



THE UNIVERSITY *of* EDINBURGH

Edinburgh Research Explorer

Synthesising Functional Invariants in Separation Logic

Citation for published version:

Maclean, E, Ireland, A & Grov, G 2012, Synthesising Functional Invariants in Separation Logic. in A Voronkov, L Kovacs & N Bjorner (eds), WING 2010. Workshop on Invariant Generation 2010. EPiC Series in Computing, vol. 1, EasyChair, pp. 183-184.

Link:

[Link to publication record in Edinburgh Research Explorer](#)

Document Version:

Publisher's PDF, also known as Version of record

Published In:

WING 2010. Workshop on Invariant Generation 2010

General rights

Copyright for the publications made accessible via the Edinburgh Research Explorer is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy

The University of Edinburgh has made every reasonable effort to ensure that Edinburgh Research Explorer content complies with UK legislation. If you believe that the public display of this file breaches copyright please contact openaccess@ed.ac.uk providing details, and we will remove access to the work immediately and investigate your claim.



Synthesising Functional Invariants in Separation Logic

Ewen Maclean¹, Andrew Ireland¹ and Gudmund Grov²

¹ Heriot-Watt University
Edinburgh
Scotland

² University of Edinburgh
Edinburgh
Scotland

The CORE system

In [2] we introduced a system which used term synthesis to generate correct loop invariants. The CORE system extends this and is capable of automatically proving fully functional properties of programs involving pointers, by utilising existing systems to eliminate *shape* parts, and extracting *function* from the *structural* statements. The system is capable of synthesising correct functional invariants which allow proofs to succeed. We describe below how we define these terms.

Shape, structure and function

Consider the loop invariant expression for in-place list reversal:

$$data_lseg(\alpha, i, nil) * data_lseg(\beta, j, nil) \wedge \alpha_0 = rev(\beta) \langle \rangle \alpha,$$

where $\langle \rangle$ represents concatenation. We define the following three properties:

Shape This describes purely the shape of the heap, and hence can be described purely as in Smallfoot as $list(i) * list(j)$. as the list segments are null-terminated. There is no information about any data that is contained in the list, purely an indication of the inductive data structures that exist in this part of the heap, in this case linked lists.

Structural This describes the inductive structures on the heap, and gives names for the data contained within them. In this case the structural content is written $data_list(\alpha, i) * data_list(\beta, j)$ in our system, as the list segments are null-terminated. When reasoning about functional properties, it is important that shape information is augmented with logical variables so that this can be extracted.

Functional This describes the *pure* fragment of the statement in separation logic. In this case $\alpha_0 = rev(\beta) \langle \rangle \alpha$ describes the functional content, which importantly relies on the logical variables introduced between the shape and structural content.

Invariant Synthesis

We are able to take invariants such as $data_lseg(\alpha, i, nil) * data_lseg(\beta, j, nil) \wedge \mathcal{F}(\alpha_0, \alpha, \beta)$ and synthesise correct instantiations for \mathcal{F} , and then automatically prove the verification conditions using IsaPlanner [1].

References

- [1] L. Dixon and J. D. Fleuriot. IsaPlanner: A prototype proof planner in Isabelle. In *Proceedings of CADE'03*, volume 2741 of *LNCS*, pages 279–283, 2003.
- [2] Ewen Maclean, Andrew Ireland, Lucas Dixon, and Robert Atkey. Refinement and Term Synthesis in Loop Invariant Generation. In Andrew Ireland and Laura Kovacs, editors, *2nd International Workshop on Invariant Generation (WING'09)*, pages 72–86, 2009.