

Role Analysis

Viktor Kuncak, Patrick Lam & Martin Rinard

Introduction: Types capture important properties of the objects that programs manipulate, increasing both the reliability and the maintainability of the program. Traditional type systems capture properties (such as the format of data items stored in the fields of the object) that are invariant over the lifetime of the object. But in many cases, properties that do change are as important as properties that do not. In a *typestate system* (first introduced in [13]) the “type” of the object changes as the values stored in its fields change.

We have studied the problem of objects that change their typestate while participating in dynamically linked data structures. In [4] we have introduced the notion of *role* that captures the current purpose of the object in the computation and changes during the lifetime of the object.

Figure 1 presents the *role reference diagram* for a process scheduler. It is a graphical representation of our role definition language. Each box in the diagram denotes a disjoint set of objects that play a given role. The labelled arrows between boxes indicate possible references between the objects in each set. As the diagram indicates, the scheduler maintains a list of live processes. A live process can be either running or sleeping. The running processes form a doubly-linked list, while the sleeping processes form a binary tree. Both kinds of processes have a reference from the `proc` field of a live list node. When a scheduler suspends a process, the object representing the process moves from the running process list into the sleeping process tree. A key observation is that the change of the conceptual purpose (role) of the object in the computation (whether it represents a running or a sleeping process) is correlated with the movement of the object between data structures (whether it participates in the running process list or the suspended process tree).

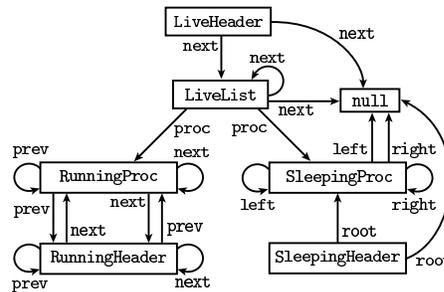


Figure 1: Role Reference Diagram for Scheduler

Approach: The key difficulty with extending a typestate system to linked data structures is that changing role of one object may require changing the roles of the objects connected to it. The fundamental idea in our approach is that the role of each object x depends on the data structures in which it participates. In particular, the role of x depends on the number and the roles of objects y such that y contains a field storing a reference to x . The fact that roles encode such data structure constraints has the following benefits:

1. The properties derived from roles enable the static analysis to check that changing role of an object x is compatible with the roles of the objects y linked to x in the heap.
2. The movement of an object between data structures is reflected in the change of object’s role. Specifying role changes is a powerful technique to summarize the behavior of the program. Roles therefore provide a useful abstraction for program design documentation and program understanding [2].
3. By verifying that the data structure properties are correctly maintained by the program, role analysis prevents violation of certain data structure invariants such as the uniqueness of a parent in a

tree or the correct implementation of forward and backward links in a doubly linked data structure. Roles can therefore be used to specify the shape of data structures. Role analysis is based on a compositional approach of shape analysis [12].

Progress: In [4], we described a role definition language, a programming model for specifying when the role constraints should be satisfied, a procedure specification language with an effect system, and a conservative compositional role analysis algorithm that guarantees that no execution of a program violates the programming model.

One of the difficult problems of role analysis is verifying that procedure preconditions are satisfied because (the problem of *context matching*, [4, Page 13]). In [5] we introduce the notion of *regular graph constraints* to study this problem. We have also introduced *boolean shape analysis constraints* which have better closure properties than regular graph constraints and illustrate the advantages of using formulas to represent summary graphs [9, 8]. We have developed *role logic* [7] as a notation for expressing constraints useful for role analysis and identified two-variable logic with counting [3] and description logics [1] as a useful approach to foundation of role analysis. We are also studying to what extent the memory model of an imperative object-oriented programming language contributes to the difficulty of reasoning about data structures. We are therefore using the theorem prover Isabelle [11] to verify the implementations of purely functional data structures.

Future: We are planning to incorporate role analysis as one of the plugins in a framework of heterogeneous modular analyses [10]. We are studying the decidability and undecidability results of shape analysis and role analysis constraints [6, 9]. We are planning to implement role analysis based on a decision procedure for two-variable logic. We are working on using role analysis and shape analysis to verify data refinement of dynamically allocated data structures [10].

Research Support: This research was supported in part by DARPA Contract F33615-00-C-1692, NSF Grant CCR00-86154, NSF Grant CCR00-63513, and the Singapore-MIT Alliance.

References:

- [1] Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, 2003.
- [2] Brian Demsky and Martin Rinard. Role-Based Exploration of Object-Oriented Programs. In *Proc. 2002 International Conference on Software Engineering*, 2002.
- [3] Erich Grädel, Martin Otto, and Eric Rosen. Two-variable logic with counting is decidable. In *Proceedings of 12th IEEE Symposium on Logic in Computer Science LICS '97, Warschau, 1997*.
- [4] Viktor Kuncak, Patrick Lam, and Martin Rinard. Role analysis. In *Proc. 29th POPL*, 2002.
- [5] Viktor Kuncak and Martin Rinard. Typestate checking and regular graph constraints. Technical Report 863, MIT Laboratory for Computer Science, 2002.
- [6] Viktor Kuncak and Martin Rinard. Existential heap abstraction entailment is undecidable. In *10th Annual International Static Analysis Symposium (SAS 2003)*, San Diego, California, June 11-13 2003.
- [7] Viktor Kuncak and Martin Rinard. On role logic. Technical Report 925, MIT CSAIL, 2003.
- [8] Viktor Kuncak and Martin Rinard. On the boolean algebra of shape analysis constraints. Technical report, MIT CSAIL, August 2003.
- [9] Viktor Kuncak and Martin Rinard. Boolean algebra of shape analysis constraints. In *5th International Conference on Verification, Model Checking and Abstract Interpretation (VMCAI'04)*, 2004.
- [10] Patrick Lam, Viktor Kuncak, and Martin Rinard. On modular pluggable analyses using set interfaces. Technical report, MIT CSAIL, 2003.
- [11] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer-Verlag, 2002.
- [12] Mooly Sagiv, Thomas Reps, and Reinhard Wilhelm. Parametric shape analysis via 3-valued logic. *ACM TOPLAS*, 24(3):217–298, 2002.
- [13] Robert E. Strom and Shaula Yemini. Typestate: A programming language concept for enhancing software reliability. *IEEE Transactions on Software Engineering*, January 1986.