Failure Semantics for the Exchange of Information in Multi-Agent Systems

Frank S. de Boer, Rogier M. van Eijk, Wiebe van der Hoek, and John-Jules Ch. Meyer

Utrecht University, Department of Computer Science
P.O. Box 80.089, 3508 TB Utrecht, The Netherlands
{frankb, rogier, wiebe, jj}@cs.uu.nl

Abstract. In this paper, we present a semantic theory for the exchange of information in multi-agent systems. We define a concurrent programming language for systems of agents that maintain their own private stores of information and that interact with each other by means of a synchronous communication mechanism that allows for the exchange of information. The semantics of the language, which is based on a generalisation of traditional failure semantics, is shown to be fully-abstract with respect to observing of each terminating computation its final global store of information.

1 Introduction

Multi-agent systems are the subject of a very active and rapidly growing research field in both artificial intelligence and computer science. Although there is no formal definition of an agent (in fact this also holds for the notion of an object, which nevertheless has proven to be a very successful concept for the design of a new generation of programming languages), generally speaking, one could say that a multi-agent system constitutes a system composed of several autonomous agents that operate in a (distributed) environment which they can perceive, reason about as well as can affect by performing actions [15]. In the current research on multi-agent systems, a major topic is the development of a standardised agent communication language for the exchange of information. Recently, several agent communication languages have been proposed in the literature, like for instance the language KQML [8]. However none of these communication languages have been given a fully formal account of their semantics [14]. The main contribution of this paper is the introduction of a formal semantic theory for the exchange of information in multi-agent systems.

1.1 Concurrent Programming

We introduce a concurrent programming language that concentrates on the information processing aspects of agents. The underlying computational model of the language has already been introduced in [5]. The basic operations of the
language for the processing of information are the *ask* and *tell* operations of Concurrent Constraint Programming (ccp) [13]. This programming paradigm derives from traditional programming by replacing the *store-as-valuation* concept of von Neumann computing by the *store-as-constraint* model. This computational model is based on a global *store*, represented by a constraint, that expresses partial information on the values of the variables that are involved in computations. The different concurrently operating processes in ccp refine this partial information by adding (*telling*) new constraints to the store. Additionally, communication and synchronisation are achieved by allowing processes to test (*ask*) if the store entails a particular constraint before they proceed in their computation. These basic operations of asking and telling are defined in terms of the logical notions of conjunction and entailment, which are supported by a given underlying constraint system.

In our programming language, however, the global store of Concurrent Constraint Programming is *distributed* among the agents of the system. That is, the above described ask and tell operations of ccp are used by an agent to maintain its own *private* store of information. More precisely, these operations are performed by concurrently executing *threads* within the agent. The agent itself, however, has no direct access to the parts of the global store that are distributed among the other agents in the system. Instead, the agents can only obtain information from each other by means of a synchronous communication mechanism.

This communication mechanism is based on a generalisation of the communication scheme of (imperative) concurrent languages like Communicating Sequential Processes (csp) [11], where the generalisation consists of the exchange of *information*, i.e. constraints, instead of the communication of simple *values*. Abstractly, communication between two agents comprises the supply of an *answer* of one agent to a posed *question* of another agent, and as such presents the basics of a *dialogue*. In particular, posing a question amounts to asking the other agent whether some information holds, while the answering agent in turn provides information from its own private constraint store that is logically strong enough to entail the question. In general, our programming language thus can be viewed upon as a particular model of the concept of *distributed knowledge* as introduced in [9]. The above described communication mechanism then provides a way in which the distributed knowledge of a multi-agent system can become shared among the agents.

### 1.2 Fully Abstract Semantics

The main result of this paper is a compositional semantics for the multi-agent language that is *fully abstract* with respect to observing the final (global) stores of terminating computations. This semantics is based on a generalisation of the *failure* semantics as developed for csp, in which failure sets are employed to give a semantic account of (possible) *deadlock* behaviour [4]. However, whereas in csp a failure set is simply given by a subset of the complement of all the *initial* actions of a process, in our framework, these failure sets are defined in terms of