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 occurence of C' in the definition of $safe(C, \tau\alpha)$ should be C''.

$$safe(C, \tau\alpha) \stackrel{def}{\equiv} safe(C, \tau) \wedge \exists C', C''.\, C \stackrel{\tau}{\longmapsto} C' \stackrel{\alpha}{\mapsto} C'' \wedge safe_{step}(C', \alpha)$$

2. **Figure 18**: The stack frame is missing its rds component.

3. **Definition** of $info_{IL}$, p.128: The first parameter of $info_{IL}.csro$ is redundant, also it should be called $crso$.

$$info_{IL}.crso : \mathbb{F}_{name} \times \mathbb{N} \times \mathbb{B}^5 \rightharpoonup \mathbb{N}_0$$

4. reported by Artem Alekhin (Saarland University)
   Definition 64: In the fifth case of the definition of $vol_f^{\pi,\theta}(e)$, when expression e identifies the field of a **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 $(e').f$.

$$vol_f^{\pi,\theta}(e) \equiv \begin{cases} \vdots & \\ \textbf{volatile} \in q \vee vol_f^{\pi,\theta}(e') & : \quad \ldots \vee e = (e').f' \wedge \ldots \\ \vdots & \end{cases}$$

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 top, not 1.

$$consis^{regs}_{\text{C-IL}}(c_{IL}, \pi, \theta, info_{IL}, h) \stackrel{def}{\equiv} \begin{array}{ll} \textit{(i)} & h.gpr(bp) = bin_{32}(base(top)) \\ \textit{(ii)} & h.gpr(sp) = bin_{32}(base(top) - dist(top)) \end{array}$$

6. reported by Artem Alekhin (Saarland University)
Definition 70: The definition of $crsa_{i,j}$ is missing the term "$.crso$" after $info_{IL}$ and it contains the typo "$csrbase$". The correct definition should read:

$$crsa_{i,j} \equiv bin_{32}\left(crsbase_i - info_{IL}.crso(f_i, loc_i, r_{i,j})\right)$$

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 $i+1$.

$$\mathcal{M}_{\mathcal{E}i}(v_{i,j}) = \begin{cases} \vdots & \\ h.m_4(csa_{i+1,j}) & : \quad r_{i,j} \in CS \wedge i < top \\ \vdots & \end{cases}$$

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 top instead of 1.

$$stackovf(c_{IL}, \pi, \theta, info_{IL}) \stackrel{def}{\equiv} (base(top) - dist(top)) < msp_{IL}$$

8. **Definition** of the $R_{S^n_{\text{MIPS}}, S^n_{\text{C-IL}}}$, p.190: Since every C-IL unit operates its own stack, every sequential simulation needs a different compiler information parameter $info_{IL}$ 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:

$$\textit{(iv)} \quad h_{n+1}.c.pc \in A^{\text{MASM}}_{cp}$$

Note here that all MASM configurations are consistency points wrt. $consis_{\mathrm{MASM}}$. The claim of Theorem 3 is extended as follows:

$$\textit{(v)} \quad h_{n+1}.c.pc \in A_{cp}^{\text{C-IL}} \wedge cp(c'_{IL}, info_{IL})$$

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 $consis_{\mathrm{MASM}}$ and $consis_{\text{C-IL}}$ 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 $badmemop$ to forbid all accesses to addresses in BM.

    In the generalized simulation framework we require BM as a mandatory component of simulation parameters in $\mathcal{P}$ 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:

$$\ldots \implies \exists par'.\ sim_p(D'.M, par', E'.M)$$

    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'), δ_p should be just δ.

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:

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

[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 $oneI\mathcal{O}(\sigma,\tau)$, p. 158: The case where $\sigma|_{io} \neq \varepsilon$ and $\tau|_{io} = \varepsilon$ is not excluded by the case split. We need the following statement:

$$oneI\mathcal{O}(\sigma,\tau) \stackrel{def}{\equiv} (\sigma|_{io} = \varepsilon \Leftrightarrow \tau|_{io} = \varepsilon) \wedge oneI\mathcal{O}(\sigma) \wedge oneI\mathcal{O}(\tau)$$

14. **Definition** of $S^n_{\text{C-IL}}.\mathcal{IO}$, p. 143: In an assignment of the form $v = *(p)$ with variable $v \in \mathbb{V}$ and volatile pointer expression p, i.e., $\tau_{Q\,c_{IL}}^{\pi,\theta}(p) = (\{\mathbf{volatile}\}, \mathbf{ptr}(q,t))$ 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 $no2vol$ predicate, such that it recurses over an expression, searching for multiple accesses to volatile variables. At the moment $no2vol^{\pi,\theta}_{c_{IL}}(p)$ only asserts that p is not a volatile pointer expression pointing to a volatile variable. A new definition is given below.

$$n2v^{\pi,\theta}_{c_{IL}}(e) \equiv \begin{cases} n2v^{\pi,\theta}_{c_{IL}}(e') & : \quad e \in \{\ominus e', (t)e', \&(*(e'))\} \\ /vol^{\pi,\theta}_{c_{IL}}(e') & : \quad e = *(e') \wedge \tau_{Q\,c_{IL}}^{\pi,\theta}(e') = (q', \mathbf{ptr}(q,t)) \\ & \quad \wedge \mathbf{volatile} \in q' \\ n2v^{\pi,\theta}_{c_{IL}}(e') & : \quad e = *(e') \wedge \tau_{Q\,c_{IL}}^{\pi,\theta}(e') = (q', \mathbf{ptr}(q,t)) \\ & \quad \wedge \mathbf{volatile} \notin q' \\ n2v^{\pi,\theta}_{c_{IL}}(e') \wedge n2v^{\pi,\theta}_{c_{IL}}(e'') & : \quad e = e' \oplus e'' \vee \\ \wedge\, /(vol^{\pi,\theta}_{c_{IL}}(e') \wedge vol^{\pi,\theta}_{c_{IL}}(e'')) & \quad e = (e' \,?\, e'' : \overline{e}) \wedge zero(\theta, [\![e']\!]^{\pi,\theta}_c) \vee \\ & \quad e = (e' \,?\, \overline{e} : e'') \wedge /zero(\theta, [\![e']\!]^{\pi,\theta}_c) \\ 1 & : \quad \text{otherwise} \end{cases}$$

Then for assigments $e = e'$ or $e' = e$ to be $\mathcal{IO}$ steps we would demand:

$$/vol^{\pi,\theta}_{c_{IL}}(e) \wedge vol^{\pi,\theta}_{c_{IL}}(e') \wedge n2v^{\pi,\theta}_{c_{IL}}(e')$$

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 slighty, adding '$e'' \in \mathbb{V}$'.

$$\ldots \wedge /vol^{\pi,\theta}_{c_{IL}}(e) \wedge vol^{\pi,\theta}_{c_{IL}}(e') \wedge (e' \in \mathbb{V} \vee (e' = *(e'')) \wedge e'' \in \mathbb{V} \wedge no2vol^{\pi,\theta}_{c_{IL}}(e''))$$