Abstract. Constraint programming offers facilities for problem modelling, constraint propagation and search. This paper discusses the resulting benefits for practical applications which exploit these facilities.

The modelling facilities are particularly exploited in applications to verification, both of circuits and of real time control systems. The propagation facilities are exploited in applications involving user feedback and graphical interfaces. The search facilities are exploited in applications such as scheduling and resource allocation, which involve combinatorial problems.

The paper surveys applications under each of these three headings.

Keywords: survey applications

1. Introduction

Constraint programming is newly flowering in industry. Several companies have recently started up to exploit the technology, and the number of industrial applications is now growing very quickly. A recent article in Byte (Pountain, 1995) picked constraint logic programming as the paradigm "likely to gain most in commercial significance over the next 5 years". This survey will seek, by examples, to explain the current success of constraint technology, and, by showing its benefits, to add to that success.

The paper attempts to give a broad coverage of applications so as to indicate areas where constraint programming is appropriate. It is neither a market survey, nor a detailed technical appraisal. Only a sample of applications could be mentioned in the space available, since a complete coverage would require several surveys of this size. Moreover the areas of focus of this survey have inevitably been influenced by the perspective of the author: the space devoted to different areas may sometimes fail accurately to reflect the amount of work that has been done. The author would welcome further surveys to help redress the balance!

1.1. A Historical Perspective

In 1963 Sutherland introduced the Sketchpad system (Sutherland, 1963), a constraint language for graphical interaction. Other early constraint programming languages were Fikes' REF-ARF (Fikes, 1970), Laurière's Alice (Laurière, 1978), Sussman's CONSTRAINTS (Sussman & Steele, 1980) and Borning's ThingLab (Borning, 1981). These languages already offered the most important features of constraint programming:

- declarative problem modelling
- propagation of the effects of decisions
• efficient search for feasible solutions

Each of these three features has been the study of extensive research over a long period. Declarative programming has a long history yielding languages such as LISP, Prolog and other purer functional and logic programming languages, and of course it underpinned the introduction of relational databases and produced SQL which, for all its faults, is today's most commercially successful declarative programming language.

Constraint propagation was used in 1972 for scene labelling applications (Waltz, 1975), and has produced a long line of local consistency algorithms (Montanari 1974, Mackworth 1977, Freuder 1978, Haralick and Elliott 1980, Freuder 82, Mohr and Henderson 1986, Davis 1987, Mackworth and Freuder 1992, Bessière 94).

The topic of search has been at the heart of AI since GPS (Newell & Simon, 1963). Influential ideas were generate and test (Golomb & Baumert, 1965), branch and bound (Lawler & Wood, 1966), the A* algorithm (Hart et al., 1968), iterative deepening (Korf, 1988), tree search guided by the global problem structure (Freuder, 1994), or by information elicited during search (Maruyama et al., 1992), and by intelligent backtracking (Kondrak & van Beek, 1995).

1.2. Technical Overview

Whilst there is "nothing new under the sun", the current flowering of constraint programming owes itself to a generation of languages in which these three features are present in a new architecture that makes them easy to understand, combine and apply. The technology has matured to the point where it is possible to isolate the essential features and offer them as libraries or embed them cleanly in general purpose host programming languages.

For example isolating constraints as libraries has made possible the development of sophisticated constraint-based scheduling systems, see (Zweben & Fox, 1994). More generally there are commercially available libraries supporting constraint handling (ILOG, 1995, COSYTEC, 1995).

On the other hand constraints fit hand in glove with declarative host programming languages. Three of the most influential constraint programming languages were embedded in Prolog (Prolog III (Colmerauer, 1990), CLP(R) (Jaffar et al., 1992) and CHIP (Dincbas et al., 1988a)). Whilst all three system are still developing further (Touraivane, 1995, Kelly et al., 1995, Simonis, 1995), there are many new constraint programming systems emerging (e.g. Codognet et al. 1993, ECLiPSe 1995, Henz et al. 1995, Van Hentenryck et al. 1994, Benhamou et al. 1994). From a theoretical point of view the extension of logic programming to Constraint Logic Programming (CLP) has been very fruitful: for example ALPS (Maher, 1987) - a form of logic programming with guards - was an extremely influential language, becoming the forerunner of the Concurrent Constraints paradigm (Saraswat, 1993). Concurrent constraint programming has in turn provided a very clean model of concurrent and multi-agent computing (Saraswat et al., 1990, Henz et al., 1995). Constraints can also be modelled in terms of information systems (Saraswat, 1992), which allows us to reason about the behaviour of constraint programs at an abstract level. One application of such abstract reasoning, abstract interpretation, is already paying dividends.