val(rs1) - R_val(rs2)) : <c,int,Es1-Et2>  [ (val-t), 11a, 16a, 17a ]

subcase Z = c
11b. R_col(rs1) = c  and  . |- Es1 : kint                       [ Canonical Forms, 9, 1 ]
12b. R_col(rs2) = c  and  . |- Es2 : kint                       [ Canonical Forms, 10, 1 ]
13b. . |- (Es1-Es2) : kint                                      [ (wf-sub), 12b ]
14b. .;P |-c R_col(rs1) (R_val(rs1) - R_val(rs2)) : <c,int,Es1-Et2> [ (val-zap-c-t), 13b, 11b ]

subcase Z = cf and (c = B or c = G)
11c. . |- Es1 : kint                                            [ Canonical Forms, 9, 1 ]
12c. . |- Es2 : kint                                            [ Canonical Forms, 10, 1 ]
13c. . |- (Es1-Es2) : kint                                      [ (wf-sub), 11c, 12c ]
14c. .;P |-cf R_col(rs) (R_val(rs1) - R_val(rs2)) : <c,int,Es1-Et2> [ (val-zap-cf-t), 12c ]

merge:
19. .;P |-Z R_col(rs) (R_val(rs1) - R_val(rs2)) : <c,int,Es-Et> [ 18a, 14b, 14c ]
2'. P |-Z R[ rd -> v' ]: G[ rd <- <c,int,Es1-Es2> ]             [ (R-t), (Inversion of (R-t), 2), (19, p1) ]

20. rd =/= ri                                                   [ Inversion of (sub-t), 6 ]
4y'. G[rd -><c,int,i>](ri) =/= <R,check,Ei> ==> . |- Ei = l     [ 12, 4y ]

21. |-Z (C, (h,l), R[ rd -> v' ],  b)                           [ (S-t), 1, 2', 3, 4x, 4y', 5' ]
* SUB complete




CASE INTEND:
~~~~~~~~~~~~

---------------------------------------------------------------- (intend)
(C, (h,l), R, intend rt; b) -->_0 (C, (h,l), R[ri -> R(rt)],  b)



a1. |-Z (C, (h,l), R, intend ri rt; b)

1. |- C : P
2. P |-Z R : G
3. |- (h,l) : A
4x. Z = cf  ? . |- Ei =/= l  : |- Ei = l
4y. G(ri) =/= <R,check,Ei> ==> . |- Ei = l
5. .;P;G;A;Ei;to |- intend ri rt; b

6. .;P;G |- intend ri rt : G[ri -> <B,go,Et>                    [ Inversion of (sequence-t), 5, Inspection of (intend-t) ]
5'. .;P;G[ ri <- <B,int,Et>];A;Ei;t |- b                        [ Inversion of (sequence-t), 5, 6 ]

7. G(rt) = <B,All[Dt](Gt,At),Et>                                [ Inversion of (intend-t), 6 ]
8. .;P |-Z R(rt) : <B,All[Dt](Gt,It,At),Et>                     [ Inversion of (R-t), 2, 7 ]

subcase Z = cf
10a. . |- Et : kint                                             [ Canonical Forms, (9, subcase assumption), 1 ]
11a. .;P |-cf R(rt) : <B,int,Et>                                [ (val-t), 10a, 11a ]

subcase Z = B
10b. . |- Et : kint  and R_col(rt) = B                          [ Canonical Forms, (9, subcase assumption), 1 ]
11b. .;P |-B R(rt) : <B,int,Et>                                 [ (val-zap-c-t), 10b ]

subcase Z =/= B and Z =/= cf
10c. . |- R_val(rt) = Et  and R_col(rt) = B                     [ Canonical Forms, (9, subcase assumption), 1 ]
11c. P |- R_val(rt) : int                                       [ (int-t) ]
12c. .;P |-G R(rt) : <B,int,Et>                                 [ (val-t), 10c, 11c ]
```

```
merge
13. .;P |-Z R(rt) : <B,int,Et>                                  [ 11a, 11b, 12c ]
14. Forall r'. .;P |-Z R(r') : G(r')                            [ Inversion of (R-t), 2 ]
2'. P |-Z R[ri -> R(rt)] : G[ri -> <B,int,Et>]                  [ (R-t), 14, 13 ]

15. G(ri) = <ci,ok,Ei>                                          [ Inversion of (intend-t), 6 ]
4y'. . |- Ei = l                                                [ 4y, 15 ]

16. |-Z (C, (h,l), R[ri -> R(rt)],  b)                          [ (S-t), 1, 2', 3, 4x, 4y', 5' ]
* INTEND complete




CASE INTENDZ-SET:
~~~~~~~~~~~~~~~~~

R_val(rz) = 0
-------------------------------------------------------------------- (intendz-set)
(C, (h,l), R, intendz rz rt; b) -->_0 (C, (h,l), R[ri -> R(rt)],  b)



a1. |-Z (C, (h,l), R, intendz rz rt; b)

1. |- C : P
2. P |-Z R : G
3. |- (h,l) : A
4x. Z = cf  ? . |- Ei =/= l  : |- Ei = l
4y. G(ri) =/= <R,check,Ei> ==> . |- Ei = l
5. .;P;G;A;Ei;to |-  intendz rz ri rt; b

6. .;P;G |- intendz rz ri rt : G'[ri <- <B,int,Ez?Ei:Et>]       [ Inversion of (sequence-t), 5, Inspection of (intendz-t) ]
5'. .; P; G[ri -> <B,int,Ez?Ei:Et>]; seq~>Ez?:Et; A |- b        [ Inversion of (sequence-t), 5, 6 ]

7. G(rt) = <B,All[Dt](Gt,It,At),Et>                             [ Inversion of (intendz-t), 6 ]
8. G(rz) = <B,int,Ez>
9. G(ri) = <B,go,Ei>

11. .;P |-Z R(rt) : <B,All[Dt](Gt,At),Et>                       [ Inversion of (R-t), 2, 7 ]
12. .;P |-Z R(rz) : <B,int,Ez>                                  [ Inversion of (R-t), 2, 8 ]
13. .;P |-Z R(ri) : <B,go,Ei>                                   [ Inversion of (R-t), 2, 9 ]

4y'. . |- Ei = l                                                [ 4y, 9 ]

subcase Z =/= B and Z =/= cf

14a. . |- R_val(rz) = Ez                                        [ Canonical Forms, 12, subcase assumption, 1 ]
15a. . |- Ez = 0                                                [ Exp Eq Transitivity Lemma, p1, 14a ]
16a. . |- Ez?Ei:Et = Et                                         [ Exp Conditional Lemma, 15a ]
17a. R_col(rt) = B  and  . |- R_val(rt) = Et                    [ Canonical Forms, 11, subcase assumption, 1 ]
18a. . |- R_val(rt) = Ez?Ei:Et                                  [ Exp Eq Transitivity, 16a, 17a ]
19a. P |- R_val(rt) : go                                        [ (rit-t) ]
20a. .;P |-Z R(rt) : <B,go,Ez?Ei:Et>                            [ (val-t), 17a, 18a, 19a ]


subcase Z = B

14b. R_col(rt) = B  and . |- Et : kint                          [ Canonical Forms, 11, subcase assumption, 1 ]
15b. . |- Ez : kint                                             [ Canonical Forms, 10, 1 ]
16b. . |- Ei : kint                                             [ Canonical Forms, 11, 1 ]
17b. . |- Ez?Ei:Et : kint                                       [ (wf-ifexp, 14b, 15b, 16b ]
18b. .;P |-B R(rt) : <B,go,Ez?Ei:Et>                            [ (val-zap-c-t), 14b, 17b ]


subcase Z = cf
14c. . |- Ei =/= l                                              [ 4x, Z=cf ]
15c. contradiction -- subcase does not apply                    [ 4y', 15c ]

merge
21. .;P |-Z R(rt) : <B,go,Ez?Ei:Et>                             [ 20a, 18b, 15c ]
2'. P |-Z R[ri -> R(rt)]  :  G[ri -> <B,go,Ez?Ei:Et>]           [ (R-t), (Inversion of (R-t), 2), 27 ]


22. |-Z (C, (h,l), R[ri -> R(rt)],  b)                          [ (S-t), 1, 2', 3, 4x, 4y', 5' ]
* INTENDZ-SET complete





CASE INTENDZ-UNSET:
~~~~~~~~~~~~~~~~~~~

R_val(rz) =/= 0
```

```
------------------------------------------------- (intendz-unset)
(C, (h,l), R, intendz rz rt; b) -->_0 (C, (h,l), R, b)


a1. |-Z (C, (h,l), R, intendz rz rt; b)

1. |- C : P
2. P |-Z R : G
3. |- (h,l) : A
4x. Z = cf  ? . |- Ei =/= l  : |- Ei = l
4y. G(ri) =/= <R,check,Ei> ==> . |- Ei = l
5. .;P;G;A;Ei;to |-  intendz rz ri rt; b

6. .;P;G |- intendz rz ri rt : G'[ri <- <B,int,Ez?Ei:Et>]       [ Inversion of (sequence-t), 5, Inspection of (intendz-t) ]
5'. .; P; G[ri -> <B,int,Ez?Ei:Et>]; seq~>Ez?:Et; A |- b        [ Inversion of (sequence-t), 5, 6 ]

7. G(rt) = <B,All[Dt](Gt,It,At),Et>                             [ Inversion of (intendz-t), 6 ]
8. G(rz) = <B,int,Ez>
9. G(ri) = <B,go,Ei>

11. .;P |-Z R(rt) : <B,All[Dt](Gt,At),Et>                       [ Inversion of (R-t), 2, 7 ]
12. .;P |-Z R(rz) : <B,int,Ez>                                  [ Inversion of (R-t), 2, 8 ]
13. .;P |-Z R(ri) : <B,go,Ei>                                   [ Inversion of (R-t), 2, 9 ]

4y'. . |- Ei = l                                                [ 4y, 9 ]

subcase Z =/= B and Z =/= cf

14a. . |- R_val(rz) = Ez                                        [ Canonical Forms, 12, subcase assumption, 1 ]
15a. . |- Ez =/0 0                                              [ Exp Eq Transitivity Lemma, p1, 14a ]
16a. . |- Ez?Ei:Et = Ei                                         [ Exp Conditional Lemma, 15a ]
17a. R_col(ri) = B  and  . |- R_val(ri) = Ei                    [ Canonical Forms, 13, subcase assumption, 1 ]
18a. . |- R_val(rt) = Ez?Ei:Et                                  [ Exp Eq Transitivity, 16a, 17a ]
19a. P |- R_val(rt) : goz                                       [ (rit-t) ]
20a. .;P |-Z R(rt) : <B,goz,Ez?Ei:Et>                           [ (val-t), 17a, 18a, 19a ]


subcase Z = B

14b. R_col(ri) = B  and . |- Ei:kint                            [ Canonical Forms, 13, subcase assumption, 1 ]
15b. . |- Ez : kint                                             [ Canonical Forms, 12, 1 ]
16b. . |- Et : kint                                             [ Canonical Forms, 11, 1 ]
17b. . |- Ez?Ei:Et : kint                                       [ (wf-ifexp, 14b, 15b, 16b ]
18b. .;P |-B R(ri) : <B,goz,Ez?Ei:Et>                           [ (val-zap-c-t), 14b, 17b ]


subcase Z = cf

14c. . |- Ei =/= l                                          [ 4x, Z=cf ]
15c. contradiction -- subcase does not apply                [ 4y', 15c ]


merge

24. .;P |-Z R(ri) : <B,goz,Ez?Ei:Et>                        [ 23a, 18b, 15c ]
2'. P |-Z R :  G[ri -> <B,goz,Ez?Ei:Et>]                    [ (R-t), (Inversion of (R-t), 2), 24 ]

15. |-Z (C, (h,l), R, b)                                    [ (S-t), 1, 2', 3, 4x, 4y', 5' ]
* INTENDZ-UNSET complete






CASE RECOVERNZ-OK:
~~~~~~~~~~~~~~~~~~

R_val(rz) = 0
---------------------------------------------------- (recovernz-ok)
(C, (h,l), R, recovernz rz; b) -->_0 (C, (h,l), R, b)


a1. |-Z (C, (h,l), R, recovernz rz; b)

1. |- C : P
2. P |-Z R : G
3. |- (h,l) : A
4x. Z = cf  ? . |- Ei =/= l  :  . |- Ei = l
4y. G(ri) =/= <R,check,Ei> ==> . |- Ei = l
5. .;P;G;A;Ei;to |- recovernz rz; b

subcase on the structure of 5 - (recovernz-t) does not apply as D is empty


SUBCASE RECOVERNZ-OK.A: using (recovernz-eq-t)
```

```
G(rz) = <R,int,Ez>
G(ri) = <R,check,Eli>
. |- Ez = Eli - Ela
. |- Eli = Ela
.; P; G[ri -> <R,ok,Eli>]; seq o Ela; Eli; to |- b
------------------------------------------------- (recovernz-eq-t)
.; P; G; seq o Ela; Eli; to |- recovernz rz ; b


6a. .;P |-Z R_val(ri) : <R,check,Eli>                          [ Inversion of (R-t), 2, pa2 ]
7a. .;P |-Z R_val(ri) : <R,ok,Eli>                             [ (val-t), Inversion/Reconstruction of val-t rules, using
                                                                   (rit-t) for (val-t) case ]
2'. P |-Z R : G[ri -> <R,ok,Eli>]                              [ (R-t), 2, 7a ]

5'. .; P; G[ri -> <R,ok,Eli>]; seq o Ela; Eli |- b             [ pa5 ]

8a.  . |- Ela = l                                              [ Inversion of (h-append-t), 3 ]
4y'. . |- Eli = l                                              [ Exp Eq Transitivity, p4, 8a ]

8a. |-Z (C, (h,l), R, b)                                       [ (S-t), 1, 2', 3, 4x, 4y', 5' ]
* RECOVERNZ-OK.A complete


SUBCASE B: using (recovernz-neq-t)

G(rz) = <R,int,Ez>
G(ri) = <R,check,Eli>
. |- Ez = Eli - Ela
. |- Eli =/= Ela
------------------------------------------------- (recovernz-neq-t)
.; P; G; seq o Ela; Eli; to |- recovernz rz; b

subsubcase B1: Z =/= R
6b1. . |- R_val(rz) = Ez                                       [ Canonical Forms, (Inversion of (R-t), 2, pb1), 1 ]
7b1. . |- Ez = 0                                               [ Exp Eq Transitivity, 6b1, p1 ]
8b1. . |- Eli = Ela                                            [ pb3, 7b1 ]
9b1. 8b1 contradicts p4. subsubcase doesn't apply
subsubcase B2: Z = R
6b2. . |- Ela = l                                              [ Inversion of (h-append-t), 3 ]
7b2. . |- Eli = l                                              [ 4x, subcase assumption ]
8b2. . |- Eli = Ela                                            [ Exp Eq Transitivity, 7b2, 8b2 ]
9b2. 8b2 contradicts p4. subsubcase doesn't apply
* RECOVERNZ-OK.B complete




CASE RECOVERNZ-HALT:
~~~~~~~~~~~~~~~~~~~~

R_val(rz) =/= 0
---------------------------------------------- (recovernz-halt)
(C, h, R, recovernz rz; b) -->_0 recover

recover(h) =/= S'
case does not apply
* RECOVERNZ-HALT complete




CASES BRZ-UNTAKEN, BRZ-TAKEN, BRZ-HW-ERROR:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Do not apply (i = brz)




CASES JMP, JMP-HW-ERROR:
~~~~~~~~~~~~~~~~~~~~~~~~

Do not apply (i = jmp).



** LEMMA PRESERVATION-NO-ELEVATION complete



================================================================================================================
```

```
==================================================================================================================




*************************************
* LEMMA PRESERVATION-JMP-BRZ-EMPTY-Z *
*************************************

If |- S  and  and b = brz or b = jmp  and  S -->_0 S'  then  |- S'



PROOF: By case analysis of S -->_0 S'
******




CASES MOVI, SUB, INTEND, INTENDZ-SET, INTENDZ-UNSET, RECOVERNZ-OK, RECOVERNZ-HALT:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

a1. |- S
a2. S -->_0 S'

1. i =/= jmp or brz                     [ inspection of rules ]
* CASES MOVI, SUB, INTEND, INTENDZ-SET, INTENDZ-UNSET, RECOVERNZ-OK, RECOVERNZ-HALT complete







CASE BRZ-UNTAKEN:
~~~~~~~~~~~~~~~~~

R_val(rz) =/= 0     l+1 in Dom(C)
-------------------------------------------------------------------------------- (brz-untaken)
(C, (h,l), R, brz rz rt) -->_0 (C, (h,l,l+1), R[ri -> R R_val(ri)], C(l+1))

a1. |- (C, (h,l), R, brz rz rt)

1. |- C : P
2. P |- R : G
3. |- (h,l) : A
4x. . |- Ei = l
4y. G(ri) =/= <R,check,Ei> ==> . |- Ei = l

5. .;P;G;A;Ei;P(l+1) |- brz rz rt

6. .;P;G;seq o Ela; Ei; All[Df](Gf,Af) |- brz rz rt             [ Inspection of (brz-t), 5 ]
7. G(ri)= <B,goz,Ez'?Ef':Et'>                                    [ Inversion of (brz-t), 6 ]
8. . |- Ef' = Ela + 1
9. G(rz) = <G,int,Ez>
10. . |- Ez = Ez'
11. Exists Sf. . |- Sf : Df
12. . |- G[ri -> <R,check,Ef'>] <= Sf(Gf)
13. . |- seq o Ela o Ef' = Sf(Af)


14. Forall r'. .;P |- R(r') : G(r')                              [ Inversion of (R-t), 2 ]
15. .;P |- R(ri) : <B,goz,Ez'?Ef':Et'>                           [ 14, 7 ]
16. . |- R_val(ri) = Ez'?Ef':Et'                                 [ Inversion of (val-t), 15 ]
17. P |- R_val(ri) : check                                       [ (rit-t) ]

18. .;P |- R(rz) : <G,int,Ez>                                    [ 14, 9 ]
19. . |- R_val(rz) = Ez                                          [ Inversion of (val-t), 18 ]
20. . |- Ez' =/= 0                                               [ Exp Eq Transitivity, 19, p1, 10 ]
21. . |- Ez'?Ef':Et' = Ef'                                       [ Exp Conditional Lemma, 20 ]
22. . |- R_val(ri) = Ef'                                         [ Exp Eq Transitivity, 16, 21 ]

23. .;P |- R R_val(ri) : <R,check,Ef'>                           [ (val-t), 17, 22 ]
24. P |- R[ri -> R R_val(ri)] : G[ri -> <R,check,Ef'>]           [ (R-t), 14, 23 ]
2'. P |- R[ri -> R R_val(ri)] : Sf(Gf)                           [ Repeated applications of Subtyping Lemma, 12, 24 ]

25. . |- l = Ela                                                 [ Inversion of (h-append), 3, 6 ]
26. . |- Ef' = l + 1                                             [ Exp Eq Transitivity, 8, 25 ]
27. . |- (h,l,l+1) : seq o Ela o Ef'                             [ (h-append), 3, 26 ]
3'. . |- (h,l,l+1) : Sf(Af)                                      [ Exp Eq Transitivity, 27, 12 ]

28. P(l+1) = All[Df](Gf,Af)                                      [ Inspection of 5, 6 ]
29. Df = (Df', Y:kint, alpha:kseq)  and  Af = alpha o (l+1)      [ Inversion of (C-t), 1, p2, 28 ]
30. Gf(ri) = <R,check,Y>
31. Df; P; Gf; Af; Y; P(l+2) |- C(l+1)
5'. .; P; Sf(Gf); Sf(Af); Sf(Y); Sf(P(l+2)) |- C(l+1)           [ Substitution Lemma, 11, 31 ]

32. . |- <R,check,Ef'> <= Sf(<R,check,Y>)                        [ 12, 30 ]
```

```
33. . |- Ef' = Sf(Y)                                                [ Inversion of (subtp-reflex), 32 ]
4'. . |- Sf(Y) = l+1                                                [ Exp Eq Transitivity, 26, 33 ]

34: |- (C, (h,l,l+1), R[ri -> R R_val(ri)], C(l+1))                [ (S-t), 1, 2', 3', 4', 4', 5' ]
* BRZ-UNTAKEN complete
```

```
CASE BRZ-TAKEN:
~~~~~~~~~~~~~~~

R_val(rz) = 0   R_val(rt) in Dom(C)
--------------------------------------------------------------------------------------------- (brz-taken)
(C, (h,l), R, brz rz rt) -->_0 (C, (h,l,R_val(rt)), R[ri -> R R_val(ri)], C(R_val(rt)))

a1. |- (C, (h,l), R, brz rz rt)

1. |- C : P
2. P |- R : G
3. |- (h,l) : A
4x. |- Ei = l
4y. G(ri) =/= <R,check,Ei> ==> . |- Ei = l
5. .;P;G;A;Ei;P(l+1) |- brz rz rt

6. .;P;G;seq o Ela; Ei; All[Df](Gf,Af) |- brz rz rt         [ Inspection of (brz-t), 5 ]
7. G(ri)= <B,goz,Ez'?Ef':Et'>                               [ Inversion of (brz-t), 6 ]
8. G(rz) = <G,int,Ez>
9. . |- Ez = Ez'
10. G(rt) = <G,All[Dt](Gt,At),Et>
11. . |- Et = Et'
12. Exists St. D |- St : Dt
13. . |- G[ri -> <R,check,Et'>] <= St(Gt)
14. . |- seq o Ela o Et' = St(At)

15. Forall r'. .;P |- R(r') : G(r')                         [ Inversion of (R-t), 2 ]
16. .;P |- R(ri) : <B,goz,Ez'?Ef':Et'>                      [ 15, 7 ]
17. . |- R_val(ri) = Ez'?Ef':Et'                            [ Inversion of (val-t), 16 ]
18. P |- R_val(ri) : check                                  [ (rit-t) ]

19. .;P |- R(rz) : <G,int,Ez>                               [ 15, 8 ]
20. . |- R_val(rz) = Ez                                     [ Inversion of (val-t), 19 ]
21. . |- Ez' = 0                                            [ Exp Eq Transitivity, 20, p1, 9 ]
22. . |- Ez'?Ef':Et' = Et'                                  [ Exp Conditional Lemma, 21 ]
23. . |- R_val(ri) = Et'                                    [ Exp Eq Transitivity, 17, 22 ]

24. .;P |- R R_val(ri) : <R,check,Et'>                      [ (val-t), 18, 23 ]
25. P |- R[ri -> R R_val(ri)] : G[ri -> <R,check,Et'>]      [ (R-t), 15, 24 ]
2'. P |- R[ri -> R R_val(ri)] : St(Gt)                      [ Repeated applications of Subtyping Lemma, 13, 24 ]

26. .;P |- rt : <G,All[Dt](Gt,At),Et>                       [ 15, 10 ]
27. . |- R_val(rt) = Et                                     [ Inversion of (val-t), 26 ]
28. . |- (h,l,R_val(rt)) : seq o Ela o Et'                  [ (h-append), 3, 27 ]
3'. . |- (h,l,R_val(rt)) : St(At)                           [ Exp Eq Transitivity, 28, 14 ]

29. Dt = (Dt', Y:kint, alpha:kseq)  and  At = alpha o R_val(rt) [ Inversion of (C-t), 1, p2, (Inversion of (val-t), 26)  ]
30. Gt(ri) = <R,check,Y>
31. Dt; P; Gt; At; Y; P(R_val(rt)+1) |- C(R_val(rt))
5'. .;P;St(Gt); St(At); St(Y); St(R_val(rt)+1) |- C(R_val(rt))  [ Substitution Lemma, 31, 12 ]

32. . |- <R,check,Et'> <= St(<R,check,Y>)                   [ 13, 30 ]
33. . |- Et' = St(Y)                                        [ Inversion of (subtp-reflex), 32 ]
4'. . |- St(Y) = R_val(rt)                                  [ Exp Eq Transitivity, 33, 27, 11 ]

34: |- (C,(h,l,R_val(rt)),R[ri -> R R_val(ri)],C(R_val(rt)))    [ (S-t), 1, 2', 3', 4', 4', 5' ]
* BRZ-TAKEN complete
```

```
CASE JMP:
~~~~~~~~~

R_val(r) in Dom(C)
-------------------------------------------------------------------------------------- (jmp)
(C, h, R, jmp rt) -->_0 (C, (h,l,R_val(rt)), R[ri -> R R_val(ri)], C(R_val(rt)))

a1. |- (C, (h,l), R, jmp rt)

1. |- C : P
2. P |- R : G
3. |- (h,l) : A
4x. . |- Ei = l
```

```
4y. G(ri) =/= <R,check,Ei> ==> . |- Ei = l

5. .;P;G;A;Ei;P(l+1) |- jmp rt

6. .; P; G; seq o Ela; Ei; t |- jmp rt                      [ Inspection of (jmp-t), 5 ]
7. G(ri)= <B,go,Et'>                                        [ Inversion of (jmp-t), 6 ]
8. G(rt) = <G,All[Dt](Gt,Rt),Et>
9. . |- Et = Et'
10. Exists S. D |- St : Dt
11. . |- G[ri -> <R,check,Et'>] <= St(Gt)
12. . |- seq o Ela o Et = St(At)


13. Forall r'. .;P |- R(r') : G(r')                         [ Inversion of (R-t), 2 ]
14. .;P |- R(ri) : <B,go,Et'>                                   [ 13, 7 ]
15. . |- R_val(ri) = Et'                                    [ Inversion of (val-t), 14 ]
16. P |- R_val(ri) : check                                  [ (rit-t) ]

17. .;P |- R R_val(ri) : <R,check,Et'>                      [ (val-t), 15, 16 ]
18. P |- R[ri -> R R_val(ri)] : G[ri -> <R,check,Et'>]      [ (R-t), 13, 17 ]
2'. P |- R[ri -> R R_val(ri)] : St(Gt)                      [ Repeated applications of Subtyping Lemma, 11, 18 ]

19. .;P |- rt : <G,All[Dt](Gt,At),Et>                       [ 13, 8 ]
20. . |- R_val(rt) = Et                                     [ Inversion of (val-t), 19 ]
21. . |- (h,l,R_val(rt)) : seq o Ela o Et'                  [ (h-append), 3, 20 ]
3'. . |- (h,l,R_val(rt)) : St(At)                           [ Exp Eq Transitivity, 21, 12 ]

22. Dt = (Dt', Y:kint, alpha:kseq)  and  At = alpha o R_val(rt) [ Inversion of (C-t), 1, p1, (Inversion of (val-t), 19)  ]
23. Gt(ri) = <R,check,Y>
24. Dt; P; Gt; At; Y; P(R_val(rt)+1) |- C(R_val(rt))
5'. .;P;St(Gt); St(At); St(Y); St(R_val(rt)+1) |- C(R_val(rt))  [ Substitution Lemma, 24, 10 ]

25. . |- <R,check,Et'> <= St(<R,check,Y>)                   [ 11, 23 ]
26. . |- Et' = St(Y)                                        [ Inversion of (subtp-reflex), 25 ]
4'. . |- St(Y) = R_val(rt)                                  [ Exp Eq Transitivity, 26, 9, 20 ]

34: |- (C, (h,l,R_val(rt)), R[ri -> R R_val(ri)], C(R_val(rt))) [ (S-t), 1, 2', 3', 4', 4', 5' ]
* JMP complete





CASES BRZ-HW-ERROR, JMP-HW-ERROR:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

R_val(rz) = 0    R_val(rz) not in Dom(C)
------------------------------------------------------------ (brz-hw-error)
(C, h, R, brz rz rd) -->_0 hw-error

R_val(r) not in Dom(C)
---------------------------------------------------------------- (jmp-hw-error)
(C, h, R, jmp r) -->_0 hw-error

(C,h,R,b) --/-->_0 S
cases do not apply
* BRZ-HW-ERROR, JMP-HW-ERROR complete.



** PRESERVATION-JMP-BRZ-EMPTY-Z COMPLETE






==============================================================================================================
==============================================================================================================




*********************************************
* LEMMA PRESERVATION-BRZ-POSSIBLE-ELEVATION *
*********************************************

If |-f (C, (h,l), R, brz rz rt)
   and  G is regfile type used
   and  G(rz) = <G,int,Ez>
   and  G(ri) = <B,goz,Ez'?Ef':Et'>
   and  G(rt) = <G,All[Dt](Gt,At),Et>
   and  |- h : seq o Ela
   and  (C,(h,l),R,brz rz rt) -->_0 S'
```

```
then
    (1) f = R   ==>   |-R  S'

    (2) f =/= cf

    (3) f = G and
        (a) R_val(rz) =/= 0 and . |- Ez =/= 0                              ==> |-G  S' // correct fallthru (
                                                                                             (rz correct)
        (b) R_val(rz) =/= 0 and . |- Et = Ela + 1                          ==> |-G  S' // correct fallthru
                                                                                             (branch/fallthru targets are equal)
        (a) R_val(rz) =/= 0 and . |- Ez = 0     and . |- Et =/= Ela + 1    ==> |-cf S' // incorrect fallthru
                                                                                             (that doesn't accidentally work out)
        (c) R_val(rz)  =  0 and . |- Ez = 0     and . |- R_val(rt) = Et    ==> |-G  S' // correct branch
        (c) R_val(rz)  =  0 and . |- Ez = 0     and . |- R_val(rt) =/= Et       ==> |-cf S' // incorrect branch (rt corrupt)
        (d) R_val(rz)  =  0 and . |- Ez =/= 0   and . |- R_val(rt) = Ela + 1    ==> |-G  S' // correct branch (two wrongs make a right)
        (b) R_val(rz)  =  0 and . |- Ez =/= 0   and . |- R_val(rt) =/= Ela + 1  ==> |-cf S' // incorrect branch
                                                                                                 (rz corrupt, doesn't accidentally work out)


    (5) f = B and
        (a) R_val(rz) =/= 0 and     . |- R_val(ri) = Ef'    ==>  |-B  S'    // correct fallthru
        (b) R_val(rz) =/= 0 and     . |- R_val(ri) =/= Ef'  ==>  |-cf S'    // incorrect faullthru
        (c) R_val(rz)  =  0 and     . |- R_val(ri) = Et'    ==>  |-B S'     // correct branch
        (d) R_val(rz)  =  0 and     . |- R_val(ri) =/= Et'  ==>  |-cf S'    // incorrect branch


PROOF: By case analysis of S -->_0 S'
******



CASES MOVI, SUB, INTEND, INTENDZ-SET, INTENDZ-UNSET, RECOVERNZ-OK, RECOVERNZ-HALT, JMP, JMP-HW-ERROR:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

a1. |-f S
a2. S -->_0 S'

1. i =/=  brz                            [ inspection of rules ]
* CASES MOVI, SUB, INTEND, INTENDZ-SET, INTENDZ-UNSET, RECOVERNZ-OK, RECOVERNZ-HALT, JMP, JMP-HW-ERROR complete






CASE BRZ-UNTAKEN:
~~~~~~~~~~~~~~~~~

R_val(rz) =/= 0     l+1 in Dom(C)
--------------------------------------------------------------------------------- (brz-untaken)
(C, (h,l), R, brz rz rt) -->_0 (C, (h,l,l+1), R[ri -> R R_val(ri)], C(l+1))

a1. |-f (C, (h,l), R, brz rz rt)

1. |- C : P
2. P |-f R : G
3. |- (h,l) : A
4x. f = cf  ? . |- Ei =/= l  :  . |- Ei = l
4y. G(ri) =/= <R,check,Ei> ==> . |- Ei = l
5. .;P;G;A;Ei;P(l+1) |- brz rz rt

6. .;P;G;seq o Ela; Ei; All[Df](Gf,Af) |- brz rz rt             [ Inspection of (brz-t), 5 ]
7. G(ri)= <B,goz,Ez'?Ef':Et'>                                   [ Inversion of (brz-t), 6 ]
8. . |- Ef' = Ela + 1
9. G(rz) = <G,int,Ez>
10. . |- Ez = Ez' and . |- Et = Et'
11. Exists Sf. . |- Sf : Df
12. . |- G[ri -> <R,check,Ef'>] <= Sf(Gf)
13. . |- seq o Ela o Ef' = Sf(Af)

14. Forall r'. .;P |-f R(r') : G(r')                                [ Inversion of (R-t), 2 ]
15. .;P |-f R(ri) : <B,goz,Ez'?Ef':Et'>                             [ 15, 7 ]
16. .;P |-f R(rz) : <G,int,Ez>                                      [ 15, 9 ]

18. P(l+1) = All[Df](Gf,Af)                                         [ Inspection of 5, 6 ]
19. Df = (Df', Y:kint, alpha:kseq)  and  Af = alpha o (l+1)         [ Inversion of (C-t), 1, p2, 18 ]
20. Gf = Gf', ri-> <R,check,Y>
21. Df; P; Gf; Af; Y; P(l+2) |- C(l+1)
22. Df' |- Gf' ok  /\ Df' |- P(l+2) ok

23. Sf = Ealpha/alpha, EY/Y, Sf'/Df'  and . |- Sf':Df'              [ Substitution Structure Lemma, 12, 19 ]
d1. Let Sfb = (seq o Ela)/alpha, R_val(ri)/Y, Sf'/Df'               [ definition ]
24. . |- Sfb : Df                                                   [ (subst-t), 23, d1 ]

6'. .; P; Sfb(Gf); Sfb(Af); Sfb(Y); Sfb(P(l+2)) |- C(l+1)           [ Substitution Lemma, 24, 21 ]
```

```
25. Forall r' =/= ri. . |- G(r') <= Sf(Gf)(r')                       [ Inversion of (G-subtp), 13 ]
26. Forall r' =/= ri. . |- G(r') <= Sfb(Gf)(r')                      [ 25, 23, d1, 20, 22 ]
27. Forall r' =/= ri.                                                [ Repeated Application of the Subtyping lemma, 14, 26 ]
     .;P |-f R[ri -> R R_val(ri)](r') : Sfb(Gf)(r')

28. P |- R_val(ri) : check                                           [ (rit-t) ]
29. .;P |-f  R R_val(ri) : <R,check,R_val(ri)>                       [ (val-t), 28 ]
30. Sfb(Gf)(ri) = <R,check,R_val(ri)>                                [ 20, d1 ]
31. .;P |-f R[ri -> R R_val(ri)](ri) : Sfb(Gf)(ri)                   [ 29, 30 ]
2''. P |-f  R[ri -> R R_val(ri)] : Sfb(Gf)                           [ (R-t), 27, 31 ]

32. |- (h,l,l+1) : seq o Ela o l+1                                   [ (h-append), 3 ]
33. |- (h,l,l+1) : Sfb(alpha o l+1)                                  [ 32, d1 ]
3'. |- (h,l,l+1) : Sfb(Af)                                           [ 33, 19 ]

subcase f = R :
34a. . |- R_val(ri) = Ez'?Ef':Et'                                    [ Inversion of (val-t), f = R, 15 ]
35a. . |- R_val(rz) = Ez                                             [ Inversion of (val-t), f = R, 16 ]
36a. . |- Ez' =/= 0                                                  [ Exp Eq Transitivity, 11, 35a, p1 ]
37a. . |- Ez'?Ef':Et' = Ef'                                          [ Exp Conditional Lemma, 36a ]
38a. . |- R_val(ri) = Ef'                                            [ Exp Eq Transitivity, 34a, 37a ]
39a. . |- Sfb(Y) = Ef'                                               [ 38a, d1 ]
40a. . |- Ela = l                                                    [ Inversion of (h-append), 3 ]
4a'. . |- Sfb(Y) = l + 1                                             [ Exp Eq Transitivity, 39a, 8, 40a ]
2a'.  P |-R  R[ri -> R R_val(ri)] : Sfb(Gf)                          [ 2'', f = R ]

subcase f = G and . |- Ez =/= 0 : green corruption does not affect branch direction ==> no fault elevation
34b. . |- R_val(ri) = Ez'?Ef':Et'                                    [ Inversion of (val-t), f = G, 15 ]
35b. . |- Ez' =/= 0                                                  [ Exp Eq Transitivity, 10, subcase assumption ]
36b. . |- Ez'?Ef':Et' = Ef'                                          [ Exp Conditional Lemma, 35b ]
37b. . |- R_val(ri) = Ef'                                            [ Exp Eq Transitivity, 34b, 36b ]
38b. . |- Sfb(Y) = Ef'                                               [ 37b, d1 ]
39b. . |- Ela = l                                                    [ Inversion of (h-append), 3 ]
4b'. . |- Sfb(Y) = l + 1                                             [ Exp Eq Transitivity, 38b, 8, 39b ]
2b'.  P |-G  R[ri -> R R_val(ri)] : Sfb(Gf)                          [ 2'', f = G ]

subcase f = G and . |- Et = Ela + 1 : branch and fallthru are equal ==> no fault elevation
34c. . |- R_val(ri) = Ez'?Ef':Et'                                    [ Inversion of (val-t), 15 ]
35c. . |- Et' = Ef'                                                  [ Exp Eq Transitivity, subcase assumption, 8 ]
36c. . |- R_val(ri) = Ef'                                            [ 34c, 35c ]
37c. . |- Sfb(Y) = Ef'                                               [ 356c, d1 ]
38c. . |- Ela = l                                                    [ Inversion of (h-append), 3 ]
4c'. . |- Sfb(Y) = l + 1                                             [ Exp Eq Transitivity, 37b, 8, 38b ]
2c'.  P |-G  R[ri -> R R_val(ri)] : Sfb(Gf)                          [ 2'', f = G ]

subcase f = G and . |- Ez = 0  and . |- Et =/= Ela + 1 : wrong branch taken ==> elevate Z to cf fault
34d. . |-G R_val(ri) = Ez'?Ef':Et'                                   [ Inversion of (val-t), f = G, 15 ]
35d. . |- Ez' = 0                                                    [ Exp Eq Transitivity, 10, subcase assumption ]
36d. . |- Ez'?Ef':Et' = Et'                                          [ Exp Conditional Lemma, 35d ]
37d. . |- R_val(ri) = Et                                             [ Exp Eq Transitivity, 34d, 36d, 10 ]
38d. . |- Sfb(Y) = Et                                                [ Exp Eq Transitivity, 37d, d1 ]
39d. . |- Ela = l                                                    [ Inversion of (h-append), 3 ]
40d. . |- Et =/= l + 1                                               [ Exp Eq Transitivity, subcase assumption, 39d ]
4d'. . |- Sfb(Y) =/= l+1                                             [ Exp Eq Transitivity, 40d, 39d ]
2d'.  P |-cf  R[ri -> R R_val(ri)] : Sfb(Gf)                         [ Repeated applications of Color Weakening Lemma, 2'' ]

subcase f = B and . |- R_val(ri) = Ef': intention set correctly ==> no fault elevation
34e. . |- Ela = l                                                    [ Inversion of (h-append), 3 ]
35e. . |- Ef' = l + 1                                                [ Exp Eq Transitivity, 34e, 8 ]
36e. . |- R_val(ri) = l+1                                            [ Exp Eq Transitivity, 35e, subcase assumption ]
4e'. . |- Sfb(Y) = l+1                                               [ 36e, d1 ]
2e'. P |-B R[ri -> R R_val(ri)] : Sfb(Gf)                            [ 2'', f = B ]

subcase f = B and . |- R_val(ri) =/= Ef': intention set wrong ==> elevate to cf fault
34f. . |- Ela = l                                                    [ Inversion of (h-append), 3 ]
35f. . |- Ef' = l + 1                                                [ Exp Eq Transitivity, 34f, 8 ]
36f. . |- R_val(ri) =/= l+1                                          [ Exp Eq Transitivity, 35f, subcase assumption ]
4f'. . |- Sfb(Y) =/= l+1                                             [ 36f, d1 ]
2f'. P |-B R[ri -> R R_val(ri)] : Sfb(Gf)                            [ Repeated applications of Color Weakening Lemma, 2'' ]

subcase f = cf:
34g. . |- Ei =/= l                                                   [ 4x, subcase assumption ]
35g. . |- Ei = l                                                     [ 5x, 7 ]
36g. contradiction, subcase doesn't apply                            [ 34g, 35g ]

merge:
4x'. f' = cf ? . |- Ei =/= l  : . |- Ei = l                          [ 4a', 4b', 4c', 4d', 4e', 4f', 36g ]
2'.  P |-f' R[ri -> R R_val(ri)] : Sfb(Gf)                           [ 2a', 2b', 2c', 2d', 2e', 2f', 36g ]

4y'. Gf(ri) = <R,check,Ei>

43. |-f (C, (h,l,l+1), R[ri -> R R_val(ri)], C(l+1))                 [ (S-t), 1, 2', 3', 4x', 4y', 5' ]
* BRZ-UNTAKEN complete
```

```
CASE BRZ-TAKEN:
~~~~~~~~~~~~~~~

R_val(rz) = 0   R_val(rt) in Dom(C)
------------------------------------------------------------------------------------------ (brz-taken)
(C, (h,l), R, brz rz rt) -->_0 (C, (h,l,R_val(rt)), R[ri -> R R_val(ri)], C(R_val(rt)))


a1. |-f (C, (h,l), R, brz rz rt)

1. |- C : P
2. P |-f R : G
3. |- (h,l) : A
4. f = cf  ==> . |- Ei =/= l  : . |- Ei = l
5. G(ri) =/= <R,check,Ei> ==> . |- Ei = l
6. .;P;G;A;Ei;P(l+1) |- brz rz rt

7. .;P;G;seq o Ela; Ei; All[Df](Gf,Af) |- brz rz rt                 [ Inspection of (brz-t), 6 ]
8. G(ri)= <B,goz,Ez'?Ef':Et'>                                       [ Inversion of (brz-t), 7 ]
9. G(rz) = <G,int,Ez>
10. . |- Ez = Ez'  and  . |- Ef' = Ela + 1
11. G(rt) = <G,All[Dt](Gt,At),Et>
12. . |- Et = Et'
13. Exists St. . |- St : Dt
14. . |- G[ri -> <R,check,Et'>] <= St(Gt)
15. . |- seq o Ela o Et' = St(At)

16. Forall r'. .;P |-f R(r') : G(r')                                [ Inversion of (R-t), 2 ]
17. .;P |-f R(ri) : <B,goz,Ez'?Ef':Et'>                             [ 16, 8 ]
18. .;P |-f R(rz) : <G,int,Ez>                                      [ 16, 9 ]
19. .;P |-f R(rt) : <G,All[Dt](Gt,At),Et>                           [ 16, 11 ]

20.P(R_val(rt)) = All[Dl](Gl,Al)                                    [ Inversion of (C-t), 1, p2 ]
21. Dl = (Dl',Y:kint,alpha:kseq)
22. Gl = (Gl',ri-><R,check,Y>)
23. Al = alpha o R_val(rt)
24. Dl' |- Gl' ok  /\ Forall r' in Dom(Gl'). Gl'(r') =/= <R,t',E'>
25. Dl' |- P(R_val(rt)+1) ok
26. Dl; P; Gl; Al; Y; P(R_val(rt)+1) |- C(R_val(rt))


8 subcases (based on f, further subdivided for f = G and B) divided into groups depending on outcome


GROUP A: Control Flow OK

subcase f = R :
30a1. . |- R_val(rt) = Et                                       [ Inversion of (val-t), f=R, 19 ]
31a1. . |- R_val(rz) = Ez                                       [ Inversion of (val-t), f=R, 18 ]
32a1. . |- R_val(ri) = <Ez'?Ef':Et'>                            [ Inversion of (val-t), f=R, 17 ]
33a1. . |- Ez' = 0                                              [ Exp Eq Transitivity, 31a1, p1, 10 ]
34a1. . |- R_val(ri) = Et'                                      [ Exp Eq Transitivity, 32a1, (Conditional Exp Lemma, 33a1) ]

subcase f = G and . |- Ez = 0 and . |- R_val(rt) = Et: green corruption does not affect branch direction or location ==> no fault elevation
30a2. . |- R_val(rt) = Et                                       [ subcase assumption ]
31a2. . |- Ez'?Ef':Et' = Et'                                    [ Exp Conditional Lemma, (subcase assumption, 10) ]
32a2. . |- R_val(ri) = Ez'?Ef':Et'                              [ Inversion of (val-t), f=G, 17 ]
33a2. . |- R_val(ri) = Et'                                      [ Exp Eq Transitivity, (Exp Conditional Lemma, 31a2), 32a2 ]

subcase f = B and . |- R_val(ri) = Et'
30a3. . |- R_val(rt) = Et                                       [ Inversion of (val-t), f=B, 19 ]
31a3. . |- R_val(ri) = Et'                                      [ subcase assumption ]

merge:
35a. . |- R_val(rt) = Et                                        [ 30a1, 30a2, 30a3 ]
36a. . |- R_val(ri) = Et'                                       [ 34a1, 33a2, 31a3 ]

37a. P(R_val(rt)) = All[Dt](Gt,At)                              [ Equal Code Labels Lemma, (Inversion of (R-t), 2, 11), 35a ]
38a. Dl = Dt, Gl = Gt, Al = At                                  [ 37a, 20 ]
39a. . |- G[ri -> <R,check,Et'>] <= St(Gt)                      [ 14 ]
40a. .;P |-f R R_val(ri) : <R,check,Et'>                        [ (val-t), (rit-t), 36a ]
41a. P |-f R[ri -> R R_val(ri)] : G[ri -> <R,check,Et'>]        [ (R-t), 2, 40a ]
2'. P |-f R[ri -> R R_val(ri)] : St(Gt)                         [ Repeated applications of Subtyping Lemma, 39a, 41a ]

6'. . ; P; St(Gt); St(At); St(Y); St(P(R_val(rt)+1))) |- C(R_val(rt))   [ Substitution Lemma, (13, 38a), 26 ]

42a. |- (h,l,R_val(rt)) : seq o Ela o Et'                       [ (h-append), 3, (Exp Eq Transitivity, 35a, 12) ]
3'. |- (h,l,R_val(rt)) = St(At)                                 [ 42a, 15 ]

43a. . |- <R,check,Et'> <= St(R,check,Y)                        [ 39a, 22 ]
44a. . |- Et' = St(Y)                                           [ Inversion of (subtp-reflex), 43a ]
4'. . |- St(Y) = R_val(rt)                                      [ Exp Eq Transitivity, 44a, 12, 35a ]
```

```
5'. St(Gl)(ri) = St(<R,check,Y>)                                [ 22 ]

45a. |-f (C, (h,l,R_val(rt)), R[ri -> R R_val(ri)], C(R_val(rt)))   [ (S-t), 1, 2', 3', 4', 5', 6' ]
* BRZ-TAKEN.A complete




GROUP B: Control Flow Messed up - elevate to cf fault

d1b. Let Sl' = {42/x | Dl(x) = kint} union {empty/x | Dl(x) = kseq }    [ build a substitution of nonsense values ]
d2b. Let Sl = Sl', (seq o Ela)/alpha, R_val(ri)/Y
29b. . |- Sl : (Dl',alpha:kseq,Y:kint)                              [ (subst-t),  d1b, d2b ]

6'. .;P;Sl(Gl);Sl(Al);Sl(Y);Sl(P(R_val(rt)+1)) |- C(R_val(rt))      [ Substitution Lemma, 29b, 28b ]

30b. Forall r' =/= ri. .;P |-cf R(r') : S(Gl)(r')                   [ (val-zap-cf-t), 24 ]
31b. P |- R_val(ri) : check                                         [ (rit-t) ]
32b. .;P |-cf R R_val(ri) : <R,check,R_val(ri)>                     [ (val-t), 31b ]
33b. .;P |-cf R[ri -> R R_val(ri)](ri) : Sl(Gl)(ri)                 [ 32b, d2b, 22 ]
2': P |-cf R[ri -> R R_val(ri)] : Sl(Gl)                            [ (R-t), 30b, 33b ]

34b. |- (h,l,R_val(rt)) : seq o Ela o R_val(rt)                     [ (h-append-t), 3 ]
3'.  |- (h,l,R_val(rt)) : Sl(Al)                                    [ 34b, d2b, 23 ]

35b. . |- R_val(ri) = Ez'?Ef':Et'                                   [ Inversion of (val-t), 17, f=G ]
36b. . |- Ela = l                                                   [ Inversion of (h-append), 3 ]

subcase: f = G and  . |- Ez =/= 0  and  . |- R_val(rt) =/= Ela + 1 : should have taken fallthru but didn't end up there
                                                                               ==> elevate to cf fault
37b1. . |- R_val(ri) = Ef'                                          [ Conditional Exp Lemma, 35b, (subcase assumption, 10) ]
38b1. . |- R_val(ri) = Ela + 1                                      [ Exp Eq Transitivity, 37b1, 10 ]
39b1. . |- R_val(ri) =/= R_val(rt)                                  [ Exp Eq Transitivity, 38b1, subcase assumption ]

subcase: f = G and . |- Ez = 0  and . |- R_val(rt) =/= Et : should have jumped, but went to wrong place ==> elevate to cf fault
37b2. . |- R_val(ri) = Et                                           [ Conditional Exp Lemma, 35b, (subcase assumption, 12) ]
38b2. . |- R_val(ri) =/= R_val(rt)                                  [ Exp Eq Transitivity, 37b2, subcase assumption ]

subcase: f = B  and  . |- R_val(ri) =/= Et' : intentions think we should have gone elsewhere
37b3. . |- R_val(rt) = Et                                           [ Inversion of (val-t), f=B, 19 ]
38b3. . |- R_val(ri) =/= R_val(rt)                                  [ Exp Eq Transitivity, subcase assumption, 12, 37b3 ]

merge:
40b. . |- R_val(ri) =/= R_val(rt)                                   [ 39b1, 38b2, 37b3 ]
4'. . |- Sl(Y) =/= R_val(rt)                                        [ 40b, db2 ]

5'. Sl(Gl)(ri) = Sl(<R,check,Y>)                                    [ 22 ]

41b. |-cf (C, (h,l,R_val(rt)), R[ri -> R R_val(ri)], C(R_val(rt)))  [ (S-t), 1, 2', 3', 4', 5', 6' ]
* BRZ-TAKEN.B complete




SUBCASE C: f = G and . |- Ez =/= 0  and  . |- R_val(rt) = Ela + 1  : Two wrongs make a right
                                                              - control flow should have taken fallthru, but ends up jmping to fallthru

30c. P(l+1) = All[Df](Gf,Af)                                        [ Inspection of 6, 7 ]
31c. Exists Sf. . |- Sf : Df                                        [ Inversion of (brz-t), 7 ]
32c. |- G[ri -> <R,check,Ef'>] <= Sf(Gf)
33c. |- seq o Ela o Ef' = Sf(Af)

34c. . |- Ela = l                                                   [ Inversion of (h-append), 3 ]
35c. . |- R_val(rt) = l + 1                                         [ Exp Eq Transitivity, subcase assumption, 34c ]

36c. . |- R_val(ri) = Ez'?Ef':Et'                                   [ Inversion of (val-t), f=G, 17 ]
37c. . |- Ez'?Ef':Et' = Ef'                                         [ Conditional Exp Lemma, (Exp Eq Transitivity, subcase assumption, 10) ]
38c. . |- R_val(ri) = Ef'                                           [ Exp Eq Transitivity, 36c, 37c ]
39c. .;P |-f R R_val(ri) : <R,check,Ef'>                            [ (val-t), (rit-t), 38c ]

40c. Df = Dl, Gf = Gl, Af = Al                                      [ 20, 30c, 35c ]
41c. . |- G[ri -> <R,check,Ef'>] <= Sf(Gf)                          [ 32c ]
42c. P |-G R[ri -> R R_val(ri)] : G[ri -> <R,check,Ef'>]            [ (R-t), 2, 39c ]
2'. P |-G R[ri -> R R_val(ri)] : Sf(Gf)                             [ Repeated applications of Subtyping Lemma, 41c, 42c ]

6'. .;P;Sf(Gf);Sf(Af);Sf(Y);Sf(P(R_val(rt)+1)) |- C(R_val(rt))      [ Substitution Lemma, 31c, 26, 40c ]

43c. . |- R_val(rt) = Ef'                                           [ Exp Eq Transitivity, 35c, 10 ]
44c. |- (h,l,R_val(rt)) : seq o Ela o Ef'                           [ (h-append-t), 3, 43c ]
3'.  |- (h,l,R_val(rt)) : Sf(Af)                                    [ Exp Eq Transitivity, 44c, 33c ]


45c. . |- <R,check,Ef'> <= Sf(<R,check,Y>)                          [ 32c, 40c, 22 ]
46c. . |- Sf(Y) = Ef'                                               [ Inversion of (subtp-reflex), 45c ]
4'.  . |- Sf(Y) = R_val(rt)                                         [ Exp Eq Transitivity, 46c, 43c ]
```

```
5'. Sf(Gf)(ri) = Sf(<R,check,Y>)                                      [ 22 ]

41b. |-G (C, (h,l,R_val(rt)), R[ri -> R R_val(ri)], C(R_val(rt)))   [ (S-t), 1, 2', 3', 4', 5', 6' ]
* BRZ-TAKEN.C complete



SUBCASE D: f = cf:

30d. . |- Ei =/= l                                                    [ 4, subcase assumption ]
41d. . |- Ei = l                                                      [ 5, 8 ]
32d. contradiction, subcase doesn't apply                             [ 30d, 31d ]
* BRZ-TAKEN.D complete

* BRZ-TAKEN complete




CASES BRZ-HW-ERROR:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

R_val(rz) = 0    R_val(rz) not in Dom(C)
------------------------------------------------------------ (brz-hw-error)
(C, h, R, brz rz rd) -->_0 hw-error


(C,h,R,b) --/-->_0 S
cases do not apply
* BRZ-HW-ERROR complete.



** LEMMA PRESERVATION-BRZ-POSSIBLE-ELEVATION





=============================================================================================================
=============================================================================================================



*********************************************
* LEMMA PRESERVATION-JMP-POSSIBLE-ELEVATION *
*********************************************

If |-f (C, (h,l), R, jmp rt)
   and  G is regfile type used
   and  G(ri) = <B,goz,Et'>
   and  G(rt) = <G,All[Dt](Gt,At),Et>
   and  (C,(h,l),R,brz rz rt) -->_0 S'

then
(1) f = R   ==>   |-R  S'

(2) f =/= cf

(3) f = G and
    (a) R_val(rt)  =  Et    ==>  |-G  S'    // correct jump target
    (b) R_val(rt) =/= Et    ==>  |-cf S'    // incorrect jump target

(4) f = B and
    (a) R_val(ri)  =  Et'   ==>  |-B  S'    // correct intention
    (b) R_val(ri) =/= Et'   ==>  |-cf S'    // incorrect intention



PROOF: By case analysis of S -->_0 S'
******



CASES MOVI, SUB, INTEND, INTENDZ-SET, INTENDZ-UNSET, RECOVERNZ-OK, RECOVERNZ-HALT, BRZ-TAKEN, BRZ-UNTAKEN, BRZ-HW-ERROR:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

a1. |-f S
a2. S -->_0 S'

1. i =/=  brz                           [ inspection of rules ]
```

```
* CASES MOVI, SUB, INTEND, INTENDZ-SET, INTENDZ-UNSET, RECOVERNZ-OK, RECOVERNZ-HALT, BRZ-TAKEN, BRZ-UNTAKEN, BRZ-HW-ERROR complete




CASE JMP:
~~~~~~~~~

R_val(rt) in Dom(C)
------------------------------------------------------------------------------- (jmp)
(C, h, R, jmp rt) -->_0 (C, (h,R_val(rt)), R[ri -> R R_val(ri)], C(R_val(rt)))


1. |- C : P
2. P |-Z R : G
3. |- (h,l) : A
4x. Z = cf  ==> . |- Ei =/= l    and   Z = R/./G ==>  |- Ei = l
4y. G(ri) =/= <R,check,Ei> ==> . |- Ei = l
5. .;P;G;A;Ei;P(l+1) |- jmp rt

6. .; P; G; seq o Ela; Ei; t |- jmp rt                            [ Inspection of (jmp-t), 5 ]
7. G(ri)= <B,go,Et'>                                              [ Inversion of (jmp-t), 6 ]
8. G(rt) = <G,All[Dt](Gt,Rt),Et>
9. . |- Et = Et'
10. Exists S. D |- St : Dt
11. . |- G[ri -> <R,check,Et'>] <= St(Gt)
12. . |- seq o Ela o Et = St(At)

13. Forall r'. .;P |- R(r') : G(r')                               [ Inversion of (R-t), 2 ]
14. .;P |- R(ri) : <B,go,Et'>                                     [ 13, 7 ]
15. .;P |- R(rt) : <G,All[Dt](Gt,Rt),Et>                          [ 13, 8 ]

16.P(R_val(rt)) = All[Dl](Gl,Al)                                  [ Inversion of (C-t), 1, p1 ]
17. Dl = (Dl',Y:kint,alpha:kseq)
18. Gl = (Gl',ri-><R,check,Y>)
19. Al = alpha o R_val(rt)
20. Dl' |- Gl' ok  /\ Forall r' in Dom(Gl'). Gl'(r') =/= <R,t',E'>
22. Dl' |- P(R_val(rt)+1) ok
23. Dl; P; Gl; Al; Y; P(R_val(rt)+1) |- C(R_val(rt))


5 subcases (based on f, further subdivided for f = G and f = B) divided into groups depending on outcome


GROUP A: Control Flow OK ( f = R   or   f = G and . |- R_val(rt) = Et   or  f = B and . |- R_val(ri) = Et' )

subcase f = R :
24a1. . |- R_val(ri) = Et'                                            [ Inversion of (val-t), f=R, 14 ]
25a1. . |- R_val(rt) = Et                                             [ Inversion of (val-t), f=R, 15 ]

subcase f = G and . |- R_val(rt) = Et: green corruption does not affect jmp location
24a2. . |- R_val(ri) = Et'                                            [ Inversion of (val-t), f=G, 14 ]
25a2. . |- R_val(rt) = Et                                             [ subcase assumption ]

subcase f = B and . |- R_val(ri) = Et': blue corruption does not affect intention
24a3. . |- R_val(ri) = Et'                                            [ subcase assumption ]
25a3. . |- R_val(rt) = Et                                             [ Inversion of (val-t), f=B, 15 ]

merge:
24a. . |- R_val(ri) = Et'
26a. . |- R_val(rt) = Et                                              [ 25a1, 25a2 ]

27a. P(R_val(rt)) = All[Dt](Gt,At)                                    [ Equal Code Labels Lemma, 8, 26a ]
28a. Dl = Dt, Gl = Gt, Al = At                                        [ 27a, 16 ]
29a. .;P |-f R R_val(ri) : <R,check,Et'>                              [ (val-t), (rit-t), (Exp Eq Transitivity, 24a, 9) ]
30a. P |-f R[ri -> R R_val(ri)] : G[ri -> <R,check,Et'>]              [ (R-t), 2, 29a ]
2'. P |-f R[ri -> R R_val(ri)] : St(Gt)                               [ Repeated applications of Subtyping Lemma, 29a, 11 ]

6'. . ; P; St(Gt); St(At); St(Y); St(P(R_val(rt)+1))) |- C(R_val(rt))      [ Substitution Lemma, (13, 28a), 23 ]

31a. |- (h,l,R_val(rt)) : seq o Ela o Et                              [ (h-append), 3, 26a ]
3'. |- (h,l,R_val(rt)) = St(At)                                       [ 31a, 12 ]

32a. . |- <R,check,Et'> <= St(<R,check,Y>)                            [ 11, (18, 28a) ]
33a. . |- Et' = St(Y)                                                 [ Inversion of (subtp-reflex), 32a ]
4'. . |- St(Y) = R_val(rt)                                            [ Exp Eq Transitivity, 33a, 9, 26a ]

5'. St(Gl)(ri) = St(<R,check,Y>)                                      [ 22 ]

45a. |-f (C, (h,l,R_val(rt)), R[ri -> R R_val(ri)], C(R_val(rt)))           [ (S-t), 1, 2', 3', 4', 5', 6' ]
* JMP.A complete



GROUP B:  Control Flow Messed up - elevate to cf fault
```

```
d1b. Let Sl' = {42/x | Dl(x) = kint} union {empty/x | Dl(x) = kseq }        [ build a substitution of nonsense values ]
d2b. Let Sl = Sl', (seq o Ela)/alpha, R_val(ri)/Y
29b. . |- Sl : (Dl',alpha:kseq,Y:kint)                                      [ (subst-t),  d1b, d2b ]

5'. .;P;Sl(Gl);Sl(Al);Sl(Y);Sl(P(R_val(rt)+1)) |- C(R_val(rt))              [ Substitution Lemma, 29b, 23 ]

30b. Forall r' =/= ri. .;P |-cf R(r') : S(Gl)(r')                           [ (val-zap-cf-t), 20 ]
31b. P |- R_val(ri) : check                                                 [ (rit-t) ]
32b. .;P |-cf R R_val(ri) : <R,check,R_val(ri)>                             [ (val-t), 31b ]
33b. .;P |-cf R[ri -> R R_val(ri)](ri) : Sl(Gl)(ri)                         [ 32b, d2b, 18 ]
2': P |-cf R : Sl(Gl)                                                       [ (R-t), 30b, 33b ]

34b. |- (h,l,R_val(rt)) : seq o Ela o R_val(rt)                             [ (h-append-t), 3 ]
3'.  |- (h,l,R_val(rt)) : Sl(Al)                                            [ 34b, d2b, 19 ]

subcase: f = G and . |- R_val(rt) =/= Et : Jumped to wrong place
35b1. . |- R_val(ri) = Et'                                                  [ Inversion of (val-t), 14, f=G ]
36b1. . |- R_val(ri) =/= R_val(rt)                                          [ Exp Eq Transitivity, subcase assumption, 12, 35b1 ]

subcase: f = B and . |- R_val(ri) =/= Et': Intended target is wrong
35b2. . |- R_val(rt) = Et                                                   [ Inversion of (val-t), 15, f = B ]
36b2. . |- R_val(ri) =/= R_val(rt)                                          [ Exp Eq Transitivity, subcase assumption, 12, 25b2 ]

merge:
36b. . |- R_val(ri) =/= R_val(rt)                                           [ 36b1/36b2 ]
4x'. . |- Sl(Y) =/= R_val(rt)                                               [ 36b, d2b ]

4y'. Sl(Gl)(ri) = Sl(<R,check,Y>)                                           [ 22 ]

41b. |-cf (C, (h,l,R_val(rt)), R[ri -> R R_val(ri)], C(R_val(rt)))  [ (S-t), 1, 2', 3', 4x', 4y', 5' ]
* JMP.B complete



SUBCASE C: f = cf:

30c. . |- Ei =/= l                                                  [ 4, subcase assumption ]
31c. . |- Ei = l                                                    [ 5, 8 ]
32c. contradiction, subcase doesn't apply                           [ 30c, 31c ]
* JMP.C complete

* JMP complete






CASES JMP-HW-ERROR:
~~~~~~~~~~~~~~~~~~~~

R_val(r) not in Dom(C)
---------------------------------------------------------------- (jmp-hw-error)
(C, h, R, jmp r) -->_0 hw-error

(C,h,R,b) --/-->_0 S
case does not apply
* JMP-HW-ERROR complete.


** LEMMA PRESERVATION-JMP-POSSIBLE-ELEVATION complete



============================================================================================================
============================================================================================================

************************
* PRESERVATION PART 1: *
************************

1. If |- S  and S -->_0 S' then |- S'


PROOF: By case analysis on the structure of S.b
******

Each case uses Lemma Preservation-No-Elevation or Lemma Preservation-Jmp-Brz-Empty-Z as appropriate.




============================================================================================================
============================================================================================================

************************
```

```
* PRESERVATION PART 2: *
************************

2. If |-f S  and S -->_0 S' then Exists Z'.  |-Z' S'  and Z' >= f

PROOF: By case analysis on the structure of S.b
******

Each case uses Lemma Preservation-No-Elevation or Lemma Preservation-Brz-Possible-Elevation or Lemma
Preservation-Jmp-Possible-Elevation as appropriate.




==============================================================================================================
==============================================================================================================

************************
* PRESERVATION PART 3: *
************************

If |- S and S -->_1 S' then Exists c. |-c S'


PROOF: By case analysis of S -->_1 S'
******


CASE ZAP-REG:
~~~~~~~~~~~~~

R(r) = c n
----------------------------------------------------- (zap-reg)
(C, (h,l), R, b) -->_1 (C, (h,l), R[r -> c n'], b)


a1. |- (C,(h,l),R,b)

1. |- C : P                                                            [ Inversion of (S-t), a1 ]
2. P |- R : G
3. . |- (h,l) : A
4x. . |- Ei = l
4y. G(ri) =/= <R,check,Ei> ==> . |- Ei = l
5. .;P;G;A;Ei |- b

6. Forall r'. P;. |- R(r') : G(r')                                     [ Inversion of (R-t), 2 ]
7. P;. |- R(r) : G(r)                                                  [ 6 ]
8. Exists t',E. G(r) = <c,t',E>                                        [ 7, p1, inspection of (val-t) ]
9. P;. |-c  c n' : <c,t',E>                                            [ (val-zap-c-t) ]
10. Forall r'. P;. |-c R(r') : G(r')                                   [ Color Weakening Lemma, 6 ]
2'. P |-c R[r -> c n'] : G                                             [ (R-t), 10, 8, 9 ]

12. |-c (C, hi, ha, R[r -> c n'], b)                                   [ (S-t), 1, 2', 3, 4x, 4x, 5 ]
* ZAP-REG complete



CASE ZAP-RECOVER:
~~~~~~~~~~~~~~~~~

R(rz) =/= 0     l' in Dom(C)
----------------------------------------------------------------- (zap-recover-linC)
(C, (h,l), R, recover rz; b) -->_1 (C, (h,l,l'), R, C(l'))

R(rz) =/= 0
----------------------------------------------------------------- (zap-recover-lnotinC)
(C, (h,l), R, recover rz; b) -->_1 hw-error(h)

a1. |- (C,(h,l),R,b)

1. |- C : P                                                                [ Inversion of (S-t), a1, Inspection of (recovernz-t)'s ]
2. P |- R : G
3. . |- (h,l) : seq o Ela
4x. . |- Ei = l
4y. G(ri) =/= <R,check,Ei> ==> . |- Ei = l1
5. .;P;G;seq o Ela;Ei |- recover rz;b

subcase on structure of 5 (recovernz-t does not apply as D = . )

subcase RECOVERNZ-EQ-T:

G(rz) = <R,int,Ez>
G(ri) = <R,check,Ei>
. |- Ez = Ei - Ela
. |- Ei = Ela
.; P; G[ri -> <R,ok,Ei>]; seq o Ela; Ei; to |- b
```

```
-------------------------------------------------- (recovernz-eq-t)
.; P; G; seq o Ela; Ei; to |- recovernz rz ; b


6a. G(rz) = <R,int,Ez>                                                [ Inversion of (recovernz-eq-t), 5 ]
7a. . |- Ez = Ei - Ela

8a. . |- R_val(rz) = Ez                                               [ Inversion of (val-t), Z = ., (2, 6a) ]
9a. . |- Ez = 0                                                       [ Exp Eq Transitivity, 8a, p1 ]
10a. . |- Ei =/= Ela                                                  [ 9a, 7a ]
11a. . |- l = Ela                                                     [ Inversion of (h-append-t), 3 ]
12a. . |- Ei =/= l                                                    [ Exp Eq Transitivity, 10a, 11a ]
subcase does not apply                                                [ 12a contradicts 4x ]

subcase RECOVERNZ-NEQ-T:

G(rz) = <R,int,Ez>
G(ri) = <R,check,Ei>
. |- Ez = Ei - Ela
. |- Ei =/= Ela
-------------------------------------------------- (recovernz-neq-t)
.; P; G; seq o Ela; Ei; to |- recovernz rz; b

subcase does not apply                                                        [ p4 contradicts 4x ]

* ZAP-RECOVER-LinC and ZAP-RECOVER-LnotinC complete




CASE ZAP-RECOVER:
~~~~~~~~~~~~~~~~~

l in Dom(C)
---------------------------------------------------------------- (zap-recover-linC)
(C, (h,l), R, recover rz; b) -->_1 (C, (h,l), R, C(l))

Rz



** Progress Part 3 complete

=============================================================================================================
=============================================================================================================



COROLLARY: PRESERVATION-FAULT-ELEVATION:
****************************************

If |-Z S  and S -->_k S' and k <= 1 then Exists Z'.  |-Z' S'  and Z' >= Z


PROOF: By Case analysis on Z and k.
******

If Z = . and k = 0, then by Preservation Part 1 and . >= .
If Z = f and k = 0, then by Preservation Part 2
If k = 1, then by Preservation Part 3

* Corollary Complete
```

Fault Tolerance Definitions

```
 ------------
| Sf sim_c S |
 ------------



------------------- (sim-val)
c' n   sim_c  c' n


--------------------------- (sim-val-zap)
c n  sim_c  c n'


Forall r. Rf(r) sim_sc R(r)
--------------------------- (sim-R)
Rf sim_c R


|-  (C,h,R, b)
|-c (C,h,Rf,b)
Rf sim_c R
--------------------------- (sim-S)
(C,h,Rf,b) sim_c (C,h,R,b)




 ------------
| S -->^l S' | Block Transition - transitition between blocks
 ------------


(C,h,R,b) -->_0 (C,(h,l),R',b')
------------------------------- (trans-eval)
(C,h,R,b) -->^l (C,(h,l),R',b')




 -------------
| S -->*_k MS | Block Evaluation -- evaluate to last instruction in current block adding k faults
 -------------  (when k is left off, assume 0)


(C,h,R,b) -->_0 recover
-------------------------- (blk-eval-recover)
(C,h,R,b) -->*_0 recover


---------------------------------------- (blk-eval-jmp)
(C,h,R,jmp rt) -->*_k  (C,h,R,jmp rt)


-------------------------------------------- (blk-eval-brz)
C,h,R,brz rz rt) -->*_k  (C,h,R,brz rz rt)



(C,h,R,b) -->_k1 (C,h,R',b')    (C,h,R',b') -->*_k2 MS
------------------------------------------------------------------- (blk-eval-sequence)
(C,h,R,b) -->*_(k1+k2) MS
```

```
 ------------
| S -->^h MS | Program Execution -- evaluate a program through a sequence of blocks
 ------------  (when k is left off, assume 0)


S -->*_k MS
------------------------- (prog-exec-blk)
S -->^()_k MS


S -->^h_k S'   S' -->_0 hw-error
-------------------------------------- (prog-exec-seq-hw-error)
S -->^h_k hw-error


S -->^h S''  S'' -->^l S'''  S''' -->*_k MS
-------------------------------------------- (prog-exec-seq-trans-blk)
S -->^(h,l)_k MS
```

Fault Tolerance Lemmas

```
Lemma SIM_C-TYPING
------------------

If Sf sim_c S then

(1) Sf = (C,h,Rf,b)
(2) S =  (C,h,R ,b)
(3) |- C : P
(4) P |- R : G
(5) P |-c Rf : G
(6) |- (h,l) : A
(7) |- Ei = l
(8) .;P;G;A;Ei;P(l+1) |- b

Proof:
------

1. Sf = (C,h,Rf,b) and S = (C,h,R,b)                      [ By inspection of (sim-S), a1 ]

2. |-c (C,h,Rf,b)                                          [ By inversion of (sim-S), a1 ]
3. |- (C,h,R,b)
4. Rf sim_c R

5. |- C : P                                                [ By inversion of (S-t), 3 ]
6. P |- R : G
7. |- (h,l) : A
8. . |- Ei = l
10. .;P;G;A;Ei;P(l+1) |- b

11. Forall r. Rf(r) sim_c R                                [ Inversion of (sim-R), 4 ]
12. Forall r. Rf_col(r) = R_col(r)                         [ 11, inspection of (val-t) and (val-zap-c-t) ]

13. Forall r. .;P |- R(r) : G(r)
14. Forall r. G(r) = <R_col(c),t'_r, E_r>                  [ def of G, 13 ]
15. Forall r. . |- E_r : kint                              [ Inversion of (val-t), 14, 14 ]

16. Forall r where R_col(r) = c.                           [ (val-zap-c-t), (14, 12), 15 ]
    .;P |-c Rf(r) : G(r)

17. Forall r where R_col(r) =/= c.                         [ Inversion of (val-t), 13 ]
    . |- E_r = R_val(r) and P |- R_val(r) : t'_r
18. Forall r where R_col(r) =/= c.                         [ (val-t), 17, (14, 12) ]
    .;P |-c Rf(r) : G(r)

19. P |-c Rf(r) : G(r)                                     [ 16/18 ]

20. conclusions 1 - 8                                      [ 1, 1, 5, 6, 19, 7, 8, 10 ]

* Lemma Sim_c-Typing complete

==============================================================================================================================================
==============================================================================================================================================

Lemma NonFaulty Block Execution
--------------------------------

If |- S  then  S --->* S'
If |- S  then  S -->^h S'

Proof: By repeated Progress Part 1 and Preservation Part 1


==============================================================================================================================================
==============================================================================================================================================

Lemma Prog Exec Split
---------------------

If S ->^h S' and length h >= 1 then  Exists S1,S2,S3 h1, h2. S -->^h1 S1 and S1 -->^l S2 and S2 -->* S3
and S3 -->^h2 S'   and h = (h1,l,h2).

Proof: By induction on the structure of S -->^h MS
```

```
=====================================================================================================================================
=====================================================================================================================================

Lemma Prog Exec Join
----------------------

If S -->^h1_k1 S1  and S1 -->^l S2 and S2 -->*_k2 S3 and S3 -->^h2_k3 S'
then S -->^(h1,l,h2)

Proof: By induction on the structure of S -->^h2_k3 S'


=====================================================================================================================================
=====================================================================================================================================

Lemma Fault Introduction
-------------------------

If S -->* S'
then Exists c. S -->*_1 Sf  and  Sf sim_c S'  or  S -->*_1 recover

Proof:
By induction on the structure of S -->* S'.  Single step as in progress 3 -- also prove the Rf sim_c R.
Use Block Step Lemma after fault occurs.
```

Control Flow Recovery Lemma

```
CF Recovery Lemma
-----------------

If |-cf Sf  then  Sf -->* recover(Sf.h)


Proof:
------

By induction on the length of b.

Length(b) = 1:
Since |-cf Sf and b = jmp or brz, contradiction and subcase does not apply

Length(b) = 2:
Call CF Step Lemma. Since b' = jmp or brz, must be that S -->_0 recover. then use blk-evak-recover.

Inductive case:
Call CF Step Lemma. Either
(1) S -->_0 recover -- use blk-exec-recover
(2) S -->_0 S' and |-cf S'. Use IH and then put back together with S-->_0 S' using blk-eval-sequence and blk-eval-recover.



==========================================================================================================
==========================================================================================================


CF Step Lemma
-------------

If |-cf (C,h,R,b)
then either (1) (C,h,R,b) -->_0 (C,h,R',b')  and |-cf S'
        (2) (C,h,R,b) -->_0 recover(h)


Proof: By case analysis on the structure of b -
------


CASE MOVI: b = movi rd v; b'
~~~~~~~~~~~~~~~~~~~~~~~~~~~~

a1. |-cf (C,h,R,movi rd v; b')

1. (C,h,R,movi rd v; b') -->_0 (C, h, R[rd -> v], b')               [ (movi) ]
2. |-cf (C, h, R[rd -> v], b')                                      [ Progress-No-Elevation, a1, 1 ]
3. (C,h,R,movi rd v;b') -->_0 (C,h,R[rd -> v],b') and |-cf (C,h,R[rd -> v],b')  [ 1, 2 ]
* MOVI complete



CASE SUB: b = sub rd rs rs; b'
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

a1. |-cf (C,h,R,sub rd rs rs; b')

d1. let v' = R_col(rs1) (R_val(rs1) - R_val(rs2))
1. (C,h,R,sub rd rs rs; b') -->_0 (C, h, R[rd -> v], b)             [ (sub), d1 ]
2. |-cf (C, h, R[rd -> v], b)                                       [ Progress-No-Elevation, a1, 1]
3. (C,h,R,sub rd rs rs;b') -->_0 (C,h,R[rd -> v],b) and |-cf (C,h,R[rd -> v],b) [ 1, 2 ]
* SUB complete



CASE INTEND: b = intend ri rt; b'
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
a1. |-cf (C,(h,l),R, intend ri rt; b')

1. . |- Ei =/= l                                             [ Inversion of (S-t), a1 ]
2. G(ri) =/= <R,check,Ei> ==> . |- Ei = l
3. .;P;G;A;Ei;P(l+1) |- intend ri rt; b'

4. G(ri) = <ci,ok,Ei>                                        [ Inversion of (sequence-t), 3, Inversion of (intend-t), ]
5. . |- Ei = l                                               [ 2, 4 ]
6. contradiction                                             [ 1, 5 ]
case does not apply
* INTEND complete




CASE INTENDZ: b = intendz rz rt; b'
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

a1. |-cf (C,(h,l),R, intendz rz rt; b')

1. . |- Ei =/= l                                             [ Inversion of (S-t), a1 ]
2. G(ri) =/= <R,check,Ei> ==> . |- Ei = l
3. .;P;G;A;Ei;P(l+1) |- intendz rz rt; b'

4. G(ri) = <R,check,Eli>                                     [ Inversion of (sequence-t), 3, Inversion of (intendz-t), ]
5. . |- Ei = l                                               [ 2, 4 ]
6. contradiction                                             [ 1, 5 ]
case does not apply
* INTENDZ complete




CASE RECOVERNZ: b = recovernz rz; b'
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

a1. |- (C,(h,l),R,recovernz rz; b')

1. . |- Ei =/= l                                             [ Inversion of (S-t), a1, Inspection of (recovernz-t, -eq-t, -neq-t) ]
2. G(ri) =/= <R,check,Ei> ==> . |- Ei = l
3. |- (h,l) : seq o Ela
4. .;P;G;seq o Ela;Ei;P(l+1) |- recovernz rz; b'

subcase on the structure of 4 (recovernz-t does not apply as D = .)


SUBCASE RECOVERNZ.A:

G(rz) = <R,int,Ez>
G(ri) = <R,check,Ei>
. |- Ez = Ei - Ela
. |- Ei = Ela
.; P; G[ri -> <R,ok,Eli>]; seq o Ela; Ei; to |- b
------------------------------------------------- (recovernz-eq-t)
.; P; G; seq o Ela; Ei; to |- recovernz rz ; b

5a. . |- l = Ela                                             [ Inversion of (h-append-t), 3 ]
6a. . |- Ei = l                                              [ Exp Eq Transitivity, pa4, 5a ]
7a. contradiction                                            [ 6a, 1 ]
subcase does not apply
* RECOVERNZ.A complete


SUBCASE RECOVERNZ.B:

G(rz) = <R,int,Ez>
G(ri) = <R,check,Ei>
. |- Ez = Ei - Ela
. |- Ei =/= Ela
------------------------------------------------- (recovernz-neq-t)
.; P; G; seq o Ela; Ei; to |- recovernz rz; b

5b. P |-cf R : G                                             [ Inversion of (S-t), a1 ]
6b. .;P |-cf R(ri) : <R,check,Ei>                            [ Inversion of (R-t), 5b, pb2 ]
7b. .;P |-cf R(rz) : <R,int,Ez>                              [ Inversion of (R-t), 5b, pb1 ]
8b. . |- R_val(ri) = Ei                                      [ Canonical Forms, 6b, (Inversion of (S-t), a1) ]
9b. . |- R_val(rz) = Ez                                      [ Canonical Forms, 7b, (Inversion of (S-t), a1) ]
10b. . |- R_val(rz) = Ei - Ela                               [ Exp Eq Transitivity, 9b, pb3 ]
11b. R_val(rz) =/= 0                                         [ 10b, pb4 ]
12b. (C, (h,l), R, recovernz rz; b) -->_0 recover(h,l)       [ (recover-halt), 11b ]
```

```
* RECOVERNZ.B complete




CASE BRZ: b = brz rz rt
~~~~~~~~~~~~~~~~~~~~~~~

a1. |-cf (C,(h,l),R, brz rz rt)

1. . |- Ei =/= l                                              [ Inversion of (S-t), a1 ]
2. G(ri) =/= <R,check,Ei> ==> . |- Ei = l
3. .;P;G;A;Ei;P(l+1) |- brz rz rt

4. G(ri) = <B,goz,Ez'?Ef':Et'>                                [ Inversion of (sequence-t), 3, Inversion of (brz-t), ]
5. . |- Ei = l                                                [ 2, 4 ]
6. contradiction                                              [ 1, 5 ]
case does not apply
* BRZ complete




CASE JMP: b = jmp rt
~~~~~~~~~~~~~~~~~~~~

a1. |-cf (C,(h,l),R, jmp rt)

1. . |- Ei =/= l                                              [ Inversion of (S-t), a1 ]
2. G(ri) =/= <R,check,Ei> ==> . |- Ei = l
3. .;P;G;A;Ei;P(l+1) |- jmp rt

4. G(ri)= <B,go,Et'>                                          [ Inversion of (sequence-t), 3, Inversion of (jmp-t), ]
5. . |- Ei = l                                                [ 2, 4 ]
6. contradiction                                              [ 1, 5 ]
case does not apply
* JMP complete
```

Block Lemmas

```
Block Execution Lemma
----------------------

If  Sf sim_c S  and  S -->* S'

then either     (1) Sf -->* Sf'  and  Sf' sim_c S'
        (2) Sf -->* recover(S.h)


PROOF: By induction on the structure of  S -->* S'
------


CASE-BLK-EVAL-RECOVER:
~~~~~~~~~~~~~~~~~~~~~~

(C,h,R,b) -->_0 recover(h)
-------------------------- (blk-eval-recover)
(C,h,R,b) -->*_0 recover(h)

S -/->* recover(h)                      [ Non Faulty Block Execution Lemma, a1 ]
subcase does not apply
* BLK-EVAL-RECOVER complete



CASE BLK-EVAL-BRZ:
~~~~~~~~~~~~~~~~~~

------------------------------------------- (blk-eval-brz)
(C,h,R,brz rz rt) -->*_0  (C,h,R,brz rz rt)

1. Sf = (C,h,Rf,jmp rt)                     [ Inversion of (sim-S), a1 ]
2. Sf -->*_0 Sf                             [ (blk-eval-brz), 1 ]
3. Sf -->* Sf and Sf sim_c S                [ 2, a1 ]
* BLK-EVAL-BRZ complete




CASE BLK-EVAL-JMP:
~~~~~~~~~~~~~~~~~~


--------------------------------------- (blk-eval-jmp)
(C,h,R,jmp rt) -->*_0  (C,h,R,jmp rt)


1. Sf = (C,h,Rf,jmp rt)                     [ Inversion of (sim-S), a1 ]
2. Sf -->*_0 Sf                             [ (blk-eval-jmp), 1 ]
3. Sf -->* Sf and Sf sim_c S                [ 2, a1 ]
* BLK-EVAL-JMP complete




CASE BLK-EVAL-SEQUENCE:
~~~~~~~~~~~~~~~~~~~~~~~


(C,h,R,b) -->_k1 (C,h,R',b')    (C,h,R',b') -->*_k2 MS''
----------------------------------------------------------------- (blk-eval-sequence)
(C,h,R,b) -->*_0 MS''
```

```
1. Sf -->_0 Sf'  and  Sf' sim_c (C,h,R',b')     [ Block Step Lemma, a1, p1 ]
   OR Sf -->_0 recover
subcase on 1.

SUBCASE BLK-EVAL-SEQUENCE.A: Faulty execution has been simulating original...
a1. Sf -->_0 Sf'
a2. Sf' sim_c (C,h,R',i';b')

3a. Sf' -->* Sf'' and Sf'' sim_c S''            [ I.H., a2, p2 ]
    OR Sf' -->* recover(h)
subcase on 3a.

subsubcase BLK-EVAL-SEQUENCE.A1: ...and continues to do so
aa1. Sf' -->* Sf''
aa2. Sf'' sim_c S''
4a1. Sf -->* Sf''                               [ (blk-eval-sequence), a1, aa1 ]
5a1. Sf -->* Sf'' and Sf'' sim_c S''            [ 4a1, aa2 ]
* subsubcase BLK-EVAL-SEQUENCE.A1 complete

subsubcase BLK-EVAL-SEQUENCE.A2: ...but recovers in the last step
aa2. Sf' -->* recover(h)
5b2. Sf -->* recover(h)                         [ (blk-eval-sequence), a1, ab1 ]
* subsubcase BLK-EVAL-SEQUENCE.A2 complete


SUBCASE BLK-EVAL-SEQUENCE.B: Faulty execution has already recovered
a1. Sf -->_0 recover(h)
2a. Sf -->* recover(h)                          [ (blk-eval-recover), a1 ]
* SUBCASE BLK-EVAL-SEQUENCE.B complete


** Block Execution Lemma Complete



=================================================================================================================================
=================================================================================================================================


Block Step Lemma
----------------

If   Sf sim_c (C,h,R,b)  and  (C,h,R,b) -->_0 (C,h,R',b')

then either     (1) Sf -->_0 Sf'  and  Sf' sim_c (C,h,R',b')
        (2) Sf -->_0 recover(h)


PROOF: By case analysis of (C,h,R,b) -->_0 (C,h,R',b')


a1. Sf sim_c (C,h,R,b)
a2. (C,h,R,b) -->_0 (C,h,R',b')

1. Sf = (C,h,Rf,b)                          [ Lemma Sim_c-Typing, a1 ]
2. |- C : P
3. P |- R : G
4. P |-c Rf : G
5. |- h : A
6. |- Ei = end(h)
7. .;P;G;A;Ei;P(l+1) |- b

8. Forall r. Rf(r) sim_c R(r)               [ Inversion of (sim-S), a1, Inversion of (sim-R) ]

9. |- (C,h,R',b')                           [ Preservation Part 1, (Inversion of (sim-S), a1), a2 ]
10. |-c (C,h,Rf,b)                          [ Inversion of (sim-S), a1 ]



CASE MOVI:
~~~~~~~~~~

---------------------------------------------------- (movi)
(C, h, R, movi rd v; b) -->_0 (C, h, R[rd -> v], b)

15. (C, h, Rf, movi rd v; b) -->_0 (C, h, Rf[rd -> v], b)       [ (movi) ]
16. |-c (C, h, Rf[rd -> v], b)                                  [ Preservation-No-Elevation, 10, 15 ]

17. v sim_c v                                                   [ (sim-val) ]
18. Rf[rd -> v]  sim_c R[rd -> v]                               [ (sim-R), 8, 17 ]
19. (C, h, Rf[rd -> v], b) sim_c (C, h, R[rd -> v], b)          [ (sim-S), 9, 16, 18 ]
```

```
20. Sf -->_0 Sf'  and  Sf' sim_c S'                            [ 15, 19 ]
* MOVI complete



CASE SUB:
~~~~~~~~~

v' = R_col(rs1) (R_val(rs1) - R_val(rs2))
--------------------------------------------------------------- (sub)
(C, h, R, sub rd, rs1, rs2; b) -->_0 (C, h, R[ rd -> v' ],  b)

15. Rf(rs1) sim_c R(rs1)                                       [ 8 ]
16. Rf_col(rs1) = R_col(rs1)                                   [ Inspection of (val-t) ]

d1. let vf' = Rf_col(rs1) (Rf_val(rs1) - Rf_val(rs2))

subcase Rf_col(rs1) = c
17a. vf' sim_c vf                                              [ (sim-val-zap), 16, subcase assumption ]
subcase Rf_col(rs1) =/= c
17b. .;P;G |- sub rd, rs1, rs2 : G'                            [ Inversion of (sequence-t), 7 ]
18b. R_col(rs1) = R_col(rs2)                                   [ Inversion of (sub-t), 3 ]
19b. Rf(rs1) = R(rs1)  and  Rf(rs2) = R(rs2)                   [ Inversion of (sim-val), 18, subcase assumption ]
20b. vf' sim_c v'                                              [ (sim-val), 19b, d1, p1 ]
merge:
21. vf' sim_c v'                                               [ 17a/20b]

22. (C, h, Rf, sub rd, rs1, rs2; b) -->_0 (C, h, Rf[ rd -> vf' ],  b)   [ (sub) ]
23. |-c (C, h, Rf[rd -> v], b)                                 [ Preservation-No-Elevation, 10, 22 ]

24. Rf[rd -> vf']  sim_c R[rd -> v']                           [ (sim-R), 8, 21 ]
25. (C, h, Rf[rd -> v], b) sim_c (C, h, R[rd -> v], b)         [ (sim-S), 9, 23, 24 ]

26. Sf -->_0 Sf'  and  Sf' sim_c S'                            [ 22, 25 ]
* SUB complete



CASE INTEND:
~~~~~~~~~~~~

----------------------------------------------------------- (intend)
(C, h, R, intend rt; b) -->_0 (C, h, R[ri -> R(rt)],  b)

15. (C, h, Rf, intend rt; b) -->_0 (C, h, Rf[ri -> Rf(rt)],  b) [ (intend) ]
16. |-c (C, h, Rf[ri -> Rf(rt)], b)                            [ Preservation-No-Elevation, 10, 15 ]

17. Rf(rt) sim_c R(rt)                                         [ 8 ]
18. Rf[ri -> Rf(rt)]  sim_c R[ri -> R(rt)]                     [ (sim-R), 8, 17 ]
19. (C, h, Rf[ri -> Rf(rt)],  b) sim_c (C, h, R[ri -> R(rt)], b)[ (sim-S), 9, 16, 18 ]

20. Sf -->_0 Sf'  and  Sf' sim_c S'                            [ 15, 19 ]
* INTEND complete




CASE INTENDZ-SET:
~~~~~~~~~~~~~~~~~

R_val(rz) = 0
-------------------------------------------------------------- (intendz-set)
(C, h, R, intendz rz rt; b) -->_0 (C, h, R[ri -> R(rt)],  b)

14. R_col(rz) = Rf_col(rz) = B                                 [ Inversion on (sequence-t), (intendz), Canonical Forms, 4, 5 ]

subcase c =/= B:
14a. R_val(rz) = Rf_val(rz)                                    [ Inversion on (val-t), c =/= B, 14a, 8 ]
15a. (C, h, Rf, intendz rz rt; b) -->_0 (C, h, Rf[ri -> Rf(rt)],  b)    [ (intendz-set), (14a, p1) ]
16a. |-c (C, h, Rf[ri -> Rf(rt)],  b)                          [ Preservation-No-Elevation, 10, 15a ]
17a. Rf[ri -> Rf(rt)] sim_c R[ri -> R(rt)]                     [ (sim-R), 8, 14a, 14 ]
18a. (C, h, Rf[ri -> Rf(rt)],  b) sim_c (C, h, R[ri -> R(rt)],  b)  [ (sim-S), 17a, 9, 16a ]
19a. Sf -->_0 Sf'  and  Sf' sim_c S'                           [ 15a, 18a ]
* INTENDZ-SET.A complete

subcase c = B  and Rf_val(rz) = 0
14b. (C, h, Rf, intendz rz rt; b) -->_0 (C, h, Rf[ri -> Rf(rt)],  b)    [ (intendz-set), subcase assumption ]
15b. |-c (C, h, Rf[ri -> Rf(rt)],  b)                          [ Preservation-No-Elevation, 10, 14b ]
16b. Rf(rt) sim_B R(rt)                                        [ (sim-val), 14, c = B ]
17b. Rf[ri -> Rf(rt)] sim_c R[ri -> R(rt)]                     [ (sim-R), 8, 16b ]
```

```
18b. (C, h, Rf[ri -> Rf(rt)],  b) sim_B (C, h, R[ri -> R(rt)],  b)  [ (sim-S), 16b, 15b, 9 ]
19b. Sf -->_0 Sf'  and  Sf' sim_c S'                               [ 14b, 18b ]
* INTENDZ-SET.B complete

subcase c = B and  Rf_val =/= 0
14c. (C, h, Rf, intendz rz rt; b) -->_0 (C, h, Rf, b)          [ (intendz-unset), subcase assumption ]
15c. |-c (C, h, Rf, b)                                         [ Preservation-No-Elevation, 10, 14c ]
16c. Rf_col(ri) = B                                            [ Inversion on (sequence-t), (intendz), Canonical Forms, 5 ]
17c. Rf(ri) sim_B R(rt)                                        [ (sim-val-zap-c), c = B, 16c, 14 ]
18c. Rf sim_B R[ri -> R(rt)]                                   [ (sim-R), 8, 17c ]
19c. (C, h, Rf, b) sim_B (C, h, R[ri -> R(rt)],  b)            [ (sim-S), 18c, 15c, 9 ]
20c. Sf -->_0 Sf'  and  Sf' sim_c S'                           [ 14c, 19c ]
* INTENDZ-SET.C complete



CASE INTENDZ-UNSET:
~~~~~~~~~~~~~~~~~~~

R_val(rz) =/= 0
------------------------------------------------- (intendz-unset)
(C, h, R, intendz rz rt; b) -->_0 (C, h, R, b)

14. R_col(rz) = Rf_col(rz) = B                                 [ Inversion on (sequence-t), (intendz), Canonical Forms, 4, 5 ]

subcase c =/= B:
14a. R_val(rz) = Rf_val(rz)                                    [ Inversion on (sim-val), c =/= B, 14a, 8 ]
15a. (C, h, Rf, intendz rz rt; b) -->_0 (C, h, Rf,  b)         [ (intendz-unset), (14a, p1) ]
16a. |-c (C, h, Rf,  b)                                        [ Preservation-No-Elevation, 10, 15a ]
17a. Rf sim_c R                                                [ Inversion on (sim-S), a1 ]
18a. (C, h, Rf,  b) sim_c (C, h, R,  b)                        [ (sim-S), 17a, 9, 16a ]
19a. Sf -->_0 Sf'  and  Sf' sim_c S'                           [ 15a, 18a ]
* INTENDZ-UNSET.A complete

subcase c = B  and Rf_val(rz) =/= 0
14b. (C, h, Rf, intendz rz rt; b) -->_0 (C, h, Rf, b)          [ (intendz-unset), subcase assumption ]
15b. |-c (C, h, Rf,  b)                                        [ Preservation-No-Elevation, 10, 14b ]
16b. Rf sim_c R                                                [ Inversion on (sim-S), a1 ]
17b. (C, h, Rf,  b) sim_B (C, h, R, b)                         [ (sim-S), 16b, 15b, 9 ]
18b. Sf -->_0 Sf'  and  Sf' sim_c S'                           [ 14b, 17b ]
* INTENDZ-SET.B complete

subcase c = B and  Rf_val = 0
14c. (C, h, Rf, intendz rz rt; b) -->_0 (C, h, Rf[ri -> R(rt)], b)  [ (intendz-set), subcase assumption ]
15c. |-c (C, h, Rf[ri -> R(rt)], b)                            [ Preservation-No-Elevation, 10, 14c ]
16c. Rf_col(rt) = B                                            [ Inversion on (sequence-t), (intendz), Canonical Forms, 5 ]
17c. Rf(rt) sim_B R(rt)                                        [ (sim-val-zap-c), c = B, 16c, 14 ]
18c. Rf[ri -> R(rt)] sim_B R                                   [ (sim-R), 8, 17c ]
19c. (C, h, Rf[ri -> R(rt)], b) sim_B (C, h, R,  b)            [ (sim-S), 18c, 15c, 9 ]
20c. Sf -->_0 Sf'  and  Sf' sim_c S'                           [ 14c, 19c ]
* INTENDZ-SET.C complete




CASE RECOVERNZ-OK:
~~~~~~~~~~~~~~~~~

R_val(rz) = 0
---------------------------------------------- (recovernz-ok)
(C, h, R, recovernz rz; b) -->_0 (C, h, R, b)

14. R_col(rz) = Rf_col(rz) = R                                 [ Inversion on (sequence-t), (intendz), Canonical Forms, 4, 5 ]

subcase Rf_val(rz) = 0:
15a. (C, h, Rf, recovernz rz; b) -->_0 (C, h, Rf, b)           [ (recovernz-ok), subcase assumption ]
16a. |-c (C, h, Rf, b)                                         [ Preservation-No-Elevation, 10, 15c ]
17a. Rf sim_c R                                                [ Inversion on (sim-S), a1 ]
18a. (C, h, Rf, b) sim_R (C, h, R, b)                          [ (sim-S), 17a, 16a, 9 ]
19a. Sf -->_0 Sf'  and  Sf' sim_c S'                           [ 15a, 18a ]
* RECOVERNZ-OK.A complete

subcase Rf_val(rz) =/= 0:
15a. (C, h, Rf, recovernz rz; b) -->_0 halt(h)                 [ (recovernz-halt), subcase assumption ]
* RECOVERNZ-OK.B complete



CASE RECOVERNZ-HALT:
~~~~~~~~~~~~~~~~~~~~
```

```
R_val(rz) =/= 0
-------------------------------------------- (recovernz-halt)
(C, h, R, recovernz rz; b) -->_0 recover(h)

S -/->_0 S'
case does not apply
* RECOVERNZ-HALT complete



CASES BRZ-UNTAKEN, BRZ-TAKEN, JMP:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

S'.h =/= S.h
cases do not apply
* BRZ-UNTAKEN, BRZ-TAKEN, JMP complete




CASES BRZ-HW-ERROR, JMP-HW-ERROR:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

S -/->_0 S'
cases do not apply
* BRZ-HW-ERROR, JMP-HW-ERROR complete


** BLOCK STEP LEMMA COMPLETE
```

Transition Lemmas

```
Block Transition Lemma
----------------------

If  Sf sim_c S  and  S -->^l S'

then either (1) Sf -->^l  Sf'  and  Sf' sim_c S'
        (2) Sf -->_0 hw_error(S.h)
        (3) Sf -->^l' Sf'  and  Sf' -->* recover(S.h,l')


Proof:
------

By analysis of the structure of S -->^l' S'.
By inspection of (trans-eval), we're looking at all -->_0 rules where the history is extended.




CASE BRZ-UNTAKEN:
~~~~~~~~~~~~~~~~~

R_val(rz) =/= 0     l+1 in Dom(C)
---------------------------------------------------------------------------- (brz-untaken)
(C, (h,l), R, brz rz rt) -->_0 (C, (h,l,l+1), R[ri -> R R_val(ri)], C(l+1))




a1. Sf sim_sc S
a2. S -->^l' S'

1. Sf = (C,(h,l),Rf,brz rz rt)                                        [ Lemma Sim_c-Typing, a1, Inspection of (brz-t) ]
2. S =  (C,(h,l),R ,brz rz rt)
3. |- C : P
4. P |- R : G
5. P |-c Rf : G
6. |- (h,l) : seq o Ela
7. |- Ei = l
8. .;P;G;seq o ELa;Ei;P(l+1) |- brz rz rt

9. Forall r. Rf(r) sim_c R(r)                                         [ Inversion of (sim-S), a1, Inversion of (sim-R) ]
10. |-c Sf                                                            [ Inversion of (sim-S), a1 ]

11. G(rz) = <G,int,Ez>                                                [ Inversion of (brz-t), 10 ]
12. .;P |- R(rz) : <G,int,Ez>                                         [ Inversion of (R-t), 4, 11 ]
13. . |- R_val(rz) = Ez and R_col(rz) = G                             [ Canonical Forms, 12, 3 ]
14. .;P |-c Rf(rz) : <G,int,Ez>                                       [ Inversion of (R-t), 5, 11 ]
15. . |- Ez =/= 0                                                     [ Exp Eq Transitivity, p1, 13 ]

16. G(rt) = <G,All[Dt](Gt,At),Et>                                     [ Inversion of (brz-t), 10 ]
17. .;P |- R(rt) : <G,All[Dt](Gt,At),Et>                              [ Inversion of (R-t), 4, 11 ]
18. . |- R_val(rt) = Et and R_col(rt) = G                             [ Canonical Forms, 17, 3 ]
19. .;P |-c Rf(rt) : <G,All[Dt](Gt,At),Et>                            [ Inversion of (R-t), 5, 11 ]

20. G(ri) = <B,goz,Ez'?Ef':Et'>                                       [ Inversion of (brz-t), 10 ]
21. .;P |- R(ri) : <B,goz,Ez'?Ef':Et'>                                [ Inversion of (R-t), 4, 11 ]
22. . |- R_val(ri) = Ez'?Ef':Et'  and  R_col(ri) = B                  [ Canonical Forms, 21, 3 ]
23. .;P |-c Rf(ri) : <B,goz,Ez'?Ef':Et'>                              [ Inversion of (R-t), 5, 11 ]


24. . |- l  = Ela                                                     [ Inversion of (h-append-t), 6]
25. . |- Ez = Ez'                                                     [ Inversion of (brz-t), 10 ]

25. |- S'                                                             [ Preservation, (Inversion of sim-S, a1),
                                                                          (Inversion of trans-eval-brz, a2) ]

subcase on c




SUBCASE BRZ-UNTAKEN.R: c = R

25r. Rf(ri) = R(ri)                                                   [ Inspection of (sim-val), 9, 22, c=R ]
```

```
26r. Rf[ri -> R Rf_val(ri)] sim_c R[ri -> R R_val(ri)]                    [ (sim-R), 9, (sim-val), 25r ]

27r. Rf(rz) = R(rz)                                                       [ Inspection of (sim-val), 9, 14, c=R ]
28r. Rf_val(rz) =/= 0                                                     [ 27r, p1 ]

29r. Sf -->_0 (C, (h,l,l+1), Rf[ri -> R Rf_val(ri)], C(l+1))              [ (brz-untaken), 28r, p2 ]
30r. |-R (C, (h,l,l+1), Rf[ri -> R Rf_val(ri)], C(l+1))                   [ Lemma Preservation-Brz-Possible-Elevation, 10, f=R ]
31r. (C, (h,l,l+1), Rf[ri -> R Rf_val(ri)], C(l+1)) sim_R S'              [ (sim-S), 26r, 30r, 25 ]

32r. Sf -->^(l+1) Sf'  and  Sf' sim_R S'                                  [ 29r, 31r ]
* BRZ-UNTAKEN.A complete




SUBCASE BRZ-UNTAKEN.B: c = B

25b. Rf(rz) = R(rz)                                                       [ Inspection of (sim-val), 9, 14, c=B ]
26b. Rf_val(rz) =/= 0                                                     [ 26b, p1 ]

27b. Sf -->_0 (C, (h,l,l+1), R[ri -> R R_val(ri)], C(l+1))                [ (brz-untaken), 26b, p2 ]

subsubcase on Rf_val(ri) ?= R_val(ri)

subsubcase B1: . |- Rf_val(ri) = R_val(ri)   -  blue fault does not affect intention value and simulation continues
28b1. Rf[ri -> R Rf_val(ri)] sim_B R[ri -> R R_val(ri)]                   [ (sim-R), 9, (sim-val), subsubcase assumption ]
29b1. . |- Ez'?Ef':Et' = Ef'                                              [ Exp Conditional Lemma, (Exp Eq Transitivity, 15, 25) ]
30b1. . |- Rf_val(ri) = Ef'                                               [ Exp Eq Transitivity, 22, 29b1, subcase assumption ]
31b1. |-B (C, (h,l,l+1), Rf[ri -> R Rf_val(ri)], C(l+1))                  [ Lemma Preservation-Brz-Possible-Elevation, 10, f=B, 26b, 30b1 ]
32b1. (C, (h,l,l+1), Rf[ri -> R Rf_val(ri)], C(l+1)) sim_B S'             [ (sim-S), 28b1, 31b1, 25 ]
32a. Sf -->^(l+1) Sf'  and  Sf' sim_B S'                                  [ 27b, 32b1 ]
* BRZ-UNTAKEN.B1 complete


subsubcase B2: . |- Rf_val(ri) =/= R_val(ri)   -  blue fault does affect intention value and faulty version recovers
28b2. . |- Ez'?Ef':Et' = Ef'                                              [ Exp Conditional Lemma, (Exp Eq Transitivity, 15, 25) ]
29b2. . |- Rf_val(ri) =/= Ef'                                             [ Exp Eq Transitivity, 22, 28b2, subcase assumption ]
30b2. |-cf (C, (h,l,l+1), Rf[ri -> R Rf_val(ri)], C(l+1))                 [ Lemma Preservation-Brz-Possible-Elevation, 10, f=B, 27b, 29b2 ]
31b2. Sf' -->* recover(h,l,l+1)                                           [ CF Lemma, 30b2 ]
32b2. Sf -->^(l+1) Sf'  and  Sf' -->* recover(h,l,l+1)                    [ 27b, 31b2 ]
* BRZ-UNTAKEN.B2 complete





SUBCASE BRZ-UNTAKEN.G: c = G

25g. Rf(ri) = R(ri)
26g. Rf[ri -> R Rf_val(ri)] sim_G R[ri -> R R_val(ri)]                    [ (sim-R), 9, (sim-val), 25g ]

subsubcase on  Rf_val(rz) =?= 0 and then Rf_val(rt) =?= Ela + 1 and then Rf_val(rt) in Dom(C)


subsubcase G1: Rf_val(rz) =/= 0: rz faults to another non-zero value, fallthru proceeds

27g1. Sf -->_0 (C, (h,l,l+1), Rf[ri -> R Rf_val(ri)], C(l+1))             [ (brz-untaken), subcase assumption, p2 ]
28g1. |-G (C, (h,l,l+1), Rf[ri -> R Rf_val(ri)], C(l+1))                  [ Lemma Preservation-Brz-Possible-Elevation, 10, f=G, 27g1,
                                                                              subcase assumption, 15 ]
29g1. (C, (h,l,l+1), Rf[ri -> R Rf_val(ri)], C(l+1)) sim_G S'             [ (sim-S), 29g1, 25, 26g ]
30g1. Sf -->^(l+1) Sf' and Sf' sim_c S'                                   [ 27g1, 29g1 ]
* BRZ-UNTAKEN.G1 complete


subsubcase G2: Rf_val(rz) = 0  and  . |- Rf_val(rt) = Ela + 1 : faulty computation incorrectly branches, but target is same as
                                                                              fallthrough, so everything a-ok

27g2. . |- R_val(rt) = l + 1                                              [ subcase assumption, 24 ]
28g2. Sf -->_0 (C,(h,l,R_val(rt)), Rf[ri -> R Rf_val(ri)], C(R_val(rt)))[ (brz-taken), subcase assumption, p2 ]
29g2. |-G (C,(h,l,R_val(rt)),Rf[ri -> R Rf_val(ri)], C(R_val(rt)))        [ Lemma Preservation-Brz-Possible-Elevation, 10, f=G,
                                                                              28g2, subcase assumptions, 15 ]
30g2. (C,(h,l,R_val(rt)),Rf[ri -> R Rf_val(ri)],C(R_val(rt))) sim_G S'  [ (sim-S), 29g2, 25, 26g, 27g2 ]
31g2. Sf -->^(l+1) Sf' and Sf' sim_c S'                                   [ 28g2, 31g2 ]
* BRZ-UNTAKEN.G2 complete


subsubcase G3: Rf_val(rz) = 0  and  . |- Rf_val(rt) =/= Ela + 1  and  . |- Rf_val(rt) in Dom(C) : faulty computations incorrectly
                                                                              branches somewhere else in C

27g3. Sf -->_0 (C,(h,l,R_val(rt)), Rf[ri -> R Rf_val(ri)], C(R_val(rt)))[ (brz-taken), subcase assumptions ]
28g3. |-cf (C,(h,l,R_val(rt)),Rf[ri -> R Rf_val(ri)], C(R_val(rt)))       [ Lemma Preservation-Brz-Possible-Elevation, 10, f=G, 27g3,
                                                                              subcase assumptions, 15 ]
29g3. Sf' -->* recover(h,l,R_val(rt))                                     [ CF Lemma, 28g3 ]
30g3. Sf -->^(l+1) Sf'  and  Sf' -->* recover(h,l,R_val(rt))              [ 27g3, 29g3 ]
* BRZ-UNTAKEN.G3 complete
```

```
subsubcase G4: Rf_val(rz) = 0  and  . |- Rf_val(rt) =/= Ela + 1  and  . |- Rf_val(rt) not in Dom(C) : faulty computations incorrectly
                                                                            branches and craps out

27g4. Sf -->_0 hw-error(h,l)                                        [ (brz-hw-error), subcase assumptions ]
* BRZ-UNTAKEN.G4 complete


* BRZ-UNTAKEN complete






CASE BRZ-TAKEN:
~~~~~~~~~~~~~~~

R_val(rz) = 0   R_val(rt) in Dom(C)
------------------------------------------------------------------------------------------ (brz-taken)
(C, (h,l), R, brz rz rt) -->_0 (C, (h,l,R_val(rt)), R[ri -> R R_val(ri)], C(R_val(rt)))


a1. Sf sim_sc S
a2. S -->^l' S'

1. Sf = (C,(h,l),Rf,brz rz rt)                                              [ Lemma Sim_c-Typing, a1, Inspection of (brz-t) ]
2. S =  (C,(h,l),R ,brz rz rt)
3. |- C : P
4. P |- R : G
5. P |-c Rf : G
6. |- (h,l) : seq o Ela
7. |- Ei = l
8. .;P;G;seq o ELa;Ei;P(l+1) |- brz rz rt

9. Forall r. Rf(r) sim_c R(r)                                               [ Inversion of (sim-S), a1, Inversion of (sim-R) ]
10. |-c Sf                                                                  [ Inversion of (sim-S), a1 ]

11. G(rz) = <G,int,Ez>                                                      [ Inversion of (brz-t), 10 ]
12. .;P |- R(rz) : <G,int,Ez>                                               [ Inversion of (R-t), 4, 11 ]
13. . |- R_val(rz) = Ez and R_col(rz) = G                                   [ Canonical Forms, 12, 3 ]
14. .;P |-c Rf(rz) : <G,int,Ez>                                             [ Inversion of (R-t), 5, 11 ]
15. . |- Ez = 0                                                             [ Exp Eq Transitivity, p1, 13 ]

16. G(rt) = <G,All[Dt](Gt,At),Et>                                           [ Inversion of (brz-t), 10 ]
17. .;P |- R(rt) : <G,All[Dt](Gt,At),Et>                                    [ Inversion of (R-t), 4, 11 ]
18. . |- R_val(rt) = Et and R_col(rt) = G                                   [ Canonical Forms, 17, 3 ]
19. .;P |-c Rf(rt) : <G,All[Dt](Gt,At),Et>                                  [ Inversion of (R-t), 5, 11 ]

20. G(ri) = <B,goz,Ez'?Ef':Et'>                                             [ Inversion of (brz-t), 10 ]
21. .;P |- R(ri) : <B,goz,Ez'?Ef':Et'>                                      [ Inversion of (R-t), 4, 11 ]
22. . |- R_val(ri) = Ez'?Ef':Et'  and  R_col(ri) = B                        [ Canonical Forms, 21, 3 ]
23. .;P |-c Rf(ri) : <B,goz,Ez'?Ef':Et'>                                    [ Inversion of (R-t), 5, 11 ]

24. . |- l  = Ela                                                           [ Inversion of (h-append-t), 6
25. . |- Ez = Ez'                                                           [ Inversion of (brz-t), 10 ]

25. |- S'                                                                   [ Preservation, (Inversion of sim-S, a1),
                                                                                (Inversion of trans-eval-brz, a2) ]



SUBCASE BRZ-TAKEN.R: c = R

25r. Rf(ri) = R(ri)                                                         [ Inspection of (sim-val), 9, 22, c=R ]
26r. Rf[ri -> R Rf_val(ri)] sim_c R[ri -> R R_val(ri)]                      [ (sim-R), 9, (sim-val), 25r ]

27r. Rf(rz) = R(rz)                                                         [ Inspection of (sim-val), 9, 14, c=R ]
28r. Rf_val(rz) = 0                                                         [ 27r, p1 ]

29r. Rf(rt) = R(rt)                                                         [ Inspection of (sim-val), 9, 18, c = R ]
30r. Rf_val(rt) in Dom(C)                                                   [ 29r ]

31r. Sf -->_0 (C,(h,l,Rf_val(rt)),Rf[ri -> R Rf_val(ri)],C(Rf_val(rt)))     [ (brz-taken), 28r, 30r ]
32r. |-R (C,(h,l,Rf_val(rt)),Rf[ri -> R Rf_val(ri)],C(Rf_val(rt)))          [ Lemma Preservation-Brz-Possible-Elevation, 10, 31r, f=R ]
33r. (C,(h,l,Rf_val(rt)),Rf[ri -> R Rf_val(ri)],C(Rf_val(rt))) sim_R S'     [ (sim-S), 26r, 32r, 25, 29r ]

32r. Sf -->^(l+1) Sf'  and  Sf' sim_R S'                                    [ 31r, 33r ]
* BRZ-TAKEN.A complete




SUBCASE BRZ-UNTAKEN.B: c = B

25b. Rf(rz) = R(rz)                                                         [ Inspection of (sim-val), 9, 14, c=B ]
```

```
26b. Rf_val(rz) =/= 0                                                              [ 26b, p1 ]

27b. Rf(rt) = R(rt)                                                                [ Inspection of (sim-val), 9, 18, c=B ]
28b. Rf_val(rt) in Dom(C)                                                          [ 27b, p1 ]

29b. Sf -->_0 (C,(h,l,Rf_val(rt)),Rf[ri -> R Rf_val(ri)],C(Rf_val(rt)))            [ (brz-taken), 26b, p2 ]

subsubcase on Rf_val(ri) ?= R_val(ri)

subsubcase B1: . |- Rf_val(ri) = R_val(ri)   -  blue fault does not affect intention value and simulation continues

28b1. Rf[ri -> R Rf_val(ri)] sim_B R[ri -> R R_val(ri)]                            [ (sim-R), 9, (sim-val), subsubcase assumption ]
29b1. . |- Ez'?Ef':Et' = Et'                                                       [ Exp Conditional Lemma, (Exp Eq Transitivity, 15, 25) ]
30b1. . |- Rf_val(ri) = Et'                                                        [ Exp Eq Transitivity, 22, 29b1, subcase assumption ]
31b1. |-B Sf'                                                                      [ Lemma Preservation-Brz-Possible-Elevation, 10, f=B,
                                                                                      29b, 30b1 ]
32b1. (C,(h,l,Rf_val(rt)),Rf[ri -> R Rf_val(ri)],C(Rf_val(rt)))sim_B S'            [ (sim-S), 28b1, 31b1, 25, 27b ]
32a. Sf -->^(l+1) Sf'  and  Sf' sim_B S'                                           [ 27b, 32b1 ]
* BRZ-UNTAKEN.B1 complete

subsubcase B2: . |- Rf_val(ri) =/= R_val(ri)   -  blue fault does affect intention value and faulty version recovers

28b2. . |- Ez'?Ef':Et' = Et'                                                           [ Exp Conditional Lemma, (Exp Eq Transitivity, 15, 25) ]
29b2. . |- Rf_val(ri) =/= Et'                                                          [ Exp Eq Transitivity, 22, 28b2, subcase assumption ]
30b2. |-cf Sf'                                                                         [ Lemma Preservation-Brz-Possible-Elevation, 10,
                                                                                          f=B, 29b, 29b2 ]
31b2. Sf' -->* recover(h,l,Rf_val(rt))                                                 [ CF Lemma, 30b2 ]
32b2. Sf -->^(l+1) Sf'  and  Sf' -->* recover(h,l,R_val(rt))                           [ 27b, 31b2, 27b ]
* BRZ-UNTAKEN.B2 complete



SUBCASE BRZ-UNTAKEN.G: c = G

25g. Rf(ri) = R(ri)
26g. Rf[ri -> R Rf_val(ri)] sim_G R[ri -> R R_val(ri)]                                 [ (sim-R), 9, (sim-val), 25g ]


subsubcase G1:  Rf_val(rz) =/= 0 and . |- Et = Ela + 1  -  faulty computation falls through when original branches,
                                                             but it's ok because branch target is same as fallthru

26g1. . |- R_val(rt) = l + 1                                                           [ subcase assumption, 24 ]
27g1. Sf -->_0 (C,(h,l,l+1), Rf[ri -> R Rf_val(ri)], C(l+1))                           [ (brz-taken), subcase assumption, p2 ]
28g1. |-G (C,(h,l,l+1), Rf[ri -> R Rf_val(ri)], C(l+1)))                               [ Lemma Preservation-Brz-Possible-Elevation, 10,
                                                                                          f=G, 27g1, subcase assumptions, 15 ]
29g1. (C,(h,l,l+1), Rf[ri -> R Rf_val(ri)], C(l+1)) sim_G S'                           [ (sim-S), 29g1, 26g1, 25, 26g ]
30g1. Sf -->^(l+1) Sf' and Sf' sim_c S'                                                [ 27g1, 29g1 ]
* BRZ-TAKEN.G1 complete


subsubcase G2:  Rf_val(rz) =/= 0 and . |- Et =/= Ela + 1  -  faulty computation falls through when original branches off elsewhere

27g2. . |- R_val(rt) = l + 1                                                           [ subcase assumption, 24 ]
28g2. Sf -->_0 (C,(h,l,l+1), Rf[ri -> R Rf_val(ri)], C(l+1))                           [ (brz-untaken), subcase assumption, p2 ]
29g2. |-cf (C,(h,l,l+1), Rf[ri -> R Rf_val(ri)], C(l+1))                               [ Lemma Preservation-Brz-Possible-Elevation, 10,
                                                                                          f=G, subcase assumptions, 15 ]
30g2. Sf' -->* recover(h,l,l+1)                                                        [ CF Lemma, 29g2 ]
31g2. Sf -->^(l+1) Sf'  and  Sf' -->* recover(h,l,l+1)                                 [ 28g2, 30g2 ]
* BRZ-TAKEN.G2 complete


subsubcase G3:  Rf_val(rz)  =  0 and . |- Rf_val(rt) = Et  -  faulty computation takes the same branch as original

26g3. . |- Rf_val(rt) = R_val(rt)                                                      [ Exp Eq Transitivity, subcase assumption, 18 ]
27g3. Sf -->_0 (C,(h,l,Rf_val(rt)),Rf[ri -> R Rf_val(ri)],C(Rf_val(rt)))               [ (brz-taken), subcase assumption, (p2, 26g3) ]
28g3. |-G Sf'                                                                          [ Lemma Preservation-Brz-Possible-Elevation, 10,
                                                                                          f=G, subcase assumptions, 15 ]
29g3. Sf' sim_c S'                                                                     [ (sim-S), 25, 29g3, 26g2, 26g ]
30g1. Sf -->^(l+1) Sf' and Sf' sim_c S'                                                [ 27g1, 29g1 ]
* BRZ-TAKEN.G3 complete


subsubcase G4:  Rf_val(rz)  =  0 and . |- Rf_val(rt) =/= Et   and  Rf_val(rt) in Dom(C)  -  faulty computation branches off elsewhere in C

26g4. Sf -->_0 (C,(h,l,Rf_val(rt)),Rf[ri -> R Rf_val(ri)],C(Rf_val(rt)))               [ (brz-taken), subcase assumptions ]
27g4. |-cf Sf'                                                                         [ Lemma Preservation-Brz-Possible-Elevation, 10,
                                                                                          f=G, subcase assumptions, 15 ]
28g4. Sf' -->* recover(h,l,Rf_val(rt))                                                 [ CF Lemma, 27g4 ]
29g4. Sf -->^(Rf_val(rt)) Sf'  and  Sf' -->* recover(h,l,Rf_val(rt))                   [ 26g4, 28g4 ]
* BRZ-TAKEN.G4 complete


subsubcase G5:  Rf_val(rz)  =  0 and . |- Rf_val(rt) =/= Et   and  Rf_val(rt) not in Dom(C)  -  faulty computation craps out
26g5. Sf -->_0 hw-error(Rf_val(rt))                                                    [ (brz-hw-error), subcase assumptions ]
* BRZ-TAKEN.G5 complete
```

```
CASE JMP:
~~~~~~~~~

R_val(rt) in Dom(C)
------------------------------------------------------------------------------- (jmp)
(C, h, R, jmp rt) -->_0 (C, (h,R_val(rt)), R[ri -> R R_val(ri)], C(R_val(rt)))

Similar to BRZ cases
*
```

Fault Tolerance Theorem

```
Fault Tolerance Theorem
-----------------------


If  |- S  and  S -->^h S'

then either (1) S -->^h_1 Sf  and  Sf sim_c S'                                     // fault wasn't detected
        (2) S -->^hf_1 hw-error(S.h,hf) and  hf prefix of h                        // hardware caught error and <= 1 wrong visited
        (3) S -->^hf_1 recover(S.h,hf)  and  hf prefix of h                        // recover called even though no cf fault
        (4) S -->^hf_1 recover(S.h,hf)  and  hf = (h1,l') and h = (h1,l,h2)        // recover called after cf fault
                                                                                      (only one wrong block visited)


Proof:
------
Case on the structure of S -->^h S'

CASE PROG-EXEC-BLK:
~~~~~~~~~~~~~~~~~~~

S -->* MS
------------------------- (prog-exec-blk)
S -->^() MS

1. Exists c. S -->*_1 Sf and Sf sim_c MS                                        [ Fault Introduction Lemma, p1 ]
 OR S -->*_1 recover(S.h)
subcase on 1.

Subcase A:
aa1. S -->*_1 Sf
aa2. Sf sim_c MS
2a. S -->^() Sf                                                             [ (prog-exec-blk), aa1 ]
3a. S -->^() and Sf sim_c MS                                                [ 2a, aa2 ]
* PROG-EXEC-BLK.A complete

Subcase B:
ab1. S -->*_1 recover(S.h)
2b. S -->^() recover(S.h)                                                   [ (prog-exec-blk), ab1 ]
3b. S -->^() recover(S.h) and () prefix of ()                               [ 2b ]
* PROG-EXEC-BLK-B complete


CASE PROG-EXEC-SEQ-HW-ERROR:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~

S -->^h S'   S' -->_0 hw-error
-------------------------------------- (prog-exec-seq-hw-error)
S -->^h hw-error

S -/->^h hw-error                                                           [ Non Faulty Execution Lemma ]
subcase does not apply
* PROG-EXEC-SEQ-HW-ERROR complete




CASE PROG-EXEC-SEQ-TRANS-BLK:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

S -->^h S''  S'' -->^l S'''  S''' -->*_k MS
-------------------------------------------- (prog-exec-seq-trans-blk)
S -->^(h,l)_k MS


1. Exists S1,S2,S3 h1, h2.                                                  [ Lemma Prog Exec Split, a2 ]
2. S -->^h1 S1
3. S1 -->^l S2
4. S2 -->* S3
5. S3 -->^h2 S'
6. h = (h1,l,h2).

7. Exists c. S2 -->*_1 Sf  and  Sf sim_c S3                                     [ Lemma Fault Introduction, 4 ]
```

```
   OR S2 -->*_1 recover(S.h,h1,l)

8. Either                                                                [ Faulty Computation Lemma, 7, 5 ]
(8.1) Sf -->^h Sf'  and  Sf' sim_c S'
(8.2) Sf -->^hf hw-error(S.h,h1,l,hf) and  hf prefix of h2
(8.3) Sf -->^hf recover(S.h,h1,l,hf)  and  hf prefix of h2
(8.4) Sf -->^hf recover(S.h,h1,l,hf)  and  hf = (h',l') and h = (h',l,h'')

9. Either                                                                [ Lemma Prog Exec Join, 8 ]
(8.1) S -->^h_1 Sf'  and  Sf' sim_c S'
(8.2) S -->^hf hw-error(S.h,h1,l,hf) and  (h1,l,hf) prefix of (h1,l,h2)
(8.3) S -->^hf recover(S.h,h1,l,hf) and  (h1,l,hf) prefix of (h1,l,h2)
(8.4) S -->^hf recover(S.h,h1,l,hf)  and  hf = (h1,l') and h = (h1,l,h2)
* PROG-EXEC-SEQ-TRANS-BLK complete


* FAULT TOLERANCE THEOREM COMPLETE



==================================================================================================================================================
==================================================================================================================================================


Faulty Computation Lemma
-------------------------
If  Sf sim_c S  and  S -->^h S'

then either (1) Sf -->^h Sf'  and  Sf' sim_c S'                      // fault wasn't detected
        (2) Sf -->^hf hw-error(S.h,hf) and  hf prefix of h          // hardware caught error and at most one wrong block visited
        (3) Sf -->^hf recover(S.h,hf)  and  hf prefix of h          // recover called even though no cf fault
        (4) Sf -->^hf recover(S.h,hf)  and  hf = (h1,l') and h = (h1,l,h2)  // recover called after cf fault




PROOF: By induction on the structure of S -->^h S'
------

a1. Sf sim_c S



CASE PROG-EXEC-BLK:
~~~~~~~~~~~~~~~~~~~

S -->* MS'
------------------------- (prog-exec-blk)
S -->^() MS'


1. MS' = S'                                                          [ NonFaulty Block Execution Lemma, Inversion of (sim-S), a1 ]
2. Sf -->* Sf' and Sf' sim_c S'                                      [ Block Execution Lemma, a1, p1 ]
   or Sf -->* recover(S.h)
subcase on 2.

SUBCASE PROG-EXEC-BLK.A: Faulty computation executes first block and continues to simulate original
aa1. Sf -->* Sf'
aa2. Sf' sim_c S'
3a. Sf -->^() Sf'                                                    [ (prog-exec-blk), aa1 ]
4a. Sf -->^() Sf' and Sf' sim_c S'                                   [ 3a, aa2 ]
* PROG-EXEC-BLK.A complete

SUBCASE PROG-EXEC-BLK.B: Faulty computation recovers while executing first block
ab1. Sf -->* recover(S.h)
1b. Sf -->^() recover(S.h)                                           [ (prog-exec-blk), ab1 ]
2b. Sf -->^() recover(S.h) and () prefix ()                          [ 1b, ab1 ]
* PROG-EXEC-BLK.B complete




CASE PROG-EXEC-SEQ-HW-ERROR:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~

S -->^h S''   S'' -->_0 hw-error
-------------------------------------- (prog-exec-blk-hw-error)
S -->^h hw-error

1. S -/->_0 hw-error                                                 [ Non Faulty Block Execution Lemma, Inversion of (sim-S), a1]
subcase does not apply
* PROG-EXEC-SEQ-HW-ERROR complete
```

```
CASE PROG-EXEC-SEQ-TRANS-BLK:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

S -->^h S''  S'' -->^l S'''  S''' -->*_k MS'
-------------------------------------------- (prog-exec-seq-trans-blk)
S -->^(h,l)_k MS'

1. MS' = S'                                                           [ NonFaulty Block Execution Lemma, Inversion of (sim-S), a1 ]
2. Sf -->^h Sf''  and  Sf'' sim_c S''                                 [ I.H., a1, p1 ]
   OR Sf -->^hf hw-error(S.h,hf) and  hf prefix of h
   OR Sf -->^hf recover(S.h,hf)  and  hf prefix of h
   OR Sf -->^hf recover(S.h,hf)  and  hf = (h1,l') and h = (h1,l2,h3)
subcase on 2



SUBCASE PROG-EXEC-SEQ-TRANS-BLK.A: First part of faulty execution simulates original
aa1. Sf -->^h Sf''
aa2. Sf'' sim_c S''

3a. Sf'' -->^l  Sf'''  and  Sf''' sim_c S'''                          [ Block Transition Lemma, aa2, p2 ]
  OR Sf'' -->_0 hw_error(S.h,h)
  OR Sf'' -->^l' Sf'  and  Sf' -->* recover(S.h,h,l')
subsubcase on 3a.

SUBSUBCASE PROG-EXEC-SEQ-TRANS-BLK.A1: Faulty execution takes the correct transition...
aa11. Sf'' -->^l  Sf'''
aa12. Sf''' sim_c S'''
3a1. Sf''' -->* Sf'  and  Sf' sim_c S'                                [ Block Evaluation Lemma, aa12, p3 ]
     OR Sf''' -->* recover(S.h,h,l)
subsubsubcase on 3a1.
subsubsubcase PROG-EXEC-SEQ-TRANS-BLK.A1A: ...and continues to simulate the original
aa1a1. Sf''' -->* Sf'
aa1a2. Sf' sim_c S'
4a1a. Sf -->^(h,l) Sf'                                                [ (prog-exec-seq-trans-blk), aa1, aa11, aa1a1 ]
5a1a. Sf -->^(h,l) Sf' and Sf' sim_c S'                               [ 4a1a, aa1a2 ]
* subsubsubcase PROG-EXEC-SEQ-TRANS-BLK.A1A complete
subsubsubcase PROG-EXEC-SEQ-TRANS-BLK.A1B: ...and recovers prematurely
aa1b1. Sf''' -->* recover(S.h,h,l)
4a1b. Sf -->^(h,l) recover(S.h,h,l)                                   [ (prog-exec-seq-trans-blk), aa1, aa11, aa1b1 ]
5a1b. Sf -->^(h,l)  recover(S.h,h,l)    and  (h,l) prefixof (h,l)     [ 4a1b ]
* subsubsubcase PROG-EXEC-SEQ-TRANS-BLK.A1B complete


SUBSUBCASE PROG-EXEC-SEQ-TRANS-BLK.A2: Faulty execution encounters a hw-error
aa21. Sf'' -->_0 hw_error(S.h,h)
3a2. Sf -->^h hw-error(S.h,h)                                         [ (prog-exec-seq-hw-error), aa1, aa21 ]
3a2. Sf -->^h hw-error(S.h,h)  and  h prefixof h                      [ 3a2 ]
* SUBSUBCASE PROG-EXEC-SEQ-TRANS-BLK.A2 complete


SUBSUBCASE PROG-EXEC-SEQ-TRANS-BLK.A3: Faulty execution goes to the wrong place
aa31. Sf'' -->^l' Sf'
aa32. Sf' -->* recover(S.h,h,l')
3a3. Sf -->^(h,l') recover(S.h,h,l')                                  [ (prog-exec-seq-trans-blk), aa1, aa31, aa32 ]
4a3. Sf -->^(h,l') recover(S.h,h,l')  and  (h,l') = (h,l')  and  (S.h,h,l) = ((S.h,h),l,.)  [ 3a3 ]
* PROG-EXEC-SEQ-TRANS-BLK.A3 complete




SUBCASE PROG-EXEC-SEQ-TRANS-BLK.B: First part of faulty execution encounters a hw-error
ab1. Sf -->^hf hw-error(S.h,hf)
ab2. hf prefix of h
3b. Sf -->^hf hw-error(S.h,hf) and hf prefix of (h,l)                 [ ab1, ab2 ]
* PROG-EXEC-SEQ-TRANS-BLK.B complete



SUBCASE PROG-EXEC-SEQ-TRANS-BLK.C: First part of faulty execution recovers prematurely
ac1. Sf -->^hf recover(S.h,h)
ac2. hf prefix of h
3c. Sf -->^hf recover(S.h,h) and hf prefix of (h,l)                   [ ac1, ac2 ]
* PROG-EXEC-SEQ-TRANS-BLK.C complete



SUBCASE PROG-EXEC-SEQ-TRANS-BLK.D: First part of faulty execution heads off on it's own and recovers
ad1. Sf -->^hf recover
ad2. hf = (h1,l') and h = (h1,l2,h3)
3d. Sf -->^hf recover  and  hf = (h1,l') and h = (h1,l2,h3,l)     [ ad1, ad2 ]
* PROG-EXEC-SEQ-TRANS-BLK.D complete
```

Translation Definitions

```
s ::=   x := n
    |  x1 :=  x2 - x3
    |  if x = 0 then s1 else s2
    |  while x=/=0 do s
    |  s1 ; s2

X ::=   .  |  X,x

L  ::=   l1...lm
is ::=   .  |  is; movi rd v  |  is; sub rd ra rb




MACROS:
-------

intendjmp lt =def=                      // lt is target
movi tb B lt; intend tb; movi tg G lt; jmp tg

intendbrz rz lt lf =def=                // rz is cond reg, lf is fallthrough label, lt is target
movi tb B lf; intend tb; movi tb B lt;  intendz rz' tb; movi tg G lt; brz rz tg

check l =def=                           // l is current label
movi tg R l; sub tg tg ri; recovernz tg




 --------
| X |- s |
 --------

x in X
------------ (wf-assign)
X |- x := n

x1 in X   x2 in X  x3 in X
--------------------------- (wf-sub)
X |- x1 := x2 - x3

x in X   X |- s1   X |- s2
----------------------------- (wf-if)
X |- if x = 0 then s2 else s2

x in X   X |- s
------------------------ (wf-while)
X |- while x =/= 0 do s

X |- s1  X |- s2
------------------ (wf-seq)
X |- s1; s2




 ---------
| Blks(s) |  The number of blocks added by translating s
 ---------

Blks(x := n) = 0
Blks(x1 :=  x2 - x3) = 0
Blks(if x = 0 then s1 else s2) = Blks(s1) + Blks(s2) + 3
Blks(while x=/=0 do s) = 3 + Blks(s)
Blks(s1 ; s2) = Blks(s1) + Blks(s2)
```

```
 ----------------
| InitRegFile(X) |  Make the initial register file corresponding to X
 ----------------

X = x1 ... xn
R = r1 -> G 0, r1' -> B 0,..., rn -> G 0, rn' -> B 0,
    ri -> B 0,
    tg -> G 0, tb -> B 0
-------------------------------------------------
InitRegFile(X) = R
```

```
 ---------
| [[X]]_l | Generate type for label l given variable context X
 ---------


X = x1 ... xn
a1...an,ahist,ari,atg1,atb1,atb2,atr1 fresh
D = a1:kint, ..., an:kint, ahist:khist, ari: int, atg:kint, atb:kint
A = ahist o l
G = { r1:<G,int,a1>, r1':<g,int,a1>, ..., rn:<G,int,an>, rn':<G,int,an>,
      ri:<R,check,ari>,
      tg:<G,int,atg>, tb:<B,int,atb> }
--------------------------------------------------------------------------------
[[X]]_l = Forall[D](G,A)
```

```
 ----------------
| X;P1 |- C : P2 |  Code Memory C is well-typed, but labels in P1 may not have corresponding blocks built yet.
 ----------------

Dom(C) = Dom(P2)
Dom(P1) intersect Dom(P2) = emptyset
Forall n in Dom(P1).
  P1(n) = [[X]]_n
Forall n in Dom(P2).
  P2(n) = [[X]]_n
  [[X]]_n = All[D]((G,ri-><R,check,Y>),alpha o n)
  D; (P1 union P2); (G,ri-><R,check,Y>); alpha o n; Y; (P1 u P2) (n+1) |- C(n)
-------------------------------------------------------------------------------------------- (trans-C-t)
X;P1 |- C : P2
```

```
 -------------
| GenPsi(X,L) | Generate the Code Typing for a set of labels
 -------------

 GenPsi(.) = .
 GenPsi(L,l) = GenPsi(L), [[X]]_l
```

```
 -----------------
| X |- (D,G) good |
 -----------------

D |- G
Forall xk in X. G(rk) = <G,tk,Ek> and G(rk') = <B,tk,Ek'> and D |- Ek = Ek'
G(ri) = <c,ok,Ei>
------------------------------------------------------------------------------------- (GD-good)
X |- (D,G) good
```

```
 ------------------
```

```
| D;P;G |- is : G' |    Strings together instances of D;P;G |- i : G'  to get the effect of a sequence of instructions
 ------------------

----------------- (is-empty-t)
D;P;G |- . : G


D;P;G |- i : G'  D;P;G' |- is : Gs
----------------------------------- (is-seq-t)
D;P;G |- i;is : Gs




 --------------
| X |- is good |   Good instruction sequences always duplicate code and use valid registers
 --------------

-------------- (is-empty-good)
X |- empty good

xd in X   X |- is good
------------------------------------- (is-movi-good)
X |- is; movi rd G n; movi rd' B n good

xd in X   X |- is good
xa in X   xb in X
----------------------------------------- (is-sub-good)
X |- is; sub rd ra rb; sub rd' ra' rb' good






 ----------------------
| X |- (L,C,is,l) good |
 ----------------------


Let P1 = GenPsi(L,l)
Let P2 = GenPsi(Dom(C))
X; P1 |- C : P2
X |- is good
---------------------------------------------------------------------------- (partial-trans-ok)
X |- (L,C,is,l) good







 ----------------------------------------
| [[X |- s]] (L,C,is,l) = (L',C',is',l') |   Statement Translation
 ----------------------------------------


is' = is; movi rk G v; movi rk' B v
----------------------------------------------------------------- (trans-assign)
[[X |- xk := v ]] (L,C,is,l) = (L,C,is',l)


is' = is; sub rk rm rn; sub rk' rm' rn'
----------------------------------------------------------------- (trans-sub)
[[X |- xk := xm - xn ]] (L,C,is,l) = (L,C,is',l)


let lf = l+1                                    // fallthrough (false) label
let lt = lf + Blks(s1)                              // target (true) label
let lj = lt + Blks(s2)                              // join label
let bl = (check l); is; intendzbrz rk lf lt
[[X |- s1]] ( (L,lf), C[l -> bl], ., lt ) = (Lt', Ct', ist', lt')
[[X |- s2]] ( (L,lt), C[l -> bl], ., lf ) = (Lf', Cf', isf', lf')
let C' = (Ct' union Cf')[ lt' -> check lt'; ist'; intendjmp lj ][ lf' -> check lf'; isf'; intendjmp lj ]
-------------------------------------------------------------------------------------------------------------- (trans-if)
[[X |- if xk = 0 then s1 else s2 ]] ( L, C, is, l ) = (L,C',.,lj)


let lb = l + 1                                  // beginning label
let ls = lb + 1                                 // body (s) label
```

```
let le = ls + Blks(s)                               // end label
let bl = check l; is; intendjmp lb
let bb = (check lb); intendzbrz rk le ls
[[X |- s]] ( (L,le), C[l -> bl][lb -> bb], ., ls ) = (Ls', Cs', iss', ls')
let bs = check ls'; iss'; intendjmp lb
let C' = Cs'[ ls' -> bs ]
------------------------------------------------------------------------------------------ (trans-while)
[[X |- while xk =/= 0 do s ]] ( L, C, is, l ) = (L, C', . le)



[[X |- s1]] ( L, C, is, l ) = (L1', C1', is1', l1')
[[X |- s2]] ( L1', C1', is1', l1') = (L', C', is', l')
--------------------------------------------------------------- (trans-seq)
[[X |- s1;s2 ]] ( L, C, is, l ) = (L', C', is', l')
```

Translation Lemmas

```
==========================================================================================================================
==========================================================================================================================

is Lemma:
---------

If  X |- (D;G) good   and   X |- is good
then  (D;P;G) |- is : G'  and  X |- (D,G') good

Proof by induction on the structure of is:

empty case - G' = G

mov/sub cases - both rk and rk' updated equivalently, new G' reflects these changes




==========================================================================================================================
==========================================================================================================================

Block Addition Lemma:
---------------------

If  X;GenPsi(L,l) |- C : GenPsi(C)  and  [[X]]_l = All[D]((G,ri-><R,check,Y>),alpha o n))
and  D; (P1 union P2); (G,ri-><R,check,Y>); alpha o n; Y; P(n+1) |- b
then X;GenPsi(L) |- C[l -> b] : GenPsi(C,l)

Proof:
------

By inversion and reconstruction of (trans-C-t).



==========================================================================================================================
==========================================================================================================================

Block Construction Lemma:
-------------------------

If X |- (L,C,is,l) ok
then Forall l' in (L,l) or Dom(C).
(1) GenPsi(X,L) |- C[l -> check l; is; intendjmp l'] : GenPsi(X,(Dom(C),l))
(2) if l+1 in (L,l) or Dom(C). GenPsi(X,L) |- C[l -> check l; is; intendbrz rz l'] : GenPsi(X,(Dom(C),l))


Proof:
------


Definitions:

dP1. P1 = GenPsi(L,l)
dP2. P2 = GenPsi(Dom(C))
dP3. P3 = P1 union P2

dD1. let D1 = D,Y:kint,alpha:kseq
dD2. let D2 = D,alpha:kseq

dto1. to = P3(l+1)

dG1. let G1 = G,ri-><R,check,Y>
dG2. let G2 = (G,ri-><R,check,Y>)[tg-><R,int,l>]
dG3. let G3 = (G,ri-><R,check,Y>)[tg-><R,int,l-Y>]
dG4. let G4 = (G,ri-><B,ok,l>)[tg-><R,int,0>]
dG5 = #15
dG6. G5[ tb -> <B,P(l'),l'> ][ tg-> <G,P(l'),l'> ][ ri -> <B,go,l'> ]
dG7 = G5[ tb -> <B,P3(l'),l'> ][ tb -> <B,P3(l+1),l+1> ][ tg -> <G,P3(l'),l'> ][ ri -> <B,goz,Ez'?l+1:l'> ]

dSt. St = E61/a1l',...,E6n/anl',  E6tg/atgl', E6tb/atbl', E6ri/aril', (alpha o l)/ahistl'
```

```
1. X;P1 |- C : P2                                                              [ Inversion of (partial-trans-ok), a1, dP1, dP2 ]
2. <deleted>
3. X |- is good

5. P1(l) = [[X]]_l                                                             [ Inversion of (trans-C-t), 1, dP1 ]
6. P1(l) = All[D1]((G,ri-><R,check,Y>),alpha o l)                              [ Inspection of ([[X]]), 6 ]
7. D |- G ok
8. Forall r' in Dom(G). G(r') =/= <R,t',E'>
9. D |- P(l+1) ok
10. Forall k = 1 to n. G(rk) = <G,int,Ek>
    and G(rk') = <B,int,Ek'> and D |- Ek = Ek'

11. D2 |- G4                                                                   [ dD2, dG4, 7 ]
12. Forall k = 1 to n. G4(rk) = <G,int,Ek>                                     [ dG4, 10 ]
    and G4(rk') = <B,int,Ek'> and D |- Ek = Ek'
13. G4(ri) = <B,ok,l>                                                          [ dG4 ]
14. X |- (D2,G4) ok                                                            [ (GD-good), 5, 11, 12, 13 ]

15. (D2;P3;G4) |- is : G5                                                      [ is Lemma, 14, 3 ]
16. X |- (D2,G5) good

17. D2 |- G5                                                                   [ Inversion of (GD-good), 16 ]
18. Forall xk in X. G(rk) = <G,int,Ek> and G(rk') = <B,int,Ek'> and D |-Ek=Ek'
19. G5(ri) = <c,ok,Ei>

dl'. let l' be a label in L

26. D2 |- S : Dl'                                                              [ dSt, 22, 25 ]
27. D2 |- G6[ ri -> <R,check,Et'>] <= St(Gl')                                  [ (G-subtp), (subtp-reflex), (subtp-int), dG6, 24, dSt ]

28. G5(rz) = <G,int,Ez>  and G5(rz') = <G,int,Ez'>  and D2 |- Ez = Ez'         [ 18 ]

dl1. assume l+1 in L
29. Deconstruct structure of P3(l+1) and build Sf as in 20 - 27 for l'         [ dl', 20 - 27, all code blocks have same type ]
```

```
                            G7(ri) = <B,goz,Ez'?l+1:l'>                  [ dG7 ]
                            D2 |- l+1 = l + 1                            [ ]
                            G7(rz) = <G,int,Ez>                          [ dG7, 28 ]
                            D2 |- Ez = Ez'                               [ dG7, 28 ]
                            G7(rt) = <G,All[Dl'](Gl',Al'),l'>            [ dG7, 21 ]
                            D2 |- l' = l'                                [ ]
                            D2 |- St : Dl'                               [ 26 ]
                            D2 |- G6[ ri -> <R,check,Et'>] <= St(Gl')    [ 27 ]
                            D2 |- alpha o l o l' = S(Al')                [ 23, dS ]
                            D2 |- Sf : Dl1                               [ 29 ]
                            D2 |- G6[ ri -> <R,check,Et'>] <= St(Gl1)    [ 29 ]
                            D2 |- alpha o l o (l+1) = S(Al1)             [ 29 ]
                            ------------------------------------------------------------------ (brz-t)
D2;P3;G5;alphaol;l;to |- movi...intendz : G7        D2;P3;G7;alphaol;l;to |- brz rz tg
------------------------------------------------------------------------------------------------------------------------------
D2;P3;G5;alphaol;l;to |- movi tb B l+1; intend tb; movi tb B (l+1); intend tb; movi tg G l'; intendz rz' tb; brz rz tg
------------------------------------------------------------------------------------------------------------------------ [macro expansion ]
D2;P3;G5;alphaol;l;to |- intendzbrz rz l l'

30. if l+1 in L.      D2;P3;G5;alphaol;l;to |- intendzbrz rz l l'
```

```
                                    G6(ri) = <B,go,l'>                          [ dG6 ]
                                    G6(rt) = <G,P(l'),l'>                       [ dG6 ]
                                    D2 |- l' = l'                               [ ]
                                    D2 |- St : Dl'                              [ 26 ]
                                    D2 |- G6[ ri -> <R,check,Et'>] <= St(Gl')   [ 27 ]
[ (mov-t), (sequence-t), (intend-t), (sequence-t), (mov-t), 19 ]        D2 |- alpha o l o l' = S(Al')           [ 23, dS ]
-----------------------------------------------------------------       --------------------------------------------------- (jmp-t)
D2;P3;G5;alphaol;l;to |- mov tb B l'; intend tb; mov tg G l';  G6       D2;P3;G6;alphaol;l;to |- jmp tg
----------------------------------------------------------------------------------------------------------------------- (sequence-t)
D2;P3;G5;alphaol;l;to |- mov tb B l'; intend tb; mov tg G l';  jmp tg
------------------------------------------------------------------------------------------------------------------------ [macro expansion ]
D2;P3;G5;alphaol;l;to |- intendjmp l'

31. D2;P3;G5;alphaol;l;to |- intendjmp l'
```

```
32. let bend be either intendjmp l' or intendzbrz rz l l'
33. D2;P3;G5;alphaol;l;to |- bend                       [ 30, 31, 32 ]
34. D2;P;G4;alphaol;l;to |- is;bend                     [ (sequence-t), 15, 33 ]




                                                          G3(tg) = <R,int,l-Y>             [ dG3 ]
                                                          G3(ri) = <R,int,Y>               [ dG3 ]
                                                          D2 |- G3/ri/rz ok                [ 7, dG3, dD2]
                                                          D2 |- alpha                      [ dD2 ]
                                                          D2 |- l                          [ dD2 ]
                                                          D2;P;G4;alphaol;l;to |- is;bend  [ 34 ]
                      ------------------------------------------(sub-t)    ------------------------------------------------- (recovernz-t)
tg =/= ri                D1;P3;G2;alphaol;Y;to |- sub tg tg ri : G3          D3,Y;kint;P;G3;alphaol;Y;to |- recovernz tg;is;bend
--------------------------(movi-t)   ------------------------------------------------------------------------------------------- (sequence-t)
D1;P3;G1 |- movi tg R l : G2          D1;P3;G2;alphaol;Y;to |- sub tg tg ri; recovernz tg; is; bend
---------------------------------------------------------------------------------------------------------------------------- (sequence-t)
D1;P3;G1;alphaol;Y;to |- movi tg R l; sub tg tg ri; recovernz tg; is; bend
----------------------------------------------------------------------------------------------------------------------- [macro expansion]
D1;P3;G1;alphaol;Y;to |- check l ;is;bend

35. D1;P3;G1;alphaol;Y;to |- check l; is; intendjmp l'
36. if l+1 in L. D1;P3;G1;alphaol;Y;to |- check l; is; intendzbrz rz l l'

37. P3 = (P1/l) union (P2[l -> [[X]]_l ])                                  [ dP3 ]

38. X;(P1/l) |- C[l -> check l; is; intendjmp l'] : P2[l -> [[X]]_l ]      [ (partial-trans-ok), 1, 37, 35 ]
39. if l+1 in L                                                           [ (partial-trans-ok), 1, 37, 36 ]
    X;(P1/l) |- C[l -> check l; is; intendbrz rz l'] : P2[l -> [[X]]_l ]

40. X;GenPsi(X,L) |- C[l -> check l; is; intendjmp l'] : GenPsi(X,(Dom(C),l))   [ 38, dP1, dP2 ]
41. if l+1 in L.                                                          [ 39, dP1, dP2 ]
    X;GenPsi(X,L) |- C[l -> check l; is; intendbrz rz l'] : GenPsi(X,(Dom(C),l))

* Block Construction Lemma Complete




========================================================================================================================
========================================================================================================================


Partial Trans Lemma
---------------------

If [[X |- s]] (L,C,is,l) = (L',C',is',l')   and  X |- (L,C,is,l) good
then X |- (L',C',is',l') good  and  L = L'


Proof:  By induction on the structure of [[X |- s]] (L,C,is,l) = (L',C',is',l')
------

a2. X |- (L,C,is,l) good



CASE TRANS-ASSIGN:
~~~~~~~~~~~~~~~~~~

is' = is; movi rk G v; movi rk' B v
----------------------------------------------------------------- (trans-assign)
[[X |- xk := v ]] (L,C,is,l) = (L,C,is',l)

d1. P1 = GenPsi(L,l)                                                                  [ Inversion of (partial-trans-ok), a2 ]
d2. P2 = GenPsi(Dom(C))
1. X; P1 |- C : P2
2. X |- is good

10. xk in X                                                                           [ Inversion of (wf-assign), a1 ]
2'. X |- is; movi rk G v; movi rk' B v good                                           [ (is-movi-good), 2, 10 ]

12. X |- (L,C,is',l) good                                                             [ (partial-trans-ok), d1, d2, 1, 2' ]
13. L = L
* TRANS-ASSIGN complete




CASE TRANS-SUB:
~~~~~~~~~~~~~~~~~~

is' = is; sub rk rm rn; sub rk' rm' rn'
```

```
------------------------------------------------------------------ (trans-sub)
[[X |- xk := xm - xn ]] (L,C,is,l) = (L,C,is',l)

d1. P1 = GenPsi(L,l)                                                                [ Inversion of (partial-trans-ok), a2 ]
d2. P2 = GenPsi(Dom(C))
1. X; P1 |- C : P2
2. X |- is good

10. xk in X  and xm in X  and Xn in X                                               [ Inversion of (wf-assign), a1 ]
2'. X |- is; sub rk rm rn; sub rk' rm' rn' good                                     [ (is-sub-good), 2, 10 ]

12. X |- (L,C,is',l) good                                                           [ (partial-trans-ok), d1, d2, 1, 2' ]
13. L = L
* TRANS-SUB complete



CASE TRANS-IF:
~~~~~~~~~~~~~~~~~~

p1. let lf = l+1                                          // fallthrough (false) label
p2. let lt = lf + Blks(s1)                                // target (true) label
p3. let lj = lt + Blks(s2)                                // join label
p4. let bl = (check l); is; intendzbrz rk lf lt
p5. [[X |- s1]] ( (L,lf), C[l -> bl], ., lt ) = (Lt', Ct', ist', lt')
p6. [[X |- s2]] ( (L,lt), C[l -> bl], ., lf ) = (Lf', Cf', isf', lf')
p7. let C' = (Ct' union Cf')[ lt' -> check lt'; ist'; intendjmp lj ][ lf' -> check lf'; isf'; intendjmp lj ]
------------------------------------------------------------------------------------------ (trans-if)
[[X |- if xk = 0 then s1 else s2 ]] ( L, C, is, l ) = (L,C',.,lj)



d1. P1 = GenPsi(L,l)                                                        [ Inversion of (partial-trans-ok), a2 ]
d2. P2 = GenPsi(Dom(C))
1. X; P1 |- C : P2
2. X |- is good

3. X |- ( (L,lt,lf), C, is, l ) good                                        [ Weakening on L, a2 ]
4. X; GenPsi(X,(L,lf,lt)) |- C[ l -> bl ] : GenPsi(X,(C,l))                 [ Block Construction Lemma, 3, p4 ]
5. X |- . good                                                              [ is-empty-good ]
6. X |- ( (L,lf), C[l -> bl], ., lt ) good                                  [ (partial-trans-ok), 4, 5 ]
7. X |- ( (L,lt), C[l -> bl], ., lf ) good                                  [ (partial-trans-ok), 4, 5 ]
8. X |- ((L,lf), Ct', ist', lt') good and {Dom(C),l,lt} subseteq {Dom(Ct'),lt'}     [ I.H., p5, 6 ]
9. X |- ((L,lt), Cf', isf', lf') good and {Dom(C),l,lf} subseteq {Dom(Cf'),lf'}     [ I.H., p6, 7 ]

10. X; GenPsi(L,lf,lt') |- Ct' : GenPsi(Ct')                                [ Inversion of (partial-trans-ok), 8 ]
11. X |- ist' good
12. X; GenPsi(L,lt,lf') |- Cf' : GenPsi(Cf')                                [ Inversion of (partial-trans-ok), 9 ]
13. X |- isf' good

14. {Dom(C),l,lt,lf} subseteq {Dom(Cf'),Dom(Ct'),lt',lf'}                   [ 8, 9 ]
15. X; GenPsi(L,lt',lf') |- (Ct' union Cf') : GenPsi(Ct' union Cf')         [ Block Addition Lemma, 10, 12, 14 ]
6. X; GenPsi(L,lt',lf',lj) |- (Ct' union Cf') : GenPsi(Ct' union Cf')       [ Weakening on L, 15 ]

17. X |- ( (L,lf',lj), (Ct' union Cf'), ist', lt') good                     [ (partial-trans-ok), 16, 11 ]
18. X; GenPsi(L,lf',lj) |-                                                  [ Block Construction Lemma, 17 ]
   (Ct' union Cf')[lt' -> check lt'; ist'; intendjmp lj] : GenPsi(Ct' union Cf', lt')

19. X |- ((L,lj),(Ct'unionCf')[lt'->check lt';ist';intendjmp lj],isf',lf') good     [ (partial-trans-ok, 18, 13 ]
20. X; GenPsi(L,lj) |- C' : GenPsi(C')                                      [ Block Construction Lemma, 19, p7 ]

21. X |- (L,C',.,lj) good                                                   [ (partial-trans-ok), 20, 5 ]
22. {Dom(C),l} subseteq {Dom(C'),lj}                                        [ 8, 9 ]
* TRANS-IF complete



CASE TRANS-WHILE:
~~~~~~~~~~~~~~~~~~

p1. let lb = l + 1                                  // beginning label
p2. let ls = lb + 1                                 // body (s) label
p3. let le = ls + Blks(s)                           // end label
p4. let bl = check l; is; intendjmp lb
p5. let bb = (check lb); intendzbrz rk le ls
p6. [[X |- s]] ( (L,le), C[l -> bl][lb -> bb], ., ls ) = (Ls', Cs', iss', ls')
p7. let bs = check ls'; iss'; intendjmp lb
p8. let C' = Cs'[ ls' -> bs ]
------------------------------------------------------------------------------------------ (trans-while)
[[X |- while xk =/= 0 do s ]] ( L, C, is, l ) = (L, C', . le)


d1. P1 = GenPsi(L,l)                                                        [ Inversion of (partial-trans-ok), a2 ]
d2. P2 = GenPsi(Dom(C))
1. X; P1 |- C : P2
2. X |- is good
```

```
3. X |- ( (L,lb), C, is, l ) good                                                   [ Weakening on L, a2 ]
4. X; GenPsi(X,(L,lb)) |- C[ l -> bl ] : GenPsi(X,(C,l))                            [ Block Construction Lemma, 3, p4 ]
5. X |- ( (L,le,ls), C[l -> bl], ., lb) good                                        [ (partial-trans-ok), 4, (is-empty-t), Weakening on L ]
6. X; GenPsi(X,(L,le,ls)) |- C[l->bl][lb->bb] : GenPsi(X,(C,l,lb))                  [ Block Construction Lemma, 5, p5 ]
7. X |- ( (L,le), C[l -> bl][lb -> bb], ., ls ) good                                [ (partial-trans-ok), 6, (is-empty-t) ]
8. X |- ( (L,le), Cs', iss', ls') good                                              [ I.H, 7, p6 ]
9. {Dom(C),l,lb,ls} subseteq {Dom(Cs'),ls'}

10. GenPsi(X,(L,le)) |- Cs'[ls' -> bs] : GenPsi(Cs',ls')                            [ Block Construction Lemma, 8, p7 ]
11. X |- (L, C', ., le) good                                                        [ (partial-trans-ok), 10, (is-empty-t), p8 ]
12. {Dom(C),l)} subseteq {Dom(C'),le}                                               [ 9 ]
* TRANS-WHILE Complete



CASE TRANS-SEQ:
~~~~~~~~~~~~~~~~~~

p1. [[X |- s1]] ( L, C, is, l ) = (L1', C1', is1', l1')
p2. [[X |- s2]] ( L1', C1', is1', l1') = (L', C', is', l')
--------------------------------------------------------------------------------------------- (trans-seq)
[[X |- s1;s2 ]] ( L, C, is, l ) = (L', C', is', l')

1. X |- (L1', C1', is1', l1') good  and  {Dom(C),l} subseteq {Dom(C1'),l1'}         [ I.H., p1, a2 ]
2. X |- (L',C',is',l') good and {Dom(C1'),l1'} subseteq {Dom(C'),l'}                [ I.H., p2, 1 ]
3. X |- (L',C',is',l')  and  {Dom(C),l} subseteq {Dom(C'),l'}                       [ 1, 2 ]
* TRANS-SEQ complete

** Partial Trans Lemma Complete
```

Translation Theorem

```
Translation Theorem
-------------------

If   X |- s  and  [[X |- s]] (.,.,.,1) = (.,C',is',l')
and InitRegFile(X) = R
then |- (C'[l'-> check l'; is'; intendjmp lh][lh -> check lh; intendjmp lh],0,R,intendjmp 1)


Proof:
------

1. GenPsi(X,1) |- . : .                                                            [ (trans-C-t), def of GenPsi() ]
2. X |- (.,.,.,1) good                                                             [ (partial-trasn-ok), (is-empty-t) ]
3. X |- (.,C',is',l') good                                                         [ Partial Trans Lemma, a2, 2 ]

6. X |- (lh},C',is',l') good                                                       [ 3, Weakening on L ]
7. GenPsi(X,{lh}) |-                                                               [ Block Construction Lemma, 6 ]
     C'[l' -> check l'; is'; intendjmp lh] : GenPsi(C',l')
8. X |- (.,C[l'->...],.,lh) good                                                   [ (partial-trans-good), 7, (is-empty-good)]
9. GenPsi(X,.) |- C'[l'->...][lh->...] : GenPsi(C',l',lh)                          [ Block Construction Lemma, 8 ]

dP. P = GenPsi(Dom(C'[l'-> check l'; is'; intendjmp lh][lh -> check lh; intendjmp lh])

4. X;. |- C'[l'-> check l';is';intendjmp lh][lh -> check lh;intendjmp lh] : P      [ Inversion on (partial-trans-good), 3 ]
1'. |- C'[l'-> check l';is';intendjmp lh][lh -> check lh;intendjmp lh] : P         [ Inversion on (trans-C-t), 4, (C-t),
                                                                                       def of [[X]]_l ]


dG = { r1 -> <G,int,0>, r1' -> <B,int,0>,
       ...,
       rn -> <G,int,0>, rn' -> <B,int,0>,
       ri -> <B,ok,0>,
       tg -> <G,int,0>, tb -> <B,int,0> }

10. Forall r. .;P |- R(r) : G(r)                                               [ a3, def of InitRegFile, dG, (int-t), (rit-t),(val-t) ]
2'. P |- R : G                                                                 [ (R-t), 11 ]

3'. |- 0 : empty o 0


dG'. G' = G[tg -> <G,P(1),1>][tb -> <B,P(1),1>][ri -> <B,go,1>]
dS.  S = 0/a1',...,0/an',  1/atg, 1/atb, 1/ari, (emptyo0)/alpha

13. P(1) = [[X]]_l                                                             [ Inversion of (trans-C-t), 1, dP ]
14. P(1) = All[D1](G1,A1)                                                      [ Inversion of ([[X]]), 13 ]
15. D1 = a1:kint, ..., an:kint, alpha:khist,
         ari: int, atg:kint, atbt:kint, atbf:kint, atr:kint
16. A1 = alpha o 1
17. G1 = { r1:<G,int,a1>, r1':<g,int,a1>,
        ...,
        rn:<G,int,an>, rn':<G,int,an>,
            ri:<R,check,ari>,
            tg:<G,int,atg>, tb:<B,int,atb> }

18. G' = {  r1:<G,int,0>, r1':<B,int,0>,                                       [ dG', dG ]
        ...,
        rn:<G,int,0>, rn':<B,int,0>,
            ri:<B,go,1>,
            tg:<G,P(1),1>, tb:<B,P(1),1> }

19. . |- S : D1                                                                [ dS, 15 ]
20. . |- G'[ri-><R,check,1>] <= S(G1)                                          [ (G-subtp), (subtp-reflex), 17, dS, dG' ]



                                   G'(ri) = <B,go,1>                       [ dG' ]
                                   G'(rt) = <G,All[D1](G1,A1),1>           [ dG' ]
                                   . |- 1 = 1                              [ ]
                                   . |- S : D1                             [ 19 ]
                                   . |- G'[ ri -> <R,check,Et'>] <= S(G1)  [ 20 ]
[ (mov-t), (sequence-t), (mov-t), (sequence-t), (intend-t), 19 ]       . |- empty o 0 o 1 = S(A1)          [ 16, dS ]
-------------------------------------------------------------------    ----------------------------------------------------------- (jmp-t)
```

```
.;P;G;emptyo0;0;undef |- mov tb B 1; intend tb; mov tg G 1; G'      .;P;G';emptyo0;0;undef |- jmp tg
------------------------------------------------------------------------------------------------------------------------------- (sequence-t)
.;P;G;emptyo0;0;undef |- mov tb B 1; intend tb; mov tg G 1; jmp tg
------------------------------------------------------------------------------------------------------------------------- [macro expansion ]
.;P;G;emptyo0;0;undef |- intendjmp 1

6'. .;P;G;emptyo0;0;undef |- intendjmp 1

4'. . |- 0 = 0
5'. . |- 0 = 0


21. |- (C'[l'-> check l'; is'; intendjmp lh][lh -> check lh; intendjmp lh],0,R,intendjmp 1) [ (S-t), 1', 2', 3', 4', 5', 6' ]


* Translation Theorem Complete
```