download SWI-Prolog, work your way to the first practical session on page 16 of Learn Prolog Now! and your first Prolog queries give their answers. This is a fantastic way to learn a piece of Prolog.

Bart Demoen

doi:10.1017/S1471068408003281

Constraint Logic Programming using ECL^{i}PS^e Krzysztof Apt and Mark Wallace Cambridge University Press, 2007 Hardback: ISBN 9780521866286, Price: £35, 348 pages.

This book is an introduction to constraint logic programming (CLP) in the ECL^{i}PS^e language. It aims to introduce CLP to senior students in Computer Science and also describe the ECL^{i}PS^e language. This book is a good introduction to CLP, which is filled with many examples. The book is also timely as it coincides with the release of ECL^{i}PS^e as open-source software free for everyone to use.

The book is divided into four parts. The first part is an introduction to logic programming while the second part describes the Prolog part of ECL^{i}PS^e. The third part, entitled programming with passive constraints, begins by discussing constraint programming (CP) in theory, and then introduces iteration for ECL^{i}PS^e, control of search, and the `suspend` library which extends ECL^{i}PS^e to allow goals to suspend until variables are fixed. The fourth part, the core of the book, introduces CP in ECL^{i}PS^e including the main solver libraries, search strategies, and optimization. The book also includes a short introduction and overview, exercises for each chapter, and sample solutions for some exercises.

The first chapter of the book introduces logic programming and pure Prolog. It does this rather succinctly, and my feeling is that the book is probably not the right place to get an introduction to logic programming. A reader should probably start with Clocksin and Mellish if they have no familiarity with Prolog. Still this material has to be there for self-containedness. The chapter introduces how to use the ECL^{i}PS^e system, discusses unification, lists, and gives a couple of simple logic programs. One of the strengths of the book is the numerous example programs, with example queries and full output of the ECL^{i}PS^e system, and this is true right from Chapter 1.

The second chapter is rather puzzling. It introduces a small programming language $\mathscr{L}_0$ and gives this an operational semantics. It then shows how Prolog programs can be translated to $\mathscr{L}_0$, and how $\mathscr{L}_0$ can be translated (or read) as a logical formula. This seems repetitive without being very elucidating. The only real difference between the two views is that the logical reading $\mathscr{L}_0$ captures the *program completion* semantics of Clark, but the book really makes no use of this. Overall I am curious as to why the authors thought the chapter was worth including.

Chapter 3 introduces (Prolog style) arithmetic in ECL^{i}PS^e, that is, the evaluation and testing of arithmetic expressions. It begins with a succinct definition of arithmetic expressions and then introduces `is` and the arithmetic comparison predicates. It does a good job of explaining the difficulties using these features that arise

with instantiation. The chapter also introduces the differentiation between *active constraints* which can affect variables, which is currently the only term equality, and *passive constraints* or tests which cannot affect variables, but can cause failure. The chapter concludes with a discussion of operators and precedence and ECL^{i}PS^e `op` declarations. The example program introduced in this chapter will be cleverly reused later on when better arithmetic is available.

Chapter 4 discusses the control and meta-programming facilities of ECL^{i}PS^e, in particular the representation of programs as terms (the ECL^{i}PS^e equivalent to) `call`, if-then-else, disjunction, and negation, `once`, the cut, and `clause`. The ubiquitous `solve` meta interpreter is introduced, and extended to give derivation lengths, or to handle the arithmetic comparison predicates of the previous chapter (but strangely not `is`)! The modification in this chapter of the unification algorithm introduced in Chapter 1 to support functors with the same name and different arity seems strange. There would have been no harm in using the modified version in Chapter 1.

Chapter 5 introduces all the structure testing, comparison, and decomposition built-ins we are used to for Prolog, like `var`, `==`, and `functor`. The comparison family `@<` is missing, but not required in the rest of the book that will concentrate on constraints rather than term manipulation. As with the rest of the book these chapters are concise, clear, and well illustrated by examples.

In Chapter 6 we are introduced to CP. The basic concepts and a number of short examples which will be reexamined through the book are discussed. There is a discussion of modelling and modelling choices as well as a brief overview of local search, top-down search (the combination of propagation and branching), and branch-and-bound search. This is a good short summary of CP touching on a lot of issues briefly, and the examples are clearly explained.

Chapter 7 introduces the iteration and array constructs in Eclipse. As a hoary old Prolog programmer I find the iteration notation confusing, but it is certainly way more succinct than plain Prolog. I am sure with practice they would become clear to me. Something I found lacking was an understanding of when (multi) iterations terminate. For the usual iterators `foreach` and `count` it seems clear that it terminates when the first of these terminates, but this does not appear to be the case for `fromto`. While `fromto([], Tail, [Head|Tail], Reverse) do Head = a` terminates immediately with `Reverse = []` the `reverse` example

```
fromto([],Tail,[Head|Tail],Reverse),foreach(El,[a,b,c]) do Head=El
```

terminates when the `foreach` terminates with `Reverse - [c,b,a]`. The array syntax of ECL^{i}PS^e is impressive, giving natural array syntax, although like arithmetic, one has to be somewhat careful on ensuring the expressions are "evaluated."

Chapter 8 discusses top-down search with passive constraints, which only check fixed values. This is CP in Prolog, how we might have tackled constraint satisfaction problems (CSPs) in Prolog 20 years ago. While the discussion is clear, and the flexibility of credit based search is impressive (and still not appreciated enough in the CP community) the question is why have this chapter when three chapters later we will revisit search, with the much more powerful active constraints? The end of the chapter introduces non-logical (non-backtracked) variables and uses them

https://doi.org/10.1017/S1471068408003281 Published online by Cambridge University Press

to count backtracks. These "nasty" features are essential for some kinds of search control, and the backtrack counting example also illustrates some of the ambiguities in quantifying search that plague CP.

Chapter 9 introduces the `suspend` library, a kind of "poor man's" constraints, which can be posted at any time, but wait until all their variables are fixed before checking. Suspension is ECL^{i}PS^e's version of `freeze` from Prolog II. There is a nice modification of the `solve` meta-interpreter to explain delay, although hardly novel. The chapter goes on to define constraint satisfaction problems (CSPs) using the `suspend` library, illustrating the separation of specification of the problem from the search. While it may be nice to illustrate this separation before we introduce active constraints, I am puzzled why the authors bothered with the `suspend` library. Active constraints will do the same thing better, with no more complex coding. I can not see many users of ECL^{i}PS^e using `suspend`, at least for CSPs. General delay is useful for other things that are not investigated in this book.

Finally in Chapter 10 we reach active (finite domain) constraints, the core of a CLP system. The chapter begins with symbolic finite domain library `sd` before introducing the integer and real finite domain library `ic`. The symbolic domains solver is more or less unique to ECL^{i}PS^e so it would have been nice to see a longer example like the map colouring example using it. The description of the `ic` solver explains the difference between real and integer variables and revisits the examples from the previous chapter. Surprisingly, there is no description of the fixpoint behaviour of the propagation engine, or indeed the difference between bounds and domain propagation. Since Section 10.3.6 deals with how to prevent the waking up of constraints this seems a notable omission, although it does not affect usability of ECL^{i}PS^e except for advanced users.

Chapter 11 discusses search in the presence of abstract constraints. Programmable adaptive search is core to CP and one of the strengths of CLP over other CP paradigms, so this is an important chapter. The chapter begins with backtrack-free and shallow backtracking search. Once more I am a bit puzzled by the inclusion of this material, as they are not very useful. The already introduced credit based search and its various uses provides a much more practical example of incomplete search strategies. The chapter provides an excellent discussion of variable ordering heuristics and value ordering heuristics and the connections between them. It neatly utilizes the iteration facilities of ECL^{i}PS^e to define the heuristics. The chapter concludes with the introduction of the ECL^{i}PS^e generic search predicate.

Optimization is the topic of Chapter 12. After introducing the `minimize` optimization predicate, and discussing the knapsack problem, there is a nice discussion about the currency design problem, illustrating the use of redundant constraints, and using the answer of one problem to set up the next problem. The next example is solving and generating Sudoku puzzles, a beautiful illustration of the power of CLP and ECL^{i}PS^e in particular. I doubt any other language could manage Sudoku generation in 3 (small) pages of code (even other CLP languages). Next the chapter explores the more sophisticated optimization predicate `bb_min` which allows the user to weaken the optimization by accepting answers that are close but not necessarily optimal, and to control the kind of optimization search: continuing from the current

https://doi.org/10.1017/S1471068408003281 Published online by Cambridge University Press

position in the search, restarting the search from scratch, or using dichotomic search. These somewhat subtle features are well explained.

Chapter 13 discusses constraint programming over reals using interval constraint solving. It nicely characterizes the kinds of problems including reals that can be tackled and illustrates each kind. Interval constraint solving over reals has a number of subtleties which can trip up naive users and these are well explored. The effect of the propagation threshold which controls the accuracy of interval propagation is well illustrated. Search with interval solving using the `locate` search predicate, and shaving using the `squash` predicate, and in particular optimization over real interval solving and its dangers is discussed in detail.

Chapter 14 considers the linear programming facilities of ECL^{i}PS^e available through its `eplex` library. The `eplex` library is quite different to other constraint libraries, requiring explicit initialization and solving calls, and returning its solutions in a different way. The chapter exposes some subtleties in ECL^{i}PS^e where the expression W :: [0..10] which would declare W as an integer variable in the `ic` solver only has the effect of declaring W to have integer bounds in the `eplex` solver, and hence can succeed with a value 2.5, with a warning about integrality not being enforced. Presumably there are reasons for this design choice, and it is certainly better to expose it than let a bewildered user find it for themselves, but it seems odd to me. The chapter continues with a delightful example of solving a non-linear optimization problem using repeated addition of linear constraints. This is good illustration of the power of the combination of linear constraint solving and backtracking search. Next the chapter explains the use of the linear solver as a constraint propagator where triggers are used to control when the propagator is invoked, and the non-linear optimization problem is revisited. The chapter concludes with increasingly more complex versions of facility location problems, culminating in a non-linear version which illustrates the power and conciseness of ECL^{i}PS^e very well.

The principle strengths of the book are the clarity of the writing and the use of well chosen examples. The programs are very well written, with efficiency considerations often subtly included. After digesting the book, and trying out examples and exercises in ECL^{i}PS^e, a reader will come away with solid grasp of CP, and the flexibility of ECL^{i}PS^e.

I found some of the choices of material included in the book and the arrangement of the material odd. I realize every arrangement of material in any book fails to satisfy some of the desires of its authors, but obviously my "objective function" is quite different from the other authors. I would have loved to see another chapter where hybrid solutions using both the `eplex` and `ic` libraries were explored, since this is one of the strongest capabilities of ECL^{i}PS^e. I was a bit disappointed that the presentation of the Prolog part of ECL^{i}PS^e was so standard. Prolog can be presented as CLP over Herbrand terms without introducing unification as anything more than a constraint solver. Admittedly ECL^{i}PS^e users may well be better off with the traditional view when interacting with arithmetic, structure inspection, and other "non-logical" features of the language.

https://doi.org/10.1017/S1471068408003281 Published online by Cambridge University Press

In summary, this book is a valuable addition to the canon of CP texts. It provides a well-exampled introduction to ECL^{i}PS^e and a good basis for an advanced course on CP using ECL^{i}PS^e as its CP language.

Peter J. Stuckey
University of Melbourne, Australia
(e-mail: pjs@cs.mu.oz.au)

https://doi.org/10.1017/S1471068408003281 Published online by Cambridge University Press