Sharing Groups Revisited

(Extended Abstract)

Andy King

Department of Electronics and Computer Science,
The University of Southampton,
Highfield, Southampton, S09 5NH, UK.

Abstract interpretation for sharing is an important topic of logic programming. Sharing analysis conventionally infers which variables can be potentially aliased and which variables are definitely grounded. Specifically, sharing (or aliasing) analysis, detects when unification binds a variable to another variable or to a term containing a variable. Applications of sharing analysis are numerous and include: the sound removal of the occur-check; optimisation of backtracking; the specialisation of unification; and perhaps most significantly, the elimination of costly checks in independent and-parallelism [6, 4, 7]. In independent and-parallelism, the body atoms of a clause can be run in parallel if they satisfy certain independence criteria. Independence follows from run-time groundness and sharing checks, but the overhead of the checks can be non-trivial. The overhead be potentially removed, however, by inferring sharing and groundness information at compile-time.

Sharing abstractions often exploit the interplay between groundness and aliasing information, and indeed, accurate analyses are often good at groundness propagation [4, 7]. A synergistic relationship also exists between sharing and type analysis. Type analysis typically infers structural properties of the set of terms to which a variable can be bound and, for instance, might infer that, at a certain program point, a variable is always bound to a list. Greater accuracy can be achieved by performing sharing and type analysis together, than if they are applied separately. By keeping track of type information, that is is inferring structural properties of substitutions, it is possible to infer more accurate sharing information. Conversely, more accurate type information can be deduced if sharing is traced. Curiously, however, type information has only been exploited to a limited extent in sharing analysis.

The accuracy issue has been partly addressed by combining sharing and freeness analysis [6]. Freeness information differentiates between a free variable, a variable which is definitely not bound to non-variable term; and a non-free variable, a variable which is possibly bound to a non-variable term [6]. Freeness information is useful in its own right, in fact it is essential in the detection of non-strict and-parallelism [11, 3]. Non-strict and-parallelism generalises independent (strict) and-parallelism by permitting atoms with shared variables to be evaluated in parallel providing certain freeness and independence criteria are satisfied. The criteria basically ensure that parallel atoms cannot affect each other's search spaces [3].

The work reported here develops the type theme and presents an abstract domain which captures type information (and consequently sharing information) to a higher degree of accuracy than in previous proposals. The domain builds on and, in fact, generalises, the sharing domains of [4, 7] by encoding structural properties of substitutions. Specifically the sharing group domains of [4] and [7] (which are essentially the same) are extended to capture the position, as well as the presence, of variables in terms. Paths are used to distinguish each occurrence of a variable within a term [9]. Previously a sharing group would record which program variables are bound to terms that share a variable. In our revised approach, a sharing group additionally expresses the positions of the shared variable, that is, where the shared variable occurs in the terms which the program variables are bound. This can

dramatically improve precision since it enables the synergistic relationship between sharing and type information to be better exploited.

The accuracy of the technique follows, in part, from the use of paths to represent structural properties of substitutions. However, a path can be of arbitrary length and correspondingly the set of abstract substitutions is a complete but not a finite lattice. Thus a fixed-point computation, of the sort employed in [1, 7], does not necessarily terminate. Hence, before the technique can be bolted into existing abstract interpretation algorithm, it must be further abstracted.

By abstracting different ways, different domains can obtained. So far, a domain for tracking linearity has been derived, which unlike previous techniques [2], possesses good groundness propagation properties [5]. The domain combines the Søndergaard domain [2] and the [4, 7] domains in coherent way as if they were being used together. Our linear approach, however, seems quite general and there is no reason why it could not be applied in the context of a depth-k sharing group domain. The depth-k domain [10, 8], formulated in terms of sharing groups, are perhaps the most obvious application our work. Indeed, a novel depth-k domain has been synthesised which generalises the freeness domain of [6]. The domains of [4, 7] come as the trivial case [5]. Moreover, an initial investigation suggests that the framework leads to a fresh technique for inferring recursive types. This is a subject for further study. The abstract domain thus acts as a intermediate semantic construction, a framework, which conveniently embeds all the information required for sharing and type analysis. Furthermore, the framework not only provides a foundation for combining, refining and deriving analyses, but it simplifies the task of justifying an analysis. Safety can be proven in terms of the framework rather than from first principles [5]. In short, in the context of sharing and typing, the framework closes the gap between the semantics and an analysis.

References

[1] M. Bruynooghe. A practical framework for the abstract interpretation of logic programs. *Journal of Logic Programming*, 10:91–124, 1991.

[2] M. Codish, D. Dams, and E. Yardeni. Derivation and safety of an abstract unification algorithm for groundness and aliasing analysis. In *Eighth International Conference of Logic Programming*, Paris, France, (1991). The MIT Press.

[3] M. Hermenegildo and F. Rossi. Non-strict independent and-parallelism. In *Proceedings of the Seventh International Conference on Logic Programming*, pages 237–252, Jerusalem, 1990. The MIT Press.

[4] D. Jacobs and A. Langen. Static Analysis of Logic Programs. *The Journal of Logic Programming*, pages 154–314, 1992.

[5] A. King. Combined sharing and type analysis. Technical report, Department of Electronics and Computer Science, Southampton University, Southampton, S09 5NH, (1993).

[6] K. Muthukumar and M. Hermenegildo. Combined determination of sharing and freeness of program variables through abstract interpretation. In *Proceedings of the Eighth International Conference on Logic Programming*, pages 49–63, Paris, France, 1991. The MIT Press.

[7] K. Muthukumar and M. Hermenegildo. Compile-time derivation of variable dependency through abstract interpretation. *The Journal of Logic Programming*, pages 315–437, 1992.

[8] A. Taylor. *High Performance Prolog Implementation.* PhD thesis, Basser Department of Computer Science, July (1991).

[9] K. Ueda and M. Morita. A new implementation technique for flat ghc. In *International Conference on Logic Programming*, pages 3–17, Jerusalem, (1990). MIT Press.

[10] R. Warren, M. Hermenegildo, and S. Debray. On the practicality of global flow analysis of logic programs. In *Fifth International Conference and Symposium on Logic Programming.* MIT Press, 1988.

[11] W. Winsborough and A. Wærn. Transparent and-parallelism in the presence of shared free variables. In *Proceedings of the Fifth International Conference and Symposium on Logic Programming*, pages 749–764. The MIT Press, 1988.