Partial Order and SOS Semantics for Linear Constraint Programs*

Eike Best¹, Frank S. de Boer² and Catuscia Palamidessi³

¹ Fachbereich Informatik, C.v.O.-Universität Oldenburg, Germany.
   e.best@informatik.uni-oldenburg.de
² Vakgroep Informatica, Universiteit Utrecht, The Netherlands.
   frankb@cs.ruu.nl
³ DISI, Università di Genova, Italy. catuscia@disi.unige.it

Abstract. In this paper we consider linear constraint programming (lcp), a non-monotonic extension of concurrent constraint programming (ccp) which allows to remove information. The entailment relation of a linear constraint system, in terms of which linear constraint programs are defined, is based on the main underlying idea of linear logic: hypotheses in a logical derivation represent physical resources which are consumed, once used in the entailment relation.
We give a semantical analysis of this extension of ccp in terms of the causal relations among occurrences of basic actions (i.e. events). Using a partial order based history model, we define truly concurrent operational and partial order semantic models of lcp. They allow us to compare – and classify – various sublanguages of the proposed extension of ccp (including ccp itself) from the point of view of the degree of parallelism they generate. The two main results of the paper are consistency and completeness of the partial order model with respect to the operational semantics, and thus – as we will argue – its adequacy.

1 Introduction

In the concurrent constraint paradigm [18, 20, 21] (ccp, for short) parallel processes interact via a common store, represented by a constraint or a set of constraints, which expresses some partial information on the values of the variables involved in the computation. One of the most characteristic features of the ccp paradigm is a formalization of the basic operations which allow to update and to query the common store, in terms of the logical notions of consistency, conjunction and entailment supported by a given underlying constraint system. An update of the common store consists of adding (consistently) a constraint, whereas a query consists of checking whether the current store entails some constraint. Thus the computational model of ccp gives rise to a monotonic evolution of the store: more and more information is added in the course of the computation. Moreover, since the definition of the entailment relation of a constraint

* This work has been supported by the HCM Project EXPRESS and by the Esprit Research Project CALIBAN.
The system is based on classical logic, it is difficult to envisage extensions of \(\text{ccp}\) which allow, for example, to remove constraints from the common store. Such extensions would greatly enhance the expressive power of \(\text{ccp}\) and would allow very compact solutions to programming problems which in a purely monotonic setting require considerably elaborate programming techniques.

One of the main contributions of [4] has been the introduction of an operator in \(\text{ccp}\) which consists of removing some constraint from the common store, and which thus gives rise to non-monotonic computations; the resulting paradigm is called *linear constraint programming* (lcp). The concept of store becomes thus similar to the notion of *blackboard* used in Linda [9, 5] and in Shared Prolog [2]. In the paper [1], it is shown that the underlying linear constraint system of lcp is closely related to a subclass of high-level Petri nets, and this relation is used to investigate the complexity of the entailment relation. In the present paper, correct and complete operational and partial order semantics of lcp are given. They are based on a specific history model which allows constraints, events, and their interdependencies, to be described formally.

The structure of this paper is as follows. In section 2 we define the notion of a linear constraint system, upon which rests the mechanism of information removal. Then, in section 3, we describe the lcp language as a suitable process algebra defined on top of a given linear constraint system. Parallel agents operate on a common store by adding and removing tokens. Adding tokens is modeled basically as in \(\text{ccp}\) by *tell* actions. Tokens can be removed by the execution of a *get* action of the form \(\text{get}(c)\), where \(c\) is a token of the underlying linear constraint system. The execution of a *get* \(c\) with respect to a given store consists of removing those tokens from which \(c\) can be derived. Thus in a sense we identify an occurrence of \(c\) with a multiset of tokens from which \(c\) can be derived. If the current store does not contain a multiset of tokens which entails \(c\), then the get action suspends waiting for parallel processes to add the necessary information. On the other hand in case the store contains different multisets entailing \(c\) one is nondeterministically selected and subsequently removed.

In sections 4 and 5, we give a semantical analysis of the proposed non-monotonic extension of \(\text{ccp}\) in terms of the *causal* relations among occurrences of basic actions (*events*). Such a semantical description is of interest because it allows to treat parallelism as a primitive concept, whereas most semantical models describe parallelism in terms of interleaving and thus reduce it to a form of non-determinism.

In section 4 we define a history model with some specific features that allow executions of a linear constraint program to be described. In this model, the elements of a store (i.e. constraints) and the events, as well as their read/write interdependences, are represented explicitly. The set of histories associated with a linear constraint program is defined in SOS (structured operational semantics) style. By abstracting, we derive an operational semantics, which allows us to observe temporal order and simultaneity of actions, from the set of histories.

In section 5 we define our partial order model, the construction of which is based on the basic distinction between two kinds of causal relations among