

CW590, Erratum

Bart Jacobs Frank Piessens

DistriNet Research Group, Department of Computer Science, Katholieke Universiteit Leuven, Belgium
{bart.jacobs,frank.piessens}@cs.kuleuven.be

There is an error in

Bart Jacobs and Frank Piessens. Expressive Modular Fine-Grained Concurrency Specification (Extended Version). Technical Report CW590, Department of Computer Science, Katholieke Universiteit Leuven. July 2010.

The definition of *simple closures* on page 5 is too strict; the concurrent set example of Figure 12 does not comply with it. Therefore, on p. 5, instead of

We say a program has *simple closures* if there exists a partitioning of the procedures into ghost procedures and non-ghost procedures such that command expressions and ghost procedures contain no `exec` commands and no calls of non-ghost procedures. It follows that closures are never recursive. Applying the specification approach of this paper requires only simple closures. As we will see, simple closures admit a very simple proof system.

read

We say a program has *simple closures* if there exists a partitioning of procedure parameters into closure parameters and non-closure parameters such that all `exec` commands are of the form `exec(x)` where x is a closure parameter, and all procedure call argument expressions for closure parameters are either command expressions or closure parameters. Applying the specification approach of this paper requires only simple closures. As we will see, simple closures admit a very simple proof system.

and, further down the column, instead of

This simple approach is sufficient for programs where the graph of procedure calls and closure executions is acyclic. This is the case if the program has simple closures and the procedure call graph is acyclic. The examples of this paper satisfy these constraints.

read

This simple approach is sufficient if the program has simple closures and an acyclic procedure call graph. Indeed, in that case, given a main command, one can inline all procedure calls to obtain an equivalent command that contains no procedure call or closure execution commands; the shape of the proof tree for the original main command will reflect the shape of the main command after inlining.

and, in Appendix A on p. 13, instead of

In particular, in the first subsection we define validity $\text{valid}_\Gamma(c, Q)$ of a command c with respect to a postcondition Q and a function environment Γ .

read

In particular, in the first subsection we define validity $\text{valid}_\Gamma(c, Q)$ of a command c with respect to a postcondition Q and a function environment Γ , where c has no free variables and all free functions are bound by Γ .

and instead of

We assume the program has simple closures and the procedure call graph is acyclic.

The definition is given in Figure 15. It is recursive. The recursion is well-founded; it descends the lexicographic order of 1) ghostness of the command, where non-ghost is greater than ghost; 2) number of procedures that are before the current procedure in the partial order; 3) syntactic size of the command. Specifically, the recursive call in the `exec` case is allowed because the caller is non-ghost and the callee is ghost.

read

We assume the program has simple closures and the procedure call graph is acyclic, and all `exec` commands in c are of the form `exec([c'])`.

The definition is given in Figure 15. It is recursive. The recursion is well-founded; at each recursive call, the size of the command after recursively inlining all procedure calls and replacing `exec([c])` commands by c decreases. Note that the recursive inlining terminates since the procedure call graph is acyclic. Note also that all `exec` commands after inlining are of the form `exec([c])` since the program has simple closures.

and, at the end of Section A.2 on p. 14, add

Seventh, to prove well-definedness of function `valid`, in the Coq proof we pass an explicit *closure level* parameter, which is a natural number that decreases at each `exec` command. Correspondingly, the correctness judgment mentions a closure level; $L, \Gamma \vdash \{P\} c \{Q\}$ means that command c is correct with respect to precondition P , postcondition Q , and function environment Γ , and closure executions in c are nested no more than L levels deep. The correspondence with the approach of the paper is as follows: the closure level of a main command is the maximum static nesting depth of `exec` commands after inlining all procedure calls.

The corrected version has been published as

Bart Jacobs and Frank Piessens. Expressive Modular Fine-Grained Concurrency Specification (Extended Version). Technical Report CW590, Department of Computer Science, Katholieke Universiteit Leuven. July 2010. Revised, August 2010.