ON TWO REPRESENTATIONS OF CONCURRENT PROGRAMS

Taras Panchenko, Sunmade Fabunmi

***Abstract**: In the paper we show that under generic conditions, for each fixed input data, the observable behavior of a shared memory concurrent program in the Interleaving Parallel Composition Language (ICPL) extended with the Start and Join operations, which allow creating threads and waiting till thread completion during runtime, is equivalent to the observable behavior of a certain execution of a parallel composition of a fixed number of copies of threads from a fixed finite set (power representation), each of which are representable in the pure ICPL language. This result can be useful for formal verification of concurrent programs which allow dynamic creation of threads during runtime, since it reduces latter problem to the problem of verifying correctness of a certain concurrent program with a fixed set of threads and no dynamic thread creation for each fixed input data.*

***Keywords**: concurrent programming, Interleaving Parallel Composition Language, formal methods, software verification.*

***ITHEA Keywords**: D.1.3 Concurrent Programming, D.2.4 Software/Program Verification.*

***ACM Classification Keywords**: F.3.1 Theory of Computation - LOGICS AND MEANINGS OF PROGRAMS - Specifying and Verifying and Reasoning about Programs, D.2.4 Software - SOFTWARE ENGINEERING - Software/Program Verification.*

Introduction

A large number of software systems used today (e.g. operating systems, database management systems, server software, etc.) are implemented as multithreaded programs with shared memory. Many of such systems form a part of the critical infrastructure of various private and public organizations. Their safety, security, and reliability are of paramount importance, which makes it desirable to obtain the strongest possible guarantees of correctness of their implementation. Such guarantees can be provided by formal methods of software development and verification. Formal methods have been in development for over 50 years, starting from the works of Floyd and Hoare on methods of proving (partial) correctness of sequential programs. Still the task of proving correctness of sequential programs remains difficult and far from being widely applied in software development. The problem of verification of concurrent programs is even more difficult. There are many existing approaches to formal verification of concurrent programs [Ashcroft, 1975; Hoare, 1985; Owicki, 1976; Jones, 1981; Jones, 1983; Xu, 1997; Harel, 1997; Pnueli, 1977; Lamport, 1993; Chandy, 1988; Lamport, 1994; Manna, 1992], but there not so many practical results, most notable of which include formal verification of small specialized microkernels and hypervisors such as seL4 and CertiKOS, which, however, have a far lower complexity than the widely used concurrent software.

Thus, the problem of finding scalable ways of verification of concurrent software is still important.

One way to deal with it is to reduce the problem of verification of programs which, potentially, have a very rich set of runtime behaviors which is difficult to analyze, e.g. the programs which can dynamically create and terminate threads, to a series of problems of verification of programs which have a much simpler, easier to analyze set of behaviors, e.g. programs with a fixed set of threads which cannot create or terminate threads during runtime.

In this paper we propose this kind of reduction. More specifically, we focus on programs expressible in the Interleaving Parallel Composition Language [Panchenko, 2008], which is a convenient formal language for expressing and verifying concurrent software. It was used for formal modeling of real world systems such as a distributed presentation software Infosoft e-Detailing [Kartavov, 2015] and proving their correctness. In this paper we prove that under generic conditions, for each fixed input data, the observable behavior of a shared memory concurrent program in IPCL extended with the Start and Join operations (which allow creating threads and waiting till thread completion during runtime), is equivalent to the observable behavior of a certain execution of a parallel composition of a fixed number of copies of threads from a fixed finite set (which we call the power representation), which are representable in the pure IPCL language and have a much simpler and predictable behavior which is easier to check for correctness.

This allows us, under generic conditions, to reduce the problem of verification of correctness of programs in IPCL with dynamic thread creation and join – to the problem of verification of pure IPCL programs, the availability of this kind of reduction allows us to focus on the simpler cases of concurrent software verification, for which specialized efficient approaches are available, and reduce to them the problems of verification of real-world, complex, large-scale concurrent software.

Interleaving Parallel Composition Language with Start and Join Operations

The syntax and semantics of the pure Interleaving Parallel Composition Language (IPCL) were described in [Panchenko, 2008]. Applications of ICPL were given in [Panchenko, 2006; Panchenko, 2004; Kartavov, 2015, Kartavov2, 2015; Polishchuk, 2015]. The syntax and semantics of the Interleaving Parallel Composition Language with Start and Join operations were proposed and described in [Panchenko, 2017].

Here we recall the main definitions.

In general, IPCL is a family of languages with formally defined syntax and semantics which allow expressing concurrent programs with shared memory, execution of which can be interpreted as interleaving execution of a set of sequential threads.

Basic syntax is defined by the following BNF:

$$P ::= \bar{x} := \bar{e} \mid P_1; P_2 \mid \mathbf{if}\ b\ \mathbf{then}\ P_1\ \mathbf{else}\ P_2 \mid \mathbf{while}\ b\ \mathbf{do}\ P \mid P_1||P_2$$

where

- P_i denotes programs,
- $\bar{e}$ denotes an expression(s) which evaluates to a value(s) (e.g. a number, string, complex data structure, etc.),
- $\bar{x}$ denotes a variable name(s),
- $:=$ denotes the atomic vector assignment operator,
- ; denotes the sequential execution operator,
- **if – then – else** and **while – do** are the usual sequential branching and loop operators,
- || is the composition of parallel execution of two threads.

Formal and detailed definition of semantics of the language can be found in [Panchenko, 2008].

If P is an IPCL program, we denote as P^n (power operator), where *n* is a natural number, the parallel composition of *n* copies of *P*, i.e. a program which performs interleaving execution of *n* threads each of which executes in accordance with *P*.

Pure IPCL can be enriched with the dynamic thread creation (*start*) and joining of threads (*join*) operations which model common multithreading constructs:

$$P ::= \bar{x} := \bar{e} \mid P_1; P_2 \mid \mathbf{if}\ b\ \mathbf{then}\ P_1\ \mathbf{else}\ P_2 \mid \mathbf{while}\ b\ \mathbf{do}\ P \mid P_1||P_2 \mid start(P) \mid join(id)$$

Informally, the $start(P)$ operation takes one argument – a program code *P* and creates a thread which executes a body of *P*. The created thread receives its unique identifier. The $start(P)$ operation can be invoked from any thread and returns immediately after creation of the new thread. Afterwards, execution of both the invoking thread and the newly created thread continues in the arbitrary interleaving fashion.

The $join(id)$ operation takes as an argument a scalar value – an identifier (*id*) of a thread previously created using the $start(P)$ operation and suspends execution of the thread which invokes $join(id)$ until the thread with the given *id* terminates (a thread terminates, if its execution reaches the end of the thread's program code). Afterwards, $join(id)$ resumes execution of the thread which has invoked it.

Formal definition of semantics of the $start(P)$ and $join(id)$ operations can be found in [Panchenko, 2017].

The Main Result

The following theorem uses the notation and terminology defined in [Panchenko, 2008] and [Panchenko, 2017].

Theorem 1. Let *P* be an IPCL program with *start* and *join* operations which takes no input data. Assume that each execution of *P* is terminating. Then the set of lengths of executions of *P* is bounded.

Proof.

Since *P* takes no input data, all executions of *P* start at one program execution state which we denote as q_0 and set of executions of *P* is the set of paths in the labeled transition system, which we denote as *L*, which describes the operational semantics of the IPCL program *P* (with *start* and *join* operations) which starts at *q*. Since at each point of program execution there can be at most finite amount of existing processes and all operations performed by individual processes available in the IPCL language are finitely non-deterministic, the outdegree of any node reachable from q_0 in *L* is finite (i.e. the program can progress from a state *q* to a state from at most finite set of possible successor states that depends on *q*). Suppose that the set of lengths of executions of *P* is unbounded. Then the set of states reachable from q_0 is infinite. Then Konig's lemma implies that *L* has an infinite run starting from q_0 which corresponds to some non-terminating execution of *P*. This contradicts the assumption that each execution of *P* is terminating. Thus, the set of lengths of executions of *P* is bounded.

Theorem is proved.

Corollary. Let *P* be an IPCL program with *start* and *join* operations which takes no input data. Assume that each execution of *P* is terminating.

Then there exist natural number *K* and IPCL programs without *start* and *join* operations $P_1, \dots, P_n$ such that the set of traces of executions of *P* is a subset of the set of traces of executions of $P_1^K || \dots || P_n^K$.

Proof (sketch).

By Theorem 1 the set of lengths of executions of *P* is bounded by some natural number *K*. Then the set of numbers of processes created during each execution of *P* is bounded from above by *K*. Let $P_1, \dots, P_n$ be all procedures in the program *P* in which the *start* operation is replaced by a nondeterministic choice of a number from the set {1,...,*K*}. The nondeterministic choice can be modeled by a set of auxiliary processes:

$$A_i\colon x := n; x := x + 1; n := x;$$

where *n* is a common global variable storing the result and *x* is a local variable for each auxiliary process.

Then it is easy to see that the trace of each execution of *P* corresponds to the trace of some execution of $P_1^K || \dots || P_n^K$ – namely the one in which the nondeterministic choice functions returned the same values as the corresponding *start*() function invocations in *P* (which, as we have shown above, are in range 1,2,...,*K*). Thus, the set of traces of executions of *P* is a subset of the set of traces of executions of $P_1^K || \dots || P_n^K$.

Corollary is proved.

Conclusion

We have shown that under very general conditions, the observable behavior of a shared memory concurrent program in the IPCL language extended with the Start and join operations for a fixed input data is equivalent to the observable behavior of a certain execution of a parallel composition of a fixed number of copies of threads from a fixed finite set (power representation), each of which are representable in the pure IPCL language. This obtained result can be useful for solving the problem of verifying the correctness of concurrent programs which allow dynamic creation of threads. The obtained result reduces this problem to the problem of verification of correctness of a certain concurrent program with a fixed set of threads and no dynamic thread creation for each fixed input data.

We plan to apply this reduction to developed concurrent software systems in the future work.

Acknowledgement

We would like to thank scientific advisors and colleagues for the fruitful discussions held and to appreciate their work. Particularly, Mykola Nikitchenko, Ievgen Ivanov, Dmytro Bui, Volodymyr Redko, and others.

Bibliography

[Ashcroft, 1975] Ashcroft E.A. Proving assertions about parallel programs // Journal of Computer and System Sciences. -- 1975. -- No. 10. -- pp. 110--135

[Chandy, 1988] Chandy K.M., Misra J. Parallel Program Design: A Foundation. -- Reading, MA: Addison-Wesley Publishing Company, 1988. -- 493 p.

[Harel, 1997] Harel D., Pnueli A. On the development of reactive systems // Apt K.R. (ed.) Logics and models of concurrent systems, NATO ASI Series, Vol. F13. -- Springer-Verlag, 1985. -- pp. 477--498

[Hoare, 1969] Hoare, C.A.R. An Axiomatic Basis for Computer Programming. Communications of the ACM. Vol. 12, no. 10, 1969, pp. 576--583

[Hoare, 1985] Hoare C.A.R. Communicating Sequential Processes. -- Prentice Hall International, 1985. -- 238 p.

[Jones, 1981] Jones C.B. Development Methods for Computer Programs Including a Notion of Interference: DPhil. Thesis. -- Oxford University Computing Laboratory, 1981. -- 315 p.

[Jones, 1983] Jones C.B. Specification and Design of (Parallel) Programs // Information Processing Letters: IFIP Information Processing'83 (In IFIP 9th World Congress). -- 1983. -- pp. 321--331

[Kartavov, 2015] Kartavov, M., Panchenko, T. and Polishchuk, N. Properties Proof Method in IPCL Application To Real-World System Correctness Proof. International Journal "Information Models and Analyses". Sofia, Bulgaria, ITHEA. Vol. 4, No. 2, 2015, pp. 142--155

[Kartavov2, 2015] Kartavov, M., Panchenko, T. and Polishchuk, N. Infosoft e-Detailing System Total Correctness Proof in IPCL [in Ukrainian]. Bulletin of Taras Shevchenko National University of Kyiv. Series: Physical and Mathematical Sciences, No. 3, 2015, pp. 80--83

[Lamport, 1993] Lamport L. Verification and Specification of Concurrent Programs // deBakker J., deRoever W., Rozenberg G. (eds.) A Decade of Concurrency, Vol. 803. -- Berlin: Springer-Verlag, 1993. -- pp. 347--374

[Lamport, 1994] Lamport L. The temporal logic of actions // ACM Transactions on Programming Languages and Systems. -- 1994. -- Vol. 16, No. 3. -- pp. 872 -- 923

[Manna, 1992] Manna Z., Pnueli A. The Temporal Logic of Reactive and Concurrent Systems, Specification. -- Berlin: Springer-Verlag, 1992. -- 427 p.

[Nipkow, 2003] Nipkow, T., Paulson, L. C., Wenzel, M. Isabelle/HOL: A Proof Assistant for Higher-Order Logic. Springer, 2003, 226 p.

[Nikitchenko, 1998] M. Nikitchenko, Technical Report IT-TR: 1998-020. A Composition Nominativej Approach to Program Semantics. Technical University of Denmark, 1998

[Owicki, 1976] Owicki S. and Gries D. An Axiomatic Proof Technique for Parallel Programs // Acta Informatica. -- 1976. -- Vol. 6, No. 4. -- pp. 319--340

[Ostapovska, 2016] Ostapovska, Yu., Panchenko, T., Polishchuk, N. and Kartavov, M. Correctness Property Proof for the Banking System for Money Transfer Payments [in Ukrainian]. Problems of Programming. No. 2-3. 2016. pp. 119--132

[Panchenko, 2004] T. Panchenko, The Methodology for Program Properties Proof in Compositional Languages IPCL [in Ukrainian], in Proceedings of the International Conference “Theoretical and Applied Aspects of Program Systems Development” (TAAPSD’2004), 2004, pp. 62–67

[Panchenko, 2006] T. Panchenko, Compositional Methods for Software Systems Specification and Verification [in Ukrainian]. (Panchenko, 2006 Thesis), Taras Shevchenko National University of Kyiv, 2006, 177 p.

[Panchenko, 2007] T. Panchenko, Parallel Addition to Shared Variable Correctness Proof in IPCL [in Ukrainian], Bulletin of Taras Shevchenko National University of Kyiv. Series: Physical and Mathematical Sciences, no. 4, 2007, pp. 187–190

[Panchenko2, 2007] T. Panchenko, Simplified State Model for Properties Proof Method in IPCL Languages and its Usage with Advances [in Ukrainian], in Proceedings of the International Scientific Conference “Theoretical and Applied Aspects of Program Systems Development” (TAAPSD’2007), 2007, pp. 319–322

[Panchenko, 2008] T. Panchenko, The Method for Program Properties Proof in Compositional Nominative Languages IPCL [in Ukrainian], Problems of Programming, no. 1, 2008, pp. 3–16

[Panchenko2, 2008] T. Panchenko, Formalization of Parallelism Forms in IPCL [in Ukrainian], Bulletin of Taras Shevchenko National University of Kyiv. Series: Physical and Mathematical Sciences, no. 3, 2008, pp. 152–157

[Panchenko, 2016] Panchenko, T. Application of the Method for Concurrent Programs Properties Proof to Real-World Industrial Software Systems. Proceedings of the International Conference on ICT in Education, Research, and Industrial Applications (ICTERI'2016), pp. 119--128

[Panchenko, 2017] Panchenko, T., Ivanov Ie., Fabunmi S., Trofimenko Ie., Skidonenko A. Extended Dynamic State and Instances Spawn Model in IPCL. Bulletin of Taras Shevchenko National University of Kyiv, Series Physics & Mathematics, No. 4, 2017

[Polishchuk, 2015] Polishchuk, N., Kartavov, M. and Panchenko, T. Safety Property Proof using Correctness Proof Methodology in IPCL. Proceedings of the 5th International Scientific Conference "Theoretical and Applied Aspects of Cybernetics". Kyiv: Bukrek, 2015, pp. 37--44

[Pnueli, 1977] Pnueli A. The temporal logic of programs // Proc. 18th Annual Symposium on the Foundations of Computer Science (Providence). -- New York: IEEE Computer Society Press, 1977. -- pp. 46--57

[Redko, 1978] V. Redko, Compositions of programs and composition programming [in Russian], Programming, no. 5, 1978, pp. 3–24

[Wiedjik, 2006] Wiedijk F. The Seventeen Provers of the World. Foreword by Dana S. Scott. F. Wiedijk (editor), Lecture Notes in Artificial Intelligence, Vol. 3600, Springer-Verlag Berlin Heidelberg, 2006

[Xu, 1997] Xu Q., de Roever W.-P., He J. The Rely-Guarantee Method for Verifying Shared Variable Concurrent Programs // Formal Aspects of Computing. -- 1997. -- Vol. 9, No. 2. -- pp. 149--174

Authors' Information

Taras Panchenko *– Panchenko, 2006, Associate Professor at the Theory and Technology of Programming Department, Faculty of Computer Science and Cybernetics, Taras Shevchenko National University of Kyiv, 64/13, Volodymyrska Street, Kyiv, Ukraine, 01601*

e-mail: tp@infosoft.ua

Major Fields of Scientific Research: Theory and Technology of Programming, Software Engineering, Software Correctness, Formal Methods

Sunmade Fabunmi *– intern at the Faculty of Computer Science and Cybernetics, Taras Shevchenko National University of Kyiv, 64/13, Volodymyrska Street, Kyiv, Ukraine, 01601*

e-mail: sunmadefabunmi@yahoo.com

Major Fields of Scientific Research: Theory and Technology of Programming, Software Engineering, Software Correctness, Formal Methods