Known Errata in the Dissertation
Ownership-Based Order Reduction and Simulation in
Shared-Memory Concurrent Computer Systems
by Christoph Baumann, Saarland University, 2014

Maintained by Christoph Baumann
(baumann@wjpserver.cs.uni-saarland.de)
May 9, 2014

1. reported by Geng Chen (Saarland University)
   Definition 14: The second and fourth occurrence of \( C' \) in the definition of \( \text{safe}(C, \tau \alpha) \) should be \( C'' \).
   \[
   \text{safe}(C, \tau \alpha) \overset{\text{def}}{=} \text{safe}(C, \tau) \land \exists C', C'' . C \xrightarrow{\tau} C' \xrightarrow{\alpha} C'' \land \text{safe}_{\text{step}}(C', \alpha)
   \]

2. Figure 18: The stack frame is missing its \( rds \) component.

3. Definition of \( \text{info}_{\text{IL}} \), p.128: The first parameter of \( \text{info}_{\text{IL}}.csro \) is redundant, also it should be called \( \text{crso} \).
   \[
   \text{info}_{\text{IL}}.crso : \text{F}_{\text{name}} \times \mathbb{N} \times \mathbb{B}^5 \rightarrow \mathbb{N}_0
   \]

4. reported by Artem Alekhin (Saarland University)
   Definition 64: In the fifth case of the definition of \( \text{vol}^\pi_{\text{f}}(e) \), when expression \( e \) identifies the field of a \textit{struct}-type variable, we should not give this field the same name as the function \( f \) in whose context we are evaluating \( e \), thus we write \( (e').f' \) instead of \( (e').f \).
   \[
   \text{vol}^\pi_{\text{f}}(e) \equiv \left\{ \begin{array}{l}
   : \ldots \\
   \text{volatile} \in q \lor \text{vol}^\pi_{\text{f}}(e') : \ldots \lor e = (e').f' \land \ldots
   \end{array} \right.
   \]


5. reported by Artem Alekhin (Saarland University)

**Definition 68:** Since the C-IL stack is indexed in ascending order from bottom to top, the top frame has index \( \text{top} \), not 1.

\[
\text{consis}_{C-\text{IL}}^{\text{reg}}(c_{IL}, \pi, \theta, \text{info}_{IL}, h) \overset{\text{def}}{=} \begin{cases} 
(i) \quad h.\text{gpr}(bp) = \text{bin}_{32}(\text{base}(\text{top})) \\
(ii) \quad h.\text{gpr}(sp) = \text{bin}_{32}(\text{base}(\text{top}) - \text{dist}(\text{top})) 
\end{cases}
\]

6. reported by Artem Alekhin (Saarland University)

**Definition 70:** The definition of \( crs_{ai,j} \) is missing the term “.crso” after \( \text{info}_{IL} \) and it contains the typo “\( crs_{base} \)”. The correct definition should read:

\[
crs_{ai,j} \equiv \text{bin}_{32}(crs_{base}i - \text{info}_{IL}.crs_{o}(fi, loc_{i}, ri,j))
\]

Also in the second case of the definition of the C-IL local variable consistency, the callee-save registers of a caller in frame \( i \) must be saved in the callee-save area of the next frame \( i + 1 \).

\[
\mathcal{M}_{E_i}(v_{i,j}) = \begin{cases} 
\vdots \\
h.m_{4}(csa_{i+1,j}) : r_{i,j} \in CS \land i < \text{top} \\
\vdots 
\end{cases}
\]

7. reported by Geng Chen (Saarland University)

**Definition of stackovf,** p.135: Since the C-IL stack is indexed in ascending order from bottom to top, we need index \( \text{top} \) instead of 1.

\[
\text{stackovf}(c_{IL}, \pi, \theta, \text{info}_{IL}) \overset{\text{def}}{=} (\text{base}(\text{top}) - \text{dist}(\text{top})) < msp_{IL}
\]

8. **Definition of the** \( R_{S^{\text{MIPS}}_{\text{C-IL}}} \), p.190: Since every C-IL unit operates its own stack, every sequential simulation needs a different compiler information parameter \( \text{info}_{IL} \) with an individual stack base address. However in the given generalized simulation theory only a single fixed simulation parameter is provided. One would need to extend the theory to provide for individual simulation parameters for each computation unit.

9. reported by Artem Alekhin (Saarland University)

**Theorems 2 and 3:** In the generalized sequential simulation theorem (Theorem 5), we demand that the resulting simulation configurations are in consistency points. This condition should also be present in the special cases for MASM and C-IL. We extend the claim of Theorem 2 by:

\[
(i0) \quad h_{n+1,c.pc} \in A_{cp}^{\text{MASM}}
\]
Note here that all MASM configurations are consistency points wrt. $\text{consis}_{\text{MASM}}$. The claim of Theorem 3 is extended as follows:

$$(v) \quad h_{n+1}.c.pc \in A_{c.p}^{c-\text{IL}} \land cp(c'_I, info_{IL})$$

10. **Proofs of Assumption 2 for MASM and C-IL**: The shared invariant guarantees memory consistency only for the shared and read-only memory. For unshared memory owned by other processes we cannot prove memory consistency from the given hypotheses of Assumption 2. Thus the assumption cannot be discharged for the current formulations of $\text{consis}_{\text{MASM}}$ and $\text{consis}_{\text{C-IL}}$ which demand consistency for all memory besides the stack and code regions.\(^1\) In order to solve the issue, one needs to introduce a set of addresses $BM \subseteq \mathbb{B}^{32}$ (bad memory) in the assembler and compiler information and exclude these addresses from the memory consistency. Naturally the code and stack region should be contained in $BM$ and we redefine software condition $\text{badmemop}$ to forbid all accesses to addresses in $BM$.

In the generalized simulation framework we require $BM$ as a mandatory component of simulation parameters in $P$ and we demand the absence of bad memory operations explicitly next to all other software conditions represented by the $sc$ predicate. In the concurrent simulation, for a process $p$ we also put all addresses that are currently unshared and owned by other processes into $p$’s $BM$ set. Thus we do not need to prove memory consistency for resources that are not visible for $p$. We reformulate the claim of Assumption 2 allowing for the choice of a new $BM$ parameter:

$$\ldots \implies \exists \text{par'} \cdot \text{sim}_p(D'.M, \text{par'}, E'.M)$$

The proof of the concurrent simulation theorem is adapted accordingly. Here one has to use ownership-safety together with the shared invariant to show that the abstract system does not access addresses in $BM$.

11. **Definition 7**: In the definition of $(u', m')$, $\delta_p$ should be just $\delta$.

12. **Definition 71**: Confusion between $rds_i$ (value in the C-IL configuration) and $rds(i)$ (memory content in the stack implementation). The second line should read:

$$(ii) \quad rds_i \neq \perp \implies rds(i) = \begin{cases} 
\text{a} : & rds_i = \text{val}(a, t) \in \text{val}_{ptr} \\
\text{alv} : & rds_i = \text{lref}((v, o), j, t) \in \text{val}_{lref} 
\end{cases}$$

\(^1\)In fact in MASM global memory consistency always holds because macros do not access the global memory. Thus global memory is updated by complete consistency blocks containing only a single write instruction on the ISA and MASM level. For C-IL this is not the case and when a process executes an incomplete consistency block, its owned global memory may differ on the ISA and C-IL level.
13. reported by Artem Alekhin (Saarland University)

**Definition** of one\(\mathcal{IO}(\sigma,\tau)\), p. 158: The case where \(\sigma|_{\text{io}} \neq \varepsilon\) and \(\tau|_{\text{io}} = \varepsilon\) is not excluded by the case split. We need the following statement:

\[
\text{one}\mathcal{IO}(\sigma,\tau) \overset{\text{def}}{=} (\sigma|_{\text{io}} = \varepsilon \iff \tau|_{\text{io}} = \varepsilon) \land \text{one}\mathcal{IO}(\sigma) \land \text{one}\mathcal{IO}(\tau)
\]

14. **Definition** of \(S^n_{\text{C-IL}}\mathcal{IO}\), p. 143: In an assignment of the form \(v = \ast(p)\) with variable \(v \in V\) and volatile pointer expression \(p\), i.e., \(\tau_{\text{C-IL}}^{\pi,\theta}(p) = (\{\text{volatile}\}, \text{ptr}(q,t))\) holds, \(p\) could contain references to more than one volatile pointer variable, e.g., by being the sum of two volatile pointer variables. We must forbid this case in order to ensure that C-IL statements do not reference more than one volatile variable. We could implement this restriction by extending the \(\text{no}_2\text{vol}\) predicate, such that it recurses over an expression, searching for multiple accesses to volatile variables. At the moment \(\text{no}_2\text{vol}^{\pi,\theta}_{\text{C-IL}}(p)\) only asserts that \(p\) is not a volatile pointer expression pointing to a volatile variable. A new definition is given below.

\[
n_{\text{C-IL}}^{\pi,\theta}(e) \equiv \begin{cases} 
n_{\text{C-IL}}^{\pi,\theta}(e') & : e \in \{\ominus e', (t)e', \&(\ast(e'))\} \\
/\text{vol}_{\text{C-IL}}^{\pi,\theta}(e') & : e = \ast(e') \land \tau_{\text{C-IL}}^{\pi,\theta}(e') = (q', \text{ptr}(q,t)) \land \text{volatile} \in q' \\
\& n_{\text{C-IL}}^{\pi,\theta}(e') & : e = e' \land \tau_{\text{C-IL}}^{\pi,\theta}(e') = (q', \text{ptr}(q,t)) \land \text{volatile} \notin q' \\
/\text{vol}_{\text{C-IL}}^{\pi,\theta}(e') \land n_{\text{C-IL}}^{\pi,\theta}(e'') & : e = e' \ominus e'' \land \\
\& \{\text{vol}_{\text{C-IL}}^{\pi,\theta}(e') \land \text{vol}_{\text{C-IL}}^{\pi,\theta}(e'')\} & : e = (e' \ominus e'' : \tau) \land \text{zero}(\theta, [e']_{\tau}^{\pi,\theta}) \lor \\
\& e = (e' \ominus e'' : e'') \land \text{zero}(\theta, [e']_{\tau}^{\pi,\theta}) & : \text{otherwise} \\
1 & : \text{otherwise}
\end{cases}
\]

Then for assignments \(e = e'\) or \(e' = e\) to be \(\mathcal{IO}\) steps we would demand:

\[
/\text{vol}_{\text{C-IL}}^{\pi,\theta}(e) \land \text{vol}_{\text{C-IL}}^{\pi,\theta}(e') \land n_{\text{C-IL}}^{\pi,\theta}(e')
\]

An alternative, quick and dirty solution would be to simply forbid pointer arithmetics using volatile variables. To this end we only need to change the last line of the definition slightly, adding \(\ast(e'') \in V\).

\[
\ldots \land /\text{vol}_{\text{C-IL}}^{\pi,\theta}(e) \land \text{vol}_{\text{C-IL}}^{\pi,\theta}(e') \land (e' \in V \lor (e' = \ast(e'')) \land e'' \in V \land \text{no}_2\text{vol}_{\text{C-IL}}^{\pi,\theta}(e''))
\]