Partial Evaluation in Constraint Logic Programming

Agata Wrzos-Kamińska

Department of Computer Systems and Telematics,
Norwegian University of Science and Technology, N-7034 Trondheim
email: agata@idt.unit.no

Abstract. Partial evaluation is an optimization technique which aims at specializing general programs in order to improve their efficiency. Within the field of Logic Programming the technique is known as partial deduction. In this paper we generalize the concept of partial deduction so that it applies to the framework of Constraint Logic Programming (CLP). We also lift the main theoretical results on partial evaluation in Logic Programming to the CLP case, thus providing a formal foundation for partial evaluation of constraint logic programs.

1 Introduction

Partial evaluation is an optimization technique which aims at specializing general programs in order to improve their performance. The use of partial evaluation can result in substantial gains of efficiency. Generally, the idea can be explained as follows [10]. If some of the input data are known beforehand, a program can be specialized, and this can be achieved in an automated way. Specialization is done by performing those computations of the program that depend only on the given input arguments, and by generating code for remaining computations which depend on the as yet unavailable data. The resulting specialized version may be not as simple and elegant as the general program, but it will be more efficient.

Partial evaluation allows an advanced and very convenient programming methodology. Namely, one may well write highly parameterized and modularized programs, each of them solving a whole class of similar problems. Such general programs may be inefficient; but with the use of a partial evaluator they may be automatically specialized to any interesting setting of parameters, yielding as many customized, more efficient versions (residual programs) as desired. The approach is excellent for documentation, modification and human usage; it can ease the programming and maintenance effort.

One of the domains where partial evaluation has been successfully applied is Logic Programming (LP). Within the field, applications of the technique include meta-programming, program optimization, explanation-based learning and software engineering.

A great advantage of LP is its declarativeness. However, the paradigm has also its limitations, probably the major of them being the syntactic nature of the domain of computation. This drawback is overcome in Constraint Logic Programming (CLP), a generalization of LP where other kinds of objects besides terms are considered, and where other kinds of relations over objects than equations may be evaluated. In CLP, Logic Programming has been combined with another declarative paradigm, that of constraint solving. An important property of constraints is that they allow to define objects implicitly, by specifying their properties. As indicated in [7], a fundamental weakness of conventional LP programs comes from the fact that they compute results in a form of substitutions, that is strictly of the explicit type. In CLP, implicit representations were introduced via constraints, significantly increasing the expressive power of the language.

In this paper, we generalize the concept of partial deduction as defined for LP [11], [13] so that it applies to the framework of Constraint Logic Programming. We also lift the main theoretical results on partial evaluation in Logic Programming to the CLP case,
thus providing a formal foundation for partial evaluation of constraint logic programs. This should allow to enjoy the virtues of partial evaluation in the powerful paradigm of Constraint Logic Programming, at the same time guaranteeing correctness of transformations.

The reader is assumed familiar with basic concepts of Logic Programming [12]. The rest of the paper is organized as follows. In the next section, we give an introduction on main ideas underlying CLP, its semantics, and the concept of partial deduction. In section 3, we define the concept of partial evaluation for CLP. In section 4, formal results for the declarative and operational semantics of CLP are presented. Finally, in section 5 we summarize the results and discuss some directions for future work.

## 2 Preliminaries

### 2.1 Constraint Logic Programming

In this section, we introduce main ideas that underly the paradigm of CLP [6], [4, 5]. We do it by comparison to conventional LP. The presentation is partially based on [7] and [8].

The key extension of CLP with respect to LP consists in allowing a user's domain of computation besides the Herbrand universe, admitting constraints other than equalities, and replacing unification by constraint solving [6].

In conventional LP, the domain of computation is the Herbrand universe, i.e. the set of ground terms of some first-order language. Note that we can view unification, which is a part of the resolution, as an implicit representation of equations over terms. CLP generalizes this framework, by incorporating constraints and constraint solving methods. The domain of computation is no longer limited to the Herbrand universe. CLP systems were developed that compute over numerous other domains, like sets, various types of graphs, Boolean expressions, integers, rationals, real numbers, or lambda expressions. Usually these are well understood and formalized domains, for which there exist natural algebraic operators (such as intersection/multiplication, union, disjunction, etc.) and basic predicates, typically including equality/isomorphism and different forms of inequalities ($\subset$, $\leq$, $\neq$, etc). These special predicates are examples of (primitive) constraints. Their meaning is assumed known and fixed, as opposed to the "ordinary" LP predicates which are defined via program clauses.

Just like a conventional definite LP program, a CLP program consists of a finite set of Horn clauses, written in the form: $h \leftarrow B$. The atom $h$ is still called the head of the clause, and $B$ stands for what is called the body. In particular, a goal (or query) is a clause of the form: $\leftarrow B$. The difference is that in CLP constraint atoms (or simply: constraints) may be used in the bodies of clauses in addition to the "usual" atoms (or simply: atoms). We will usually represent CLP clauses as

$$h \leftarrow A, C$$

where $A$ is the finite collection of atoms and $C$ is the finite collection (conjunction) of constraints in the body of the clause.

For a single atom $a$ and a finite collection of constraints $C$, a fact in CLP is a clause of the form $a \leftarrow C$ (also called a conditional atom). In addition, we will use the notion of constrained atom to denote a conjunction of the form $a \land C$ (or: $a, C$). The operators of conjunction ($\land$ or $\cdot$) and set union ($\cup$) will often be used interchangeably.

In CLP, ordinary atoms control the resolution procedure, as it is the case in conventional LP. Constraint atoms are accumulated, possibly in a simplified/normalized form, in the so called constraint store.

In order to deal with the more general form of program clauses, a CLP interpreter contains a domain-specific constraint solving engine in addition to a standard resolution module. The job of the constraint engine is to maintain the constraint store in a standard form and to decide, upon request from the resolution engine, whether a collection of constraints is consistent (satisfiable).