Semantics for a Distributed Programming Language Using SACS and Weakest Pre-Conditions

A V S Rajan, A S Bavan and G Abeysinghe

School of Computing Science, Middlesex University, The Burroughs, London NW4 4BT, United Kingdom
a.rajan, s.bavan, g.abeysinghe@mdx.ac.uk

This paper describes the semantics for a distributed programming language called LIPS (Language for Implementing Parallel Systems). The formalism presented is used for the specification and verification of LIPS programs which uses point-to-point inter-communication to implement distributed systems. The main focus of the paper is to define the semantics of the computational part of LIPS using Dijkstra's weakest preconditions and demonstrate the integration of SACS(Specification of Asynchronous Communication Systems), a variant of SCCS specially developed for specifying point-to-point asynchronous message passing systems, with GCL(Guarded Command Language) using an example.

1. INTRODUCTION

The semantics of a programming language define the meaning to syntactically valid strings while the syntax defines the structural properties of a language. In order to have a better understanding of the language to devise methods for verification and to create the abstract model of the language, it is necessary to construct its syntax and semantics explicitly. Formal languages are most widely used to express the semantics of any programming language. There are three approaches to defining semantics [1]: 1. Operational semantics-describes the actions a running program takes. 2. Denotational semantics-describes the function a program computes 3. Axiomatic semantics-describes the expected behaviour of a program.

Axiomatic semantics expresses the behaviour of a program as a set of assertions (generally, using an assertion language), so that verification of the correctness can be performed to check that the program meets its specification [2]. This document describes a formal model, based on Dijkstra's weakest preconditions [3], for reasoning the computational part of LIPS, a Language for Implementing Parallel/distributed System [5] and combines it with the formal specification, Specification of Asynchronous Communicating Systems (SACS)[4], which is used to specify the communication part of LIPS. LIPS is an asynchronous message passing parallel programming language which has the following features:

- Communication by assignment
- Separation of communication from computation
- A data flow nature coupled with asynchronous message passing
- Portable.

LIPS program is made up of processing nodes (processes) which are linked by unidirectional data channels which carry messages between cooperating nodes. The detailed explanation about LIPS can be found in [5]. A LIPS program is divided into two parts: network definition and nodes definition. The network definition describes the topology of the program by naming each node (representing a process) and its relationships (in terms of input and output data) to other nodes in the system. The message passing between the various nodes is specified using Specification of Asynchronous Communication System (SACS), a synchronous variant of
The second conditional guarded process denoted as G2 has a conditional guard which waits for the input channel deliver to receive its value. The semantic notation using weakest precondition for the conditional guarded process which waits for the deliver signal the values of button and coin may be specified as shown below:

\[
\text{fdeliver} = \text{true} \ (S_{\text{cond}}, \ \text{fdeliver} = \text{false}) \\
\text{S}_{\text{cond}} => (\ \text{deliver} = \text{true}) \\
\quad \quad (\ \text{print(deliver)}, \ \text{deliver} = \text{true})
\]

This synchronises with the SACS specification. The next process/node is machine interface. SACS notation for the MACHINE INTERFACE is:

\[
\text{MACHINE INTERFACE} = \text{coin}?: \ \text{Button}? : \\
\text{drkSig}!: \ \text{MACHINE INTERFACE}
\]

The behaviour of this node can be specified in the form of a conditional guarded process. Conditional guarded process of MACHINE INTERFACE receives the coin and button from CUSTOMER and sends the drkSig to the MACHINE INTERNALS for it to prepare the drink. The semantic notation using the weakest pre-condition for the conditional guarded process which sets the value of drkSig can be specified as shown below:

\[
(f\text{coin} = \text{true} \land f\text{button} = \text{true}) \ (S_{\text{cond}}, f\text{coin} = \text{false} \land f\text{button} = \text{false} \land \text{drkSig} = \text{true}) \\
\text{S}_{\text{cond}} => \\
\quad (\text{coin} = 80 \land \text{button} = \text{true}) \\
\quad \quad (\ \text{S1;} \ \text{drkSig} := \text{true}) \\
\text{S1} => (\text{coin} = 80 \land \text{button} = \text{true}) \\
\quad \quad (\ \text{print(coin), coin} = 80 \land \text{button} = \text{true}) \\
\text{S} => \text{S1(drkSig} := \text{true, drkSig=true})
\]

This synchronises with the list of inputs and output channels of the SACS specification. The next process/node is the MACHINE INTERNALS. SACS notation for the MACHINE INTERNALS is:

\[
\text{MACHINE INTERNALS} = (\text{drkSig}? . \text{trayEmpty}?) : \\
\text{deliver}! : \text{MACHINE INTERNALS}
\]

The behaviour of this node can be specified in the form of a conditional guarded process. The conditional guarded process receives the drkSig from MACHINE INTERFACE and trayEmpty from host and sends the drink to the CUSTOMER with deliver signal. The semantic notation using weakest precondition for this conditional guarded process can be specified as below.

\[
(f\text{trayEmpty} = \text{true} \land f\text{drkSig} = \text{true}) \\
(S_{\text{cond}}, f\text{trayEmpty} = \text{false} \land f\text{drkSig} = \text{false} \land \text{deliver} = \text{true})
\]

\[
\text{S}_{\text{cond}} => \text{trayEmpty} = \text{true} \land \text{drkSig} = \text{true}(\text{S1}; \text{S2}, \text{deliver} = \text{true})
\]

\[
\text{S1} => \text{trayEmpty} = \text{true} \land \text{drkSig} = \text{true} \land \text{deliver} = \text{true}(\text{if } (((\text{trayEmpty}) \land \text{drkSig}) \rightarrow \text{S11}; \text{S1}) \land \text{trayEmpty} = \text{true})
\]

\[
\text{S11} => \text{trayEmpty} = \text{true} \land \text{drkSig} = \text{true} \land \text{deliver} = \text{true}(\text{print(makingdrink), trayEmpty} = \text{true})
\]

\[
\text{S2} => \text{S1(print(DRINK READY), deliver} = \text{true})
\]

This synchronises with the list of inputs and output channels of the SACS specification.

6. CONCLUSION

This paper described the formalisms which will be used for the specification and verification of LIPS programs for the successful point-to-point inter-communication in distributed systems. The paper mainly describes the semantics of computational part of LIPS using Dijkstra’s weakest preconditions and illustrates the integration of SACS with GCL using an example. When SACS defines the semantics by answering the question what the network is, the weakest precondition semantics answers the question of how does the network function. Based on our experience, we found that the weakest precondition based specification interfaces with SACS naturally without producing
any ill effects.

REFERENCES


Dr. A. S. Bavan is a principal lecturer in the School of Computing Science at Middlesex University since 2001. Prior to coming to Middlesex, he has worked in various Universities and industries as a computer scientist.

His research interests include Parallel and Distributed Computing and Artificial Intelligence. He has published over forty research papers in refereed conferences and journals.

Dr. G. K. Abeysinghe is a senior lecturer in the School of Computing Science at Middlesex University since 1997. Prior to coming to Middlesex, she worked as a post doctoral research fellow at University of Southampton, UK.

Her main research interest is in process modelling which includes modelling distributed systems. Other areas of interest are natural language processing and Information systems.