Compositional Proofs of Concurrent Programs

Lawrence C. Paulson

Computer Laboratory, University of Cambridge

1 Previous Research and Track Record

This proposal concerns proving the correctness of programs expressed in the UNITY formalism. Under an existing EPSRC project, Paulson has already developed an environment for verifying UNITY programs. The environment is based on and distributed with Isabelle, a proof assistant developed at Cambridge. The novelty in this proposal is to allow program components to be specified and verified independently of one another. When a system is built from such components, the correctness proof should refer to the properties previously proved rather than regarding the composite system as one giant program. Towards this end, researchers have published many proof methods [4, 12, 15]. By mechanizing these methods and performing case studies, the current project will subject their work to formal scrutiny. Should the methods turn out to work well in practice, then the mechanization will be useful in itself as a tool.

The work will be done within the Cambridge Automated Reasoning Group. Hardware verification was pioneered here by Prof. M. J. C. Gordon and his students. Techniques such as the use of higher-order logic to model hardware spread from the Computer Laboratory into general acceptance. The group's work continues to attract worldwide attention. For example, John Harrison won the Distinguished Dissertation Award for his thesis on verification involving floating-point arithmetic; his recent move to Intel Corp. is evidence that formal proof is relevant to industry.

The group has built two of the most important proof environments used today, namely HOL and Isabelle. Isabelle (originated by Paulson) is a generic theorem prover. It supports interactive proof in several formal systems, including first-order logic, higher-order logic and Zermelo-Frankel set theory. Derived logics can be supported as well as primitive formalisms. Researchers have used Isabelle to support complicated specification languages such as TLA [13] and Z [9].

Several recent projects at Cambridge involve Isabelle:

- *Combining HOL and Isabelle* (SERC ref. GR/H40570), 1992-95. This project applied Isabelle to HOL-style problems, the main application being proof support for Lamport's TLA (Temporal Logic of Actions) [10]. The project produced a detailed comparison between HOL and Isabelle. It supported extensive development of Isabelle, in particular of its *classical rea-*

soner, for automating proofs in predicate logic.

- *Verifying ML Programs using Evaluation Logic* (SERC ref. GR/G53279), 1991–95. This project clarified some of the highly subtle interactions that occur when references to a store interact with higher-order functions. It developed new tools for proving properties of recursively defined domains. Two constructive logics were implemented in Isabelle: Intuitionistic Linear Logic and a variant of Evaluation Logic.

- *Authentication Logics: New Theory and Implementations* (EPSRC ref. GR/K77051), 1996-99. This project is concerned with proving the correctness of security protocols. Originally aiming to extend the authentication logic approach, this project led to a new and highly successful approach to verifying security protocols: the *inductive method*. The actions of agents in the network are modelled as the set of all possible traces. Early work on new authentication logics yielded a detailed analysis of a digital cash protocol.

The most relevant existing project is *Mechanising Temporal Reasoning* (EPSRC ref. GR/K57381), 1995-99. This project is investigating the verification of reactive systems. Much effort has gone into building powerful automatic proof procedures. Originally, the plan was to work equally on UNITY, TLA and model-checking. The work has concentrated on UNITY, mainly because of new developments concerning that formalism.

The Isabelle distribution now includes standard UNITY case-studies. The largest is Anderson's Lift Controller example [1]; others are drawn from the UNITY text [3] and from Misra's recent papers [14, 16, 17]. The full UNITY theory has been mechanized, taking a definitional rather than axiomatic approach. The mechanized theory includes the PSP law (progress-safety-progress), induction principles for progress, and the Completion Theorem. A high level of automation is possible, especially for safety properties and for verifying the meta-theory.

The groundwork has already been laid for the proposed research. Paulson is investigating one theory of program compositions [4] and is mechanizing a case-study by Chandy and Charpentier [2]. EPSRC funding will allow the work to be extended and validated. With the help of a research assistant and a PhD student, more theories can be examined, and substantial case studies can be undertaken. A renewed project will strengthen existing collaborations with the UNITY developers Chandy and Misra, and their groups at Caltech and Austin, respectively.

2 Description of Proposed Research

2 A. Background

UNITY [3] is a formalism for proving the correctness of concurrent systems. It supports a simple model of concurrent programming. There is a single, global state. A program consists of an *initial condition* and a set of *actions*. The latter is a collection of guarded atomic commands that are repeatedly executed under some fairness constraint. UNITY includes a small fragment of temporal logic. While primitive compared with TLA [10], it supplies well-understood methods for proving both safety and progress (liveness) properties. UNITY is of enduring interest: introduced in 1988, it is still investigated by many researchers.

UNITY gives a general treatment of concurrent systems, especially those based on shared variables. The original textbook [3] specified commands to be simultaneous assignments to variables. Researchers later realized that any commands could be allowed, provided a command always terminated and that each program included a **skip** (do nothing) command [16, 17]. UNITY supersedes the ad-hoc formalisms that authors sometimes introduce when verifying concurrent programs. The input language of Murphi, a popular model-checker, is based on UNITY [6].

The Isabelle mechanization regards a program as a triple: the set of program states, the set of allowed initial states, and the set of commands. A command (or action) is a relation on the set of program states. The set of possible finite traces is defined inductively. Predicates on states are identified with sets of states. Similarly, program properties (or specifications) are identified with sets of programs, by analogy with the 'proofs as programs' approach. Identifying program properties with sets allows a smooth formalization of the guarantees operator discussed below.

Safety properties are expressed using the *constrains* operator co (also called next), which expresses the usual precondition-postcondition relationship. The meaning of $F \in A \,\mathsf{co}\, B$ is that if A holds and some action of program F is executed, then B will hold afterwards. (Recall that specifications are identified with sets of programs; equivalently, co may be regarded as a 3-place relation.) If the postcondition equals the precondition then the property holds forever once established and is called *stable*. A stable property that holds initially is called *invariant*.

Progress properties are expressed using the *leads-to* relation. The meaning of $F \in A \leadsto B$ is that if A holds now then B will hold eventually as program F executes. Leads-to properties depend upon which fairness policy is adopted. Misra [16] describes three: minimal progress, weak fairness and strong fairness. Most work assumes *weak fairness*, which guarantees that a command will eventually be executed if it is enabled continuously.

UNITY has been mechanized using other tools. HOL-UNITY [1] is impressive. It provides much automation and includes a graphical interface for outlining progress proofs, but it does not address program composition. Coq-UNITY [8] has its own treatment of composition: a program has an additional component defining the set of actions it may be composed with. This treatment seems limited and does not appear to be used by other researchers.

This proposal starts from the notion of the *union* of two programs: essentially, *parallel composition*. If F and G are two programs defined over the same state space, then $F \parallel G$ is also a program. Its initial condition is the intersection of those of F and G, while its set of actions is the union of F's and G's actions. Our goal is to derive properties of $F \parallel G$ from the abstract properties of F and G without having to consider the actions of $F \parallel G$ explicitly.

We can reason in this compositional way about safety. If $F \in A \textsf{ co } B$ and $G \in A \textsf{ co } B$ then clearly $F \parallel G \in A \textsf{ co } B$, since an action of $F \parallel G$ is either an action of F or an action of G, and all those actions take the precondition A to the postcondition B. However, such reasoning does not work for progress. Even if $F \in A \leadsto B$ and $G \in A \leadsto B$, we cannot expect that $F \parallel G \in A \leadsto B$ because the two programs might interfere with each other. For example, if x is a shared variable, then F might increment x and G might decrement x; each program guarantees that eventually $|x| > 10$, but $F \parallel G$ does not: it can alternately increment and decrement x forever.

Several researchers have proposed methods for reasoning about compositional systems. Chandy and Sanders [4] base their work on *existential* and *universal* properties. A property X is existential provided that if F satisfies X then $F \parallel G$ satisfies X. It is universal provided that if F, G satisfy X then $F \parallel G$ satisfies X. These simple notions allow a compositional approach to progress based upon transient assertions (which are existential) and safety assertions (which are universal), combined using the PSP law.

Chandy and Sanders [4] also introduce guarantees assertions. If $F \in X$ guarantees Y, then for all G, if $F \parallel G$ satisfies X, then $F \parallel G$ also satisfies Y. (Here X and Y are program properties: safety, progress, or even other guarantees properties. Such assertions provide a general means of proving safety and progress properties of systems that take F as a component.

The theory outlined above allows reasoning about $F \parallel G$ where the two components co-operate to make progress. Equally important is the case where F makes progress and G does not interfere. Meier and Sanders [12] give a general treatment of non-interference, superseding the work of several previous authors. Central to their approach is the notion of *progress set*, which generalizes the sufficient conditions of previous non-interference theorems. The literature includes much more in this vein. With the help of mechanical proof tools, this theory can be subjected to formal scrutiny and applied to examples.

2 B. Programme and Methodology

UNITY proofs have traditionally been done by hand. Many unstated assumptions make mechanization difficult. For example, if x is a local variable of F, then obviously the only actions of $F \parallel G$ that can modify x are actions of F. All such 'obvious' properties have to be made explicit and given effective proof support.

The very notion of *state* is problematical. Insisting that a state should be a function from variable names to values is restrictive. Allowing states to remain abstract yields a more elegant formalization, allowing for instance Cartesian product

constructions over state spaces. The elegance is marred when we consider variable sharing, particularly the sort that identifies the variables $F[i].out$ and $G.in[i]$. We now have the choice between complicating the theory to admit such sharing or forcing states after all to be functions over variables. In the later case, the variable names will become equivalence classes. Both approaches are complicated. A third approach to extending and renaming state variables could be based upon the work of Marques [11]. Determining which approach is best will require experimentation. This is a key task of the proposal, and is independent of the various theories of composition.

The next step is to mechanize the theory of Chandy and Sanders [4], which is the most attractive of the existing theories of composition. One feature that distinguishes their work from other 'rely-guarantee' models is that assertions refer to the full system, rather than distinguishing the component from its environment. This facilitates the analysis of systems comprising many components. Paulson has already mechanized parts of this theory, but much work remains, and all such work is preliminary until the notion of state (discussed above) is finalized.

Finally, Charpentier's theory of observation [5] needs to be mechanized. This theory streamlines the treatment of message-passing systems, reducing them to shared variables, namely message histories. At this stage, the project will have produced a mechanized formal environment for verifying compositional systems.

The next step is to evaluate the environment by performing some case studies. The first will be the Allocator of Chandy and Charpentier [2]. In this example, several clients request and return resources (represented by tokens), while an allocator attempts to satisfy the requests. Since the design is compositional, the allocator and a typical client are specified separately. Properties of the allocator are proved under the assumption that clients are well-behaved (for instance, they return resources eventually). A typical client is verified under analogous assumptions. Many clients and the allocator can be combined to form a system, which is verified by reasoning about how the components interact. This proof is expressed in terms of the abstract properties of the components and can be performed before the components themselves have been verified.

The Allocator example admits several variations and generalizations. For example, the clients and allocator can communicate via shared variables (instantaneously) or via a network. Such distinctions are important; exhausting these possibilities will take some time. The expertise and mechanical theories derived from the first case study will allow more elaborate ones to be investigated. There are other examples in the literature, but ideally, new examples will be designed in collaboration with the UNITY teams at Caltech or Austin.

Other theories for verifying compositional systems will also be investigated. One possibility is the progress sets of Meier and Sanders [12]; another is Misra's *closure properties* [15]. More speculatively, another possibility is to apply the mechanical formalisms to Misra's theory of multiprogramming, Seuss [18].

Supporting all this work will be continuous development of Isabelle. Greater automation is a priority: in particular, improved support for the integers.

PROJECT OBJECTIVES

1. To mechanize least one theory for composing systems in UNITY
2. To build a proof environment for verifying compositional systems and to test it by performing case studies
3. To conduct a critical evaluation of the theories investigated

Mechanization of published theories will undoubtedly detect minor flaws in their presentation; some have turned up already. However, 'evaluation' refers to more important matters, such as these: does the theory scale up to significant case studies? Does it support a natural problem decomposition? Does it provide feedback to help locate faults in the system being verified?

2 C. Relevance to Beneficiaries

Beneficiaries include the academic community and industry. Researchers such as Chandy, Misra and Sanders will benefit from having mechanical support for their theories. Developing such support involves scrutinizing the theories, uncovering errors and limitations. Many research groups are investigating temporal logic and reactive systems. UNITY is popular and has much in common with competing formalisms, so results developed under this project will be widely applicable.

Reactive programs are commonplace in industrial products such as operating systems and embedded systems. They are notoriously prone to error. Formal methods have the potential to help, but first we need to have fundamental research of the sort proposed here. A few companies are undertaking such work themselves: HOL-UNITY was developed with the support of Tele Danmark Research.

The project involves further development of Isabelle, which benefits its many users as well as the academic researchers working on related tools.

2 D. Dissemination and Exploitation

The project involves constructing mechanized theories and case studies. The resulting Isabelle proof scripts will be distributed via the Internet. The results of the work will be described in journal and conference papers, in lectures, and on Paulson's extensive Internet site. As Isabelle continues to develop, new releases will continue to be issued.

2 E. Justification of Resources

STAFF

Paulson will work part-time on the project, both managing it and participating in the work itself. He proposes to employ one research assistant full-time to conduct the proofs and to include a PhD studentship. The workplan does not specify a task for the student. The PhD degree is awarded for original research. A typical thesis topic could be to mechanize some other UNITY-related theory (the literature is considerable). The student could have considerable independence. Many other topics, from designing a user interface to building a link-up between Isabelle and an automatic theorem prover, would support the aims of this project.

TRAVEL AND SUBSISTENCE

Conference attendance is essential to keep abreast of developments and to disseminate results. We are requesting funds to attend some of the main conferences such as CADE and CAV. We may wish to attend relevant workshops at Schloß Dagstuhl and elsewhere. We are also requesting funds for visits to other institutions. Our request of £10,800 is based upon three trips per person per year (excluding the student), costing an average of £600 each. The same sum covers three trips to the USA at £1800 each and nine trips at £600.

EQUIPMENT

Isabelle is implemented in Standard ML and is computationally demanding. We request three Dual Pentium IIs with 256MB of RAM, costing £3456 each. (Previously, we would have requested one larger machine and two X terminals; such a configuration represents poor value for money at present.)

A laptop computer will allow working and demonstrations when travelling or at home; we propose a Toshiba Portégé costing £2572. For storing large Isabelle images, we require a large disc costing £2620.

We include standard costs towards Laboratory computing infrastructure: networking, filing systems, email, etc., at £1750 per year. We request £250 consumables per year made up of 2 toner cartridges, 2 boxes paper, 10 backup tapes, workshop supplies, etc.

Under software we have budgeted for Harlequin MLWorks. We can use Standard ML of New Jersey, but it is safer to have a choice of ML systems, especially now that Poly/ML has been withdrawn. They quote a price of $1050 for three licenses and $2000 per year for support, totalling $7050 for three years. This comes to £4270 at current exchange rates. (All costs exclude VAT.)

References

[1] Flemming Andersen, Kim Dam Petersen, and Jimmi S. Pettersson. Program verification using HOL-UNITY. In J. Joyce and C. Seger, editors, *Higher Order Logic Theorem Proving and Its Applications: HUG '93*, LNCS 780, pages 1–15. Springer, 1994.

[2] K. Mani Chandy and Michel Charpentier. An experiment in program composition and proof, 1998. preprint.

[3] K. Mani Chandy and Jayadev Misra. *Parallel Program Design: A Foundation*. Addison-Wesley, 1988.

[4] K. Mani Chandy and Beverly A. Sanders. Reasoning about program composition, 1998. preprint.

[5] Michel Charpentier, Mamoun Filali, Philippe Mauran, Gérard Padiou, and Philippe Quéinnec. Tailoring UNITY to distributed program design. In Rolim [19], pages 820–832.

[6] David L. Dill, Andreas J. Drexler, Alan J. Hu, and C. Han Yang. Protocol verification as a hardware design aid. In *Computer Design: VLSI in Computers and Processors*, pages 522–525. IEEE Computer Society Press, October 1992.

[7] Jim Grundy and Malcolm Newey, editors. *Theorem Proving in Higher Order Logics: Emerging Trends. Supplementary procedings, TPHOLs '98*. Technical report 98-08, Department of Computer Science, Australian National University, 1998.

[8] Barbara Heyd and Pierre Crégut. A modular coding of UNITY in COQ. In von Wright et al. [20], pages 251–266.

[9] Kolyang, T. Santen, and B. Wolff. A structure preserving encoding of Z in Isabelle/HOL. In von Wright et al. [20], pages 283–298.

[10] Leslie Lamport. The temporal logic of actions. *ACM Transactions on Programming Languages and Systems*, 16(3):872–923, May 1994.

[11] François Marques. Program composition in COQ-UNITY. In Grundy and Newey [7], pages 95–104.

[12] David Meier and Beverly Sanders. Composing leads-to properties. Technical Report TR 96-013, Department of Computer and Information Science, University of Florida, 1996.

[13] Stephan Merz. Yet another encoding of TLA in Isabelle. Technical report, Institut für Informatik, TU München, 1997.

[14] Jayadev Misra. A family of 2-process mutual exclusion algorithms. `ftp://ftp.cs.utexas.edu/pub/psp/unity/notes/13-90.ps.Z`, February 1990. Notes on UNITY: 13-90.

[15] Jayadev Misra. Closure properties. At URL `ftp://ftp.cs.utexas.edu/pub/psp/unity/new_unity/closure.ps.Z`, sep 1994.

[16] Jayadev Misra. A logic for concurrent programming: Progress. *Journal of Computer and Software Engineering*, 3(2):273–300, 1995.

[17] Jayadev Misra. A logic for concurrent programming: Safety. *Journal of Computer and Software Engineering*, 3(2):239–272, 1995.

[18] Jayadev Misra. An object model for multiprogramming. In Rolim [19].

[19] José Rolim, editor. *Parallel and Distributed Processing*, LNCS 1388, 1998.

[20] J. von Wright, J. Grundy, and J. Harrison, editors. *Theorem Proving in Higher Order Logics: TPHOLs '96*, LNCS 1125, 1996.

3 Diagrammatic project plan