Chapter 3

Constraint Programming
as Declarative Algorithmics

It is a great honor to receive the second ACP award for research excellence in constraint programming. It is also a humbling experience as I reflect how lucky I have been to meet and work with so many talented people at many stages in my career. As William E. Woodward said, “in the queer mess of human destiny the determining factor is luck. For every important place life, there are many men of fairly equal capacities. Among them, luck divides who shall accomplish the great work, who shall be crowned with laurel, and who should fall back in obscurity and silence”. Jean-Charles Régin expressed it a bit more bluntly when he once told me: “everything you did would have been contributed by someone else anyway.” It is obvious that I have been fortunate to be at the right place at the right time many times.

I became interested in research here in Brittany. The university of Namur in Belgium had the great practice of sending students in their final year abroad for about 4 months. I went to the center for telecommunications (CNET) in Lannion for studying software engineering. After two months, my fellow student and I had nothing left to work on, so we started looking for other opportunities in the center. I was told that there were two great researchers I should consider, one in concurrent programming and one in logic programming. Concurrent programming sounded more interesting so I went knocking at the door of this fellow several days in a row without success. I gave up and met Mehmet Dincbas the next day. Mehmet gave me papers and books to read, suggested all kinds of programming projects in logic programming, answered all my questions (and there were many), and spent considerable time explaining me the grand

Chapter written by Pascal Van Hentenryck.
challenges and the folklore of logic programming. Before I was about to leave, he told me that he was joining Hervé Gallaire at the European Computer-Industry Research Centre (ECRC) and he asked me whether I was interested to come along. I became the first PhD student at ECRC in August 1985 where I spent about 4.5 years in the CHIP project. I then joined Brown University in 1990 where I was recruited by Paris Kanellakis. I also had the opportunity to spend time at the university of Marseilles, at MIT, and at the university of Louvain, and to collaborate with Ilog.

In this talk, I would like to reflect on some of the lessons I learned in the last 20 years and to illustrate them on several research projects. These lessons are

1) Find the missing link;
2) Build systems;
3) Be driven by applications;
4) Just do it; then clean the mess;
5) Strive for the declarative ideal;
6) Find great people to work with.

I do not claim that these lessons are universal. I found them useful for the kind of research I engage in. I also did not invent them. I often learned them the hard way, but I keep coming back to them whenever I contract the research blues. Finally, they are heavily influenced by the research vision promoted by Hervé Gallaire at ECRC.

3.1. The CHIP Project

Find the missing link: When I came to ECRC, I was given Alain Colmerauer’s slides on PROLOG III and Jean-Louis Lauriere’s paper on ALICE. Alain’s slides had the slogan

The Power of PROLOG III = Constraint Solving + Nondeterminism

while the ALICE system showed how to use constraints to solve a variety of combinatorial optimization problems. Alain and Jean-Louis lived in Marseilles and Paris respectively and could not be more different. My role at ECRC was to bridge their work. I read a lot of papers in the fall of 1985 but could not see the connection. During my Christmas vacation in Belgium, I sneaked out one morning and went the library at the university of Namur to look for the 1977 paper by Alan Mackworth in the AI journal. I read it there sitting on the floor. This is a beautifully written paper with an amazing footnote and I realized then that I had found the missing link: arc consistency was declarative and could unify the goals of ALICE and PROLOG III: solving complex problems in a declarative language. This is obvious a posteriori but, at that time, arc consistency was exotic: there were very few papers on this topic in 1985 and AC-4 was not even published yet. I wrote my first constraint paper “Domains in Logic
Programming” which was published at AAAI’86. I gave the talk in the last session of the conference in a remote building and there were 2 persons in the audience besides the speakers and the session chair. But I had the chance to meet Alan who could not have been more encouraging.

Given the research focus in the CP conferences in recent years, it is interesting to mention why it was an interesting issue to unify ALICE and Prtolog III. ALICE was a back-box solver which limited its applicability. Even Jean-Louis Lauriere was not actively working on ALICE, but was focusing on his ruled-based system SNARK. The contribution of the CHIP system could be captured by the slogan

The Power of CHIP = Declarative Filtering + Nondeterminism.

CHIP allowed us to solve much more complex applications than ALICE by opening the search and the language. I will come back to this issue later in the paper.

Build systems: After finishing the AAAI-86 paper, Hervé and Mehmet told me to implement the ideas. I really did not see the point: I knew that they would work and I wanted to move to another topic and write more papers. But research, Hervé told me, is not about writing papers: It is about creating and communicating new ideas. System building plays a fundamental role in this process. Systems complement papers in communicating ideas: they often spread the ideas much faster, increase awareness in other communities, and help technology transfer. Systems also drive the research as they tell us what really works in practice and often suggests new algorithmic and complexity issues. Hervé, Alain Colmerauer, and Jean-Louis Lassez constantly reminded me of the value of system building.

Be driven by applications: In preparing this talk, I wanted to stress the importance of applications for our community. But I was afraid to lose 80% of my audience after seven slides, so I decided to research this topic a bit more. Ron Rivest’s Turing award slides gave me the courage to approach this issue as he wrote that his research in cryptography tries “to solve practical applications using computer science theory”. In 1986, Helmut Simonis joined ECRC and became my office mate, showing Mehmet’s talent in recruiting great people. Helmut has no fear: He would try CHIP on every possible application and this often boosted the development of the system tremendously. In fact, ECRC itself was a great place for developing a system as industrials would come and give us applications to solve.
Jean-Philippe Carillon visited ECRC in 1987 and demonstrated a package for warehouse location. The first model I wrote in CHIP for this problem looked like this (using mathematical notations, not CHIP syntax):

\[
\begin{align*}
\text{minimize} & \quad \sum_{w \in W} f_w y_w + \sum_{c, w \in W} t_{c, w} x_{c, w} \\
\text{subject to} & \quad \forall w \in W : y_w = 0 \rightarrow \forall c \in C : x_{c, w} = 0 \\
& \quad \forall c \in C : \sum_{w \in W} x_{c, w} = 1 \\
& \quad \forall w \in W : y_w \in \{0, 1\} \\
& \quad \forall c \in C, w \in W : x_{c, w} \in \{0, 1\}
\end{align*}
\]

It was basically a traditional MIP model, exploiting some of the CHIP coroutining facilities to link the warehouse variables \(y_w\) and the decision variables \(x_{c, w}\) assigning the customers to the warehouses. I showed the model to Hervé and you could see he was disappointed by the model and the performance, although he was too diplomatic to say so directly. Hervé always had very high standards, which included coming to ECRC before, and leaving after, anyone else. I then wrote another model which was much more efficient. Using mathematical notations, this model could be written as

\[
\begin{align*}
\text{minimize} & \quad \sum_{w \in W} f_w y_w + \sum_{c, w \in W} t_{c, w} x_{c, w} \\
\text{subject to} & \quad \forall w \in W : y_w = 0 \rightarrow \forall c \in C : x_c \neq 0 \\
& \quad \forall w \in W : y_w \in \{0, 1\} \\
& \quad \forall c \in C : x_c \in W
\end{align*}
\]

The key is the expression \(t_{c, x_c}\) which features the element constraint, avoiding the 0/1 decision variables \(x_{c, w}\). The warehouse location opened a pandora box for us: constraints did not have to be numeric: They could capture any relation and hence preserve the high-level structure of the applications. I gave the talk on the resulting paper at AAAI’88 and Ken McAloon was in the audience (there were more people this time around). He was amazingly encouraging and excited about the benefits of capturing the structure of applications.

After that, we saw element constraints everywhere. Helmut started solving the car-sequencing problem and the early part of the model looked like (using OPL syntax)

1. solve 
2.   forall(o in Options & s in Slots) 
3.     setup[o,s] = option[o,slot[s]]; 
4.   forall(o in Options & s in 1..nbSlots-cap[o].u+1) 
5.     sum(j in s..s+cap[o].u-1) setup[o,j] <= cap[o].1; 
6.   ... 
7. }
Lines 2–3 feature the `element` constraint to link the option and slot variables, while lines 4–5 specify the capacity constraints on the options. However the issue was how to express the demand constraints on the cars. Helmut obviously did not want to introduce 0/1 variables and he asked me if I had another idea (recall that we were separated only by two (bulky) monitors). The pandora box was open and I told him that it was easy to introduce this cardinality constraint in CHIP. Once again, we had captured another important substructure, simply by considering an interesting application.

The resulting program would quickly solve small instances but would backtrack forever on large ones. Helmut then built a visualization of the program and we spent time looking at the screen and understanding what was going on. We eventually realized that the propagation was only detecting failures very late. We designed redundant constraints (which are sometimes called implied constraints now) to address this pathological behaviour and obtained the program described in our ECAI’88 paper which looks like this:

```
1. solve {
2.   forall(o in Options & s in Slots)
3.     setup[o,s] = option[o,slot[s]];
4.   forall(o in Options & s in 1..nbSlots-cap[o].u+1)
5.     sum(j in s..s+cap[o].u-1) setup[o,j] <= cap[o].l;
6.     atmost(slot,demand);
7.   forall(o in Options & i in 1..optionDemand[o])
8.     sum(s in 1..nbSlots-i*cap[o].u) setup[o,s] >= optionDemand[o] - i*cap[o].l;
10. }
```

Line 6 contains the cardinality constraint, while lines 7–9 describe the redundant constraints. These recognize that only so many cars requiring an option can be produced in a given time window and ensure that the remaining cars are produced before.

The car-sequencing application was an amazing learning experience for us. It led to cardinality constraints, to visualizations, and to redundant constraints. The modeling itself removes symmetries (cars requiring the same options are interchangeable) and was the motivation for Freuder’s paper in 1992 on interchangeability. Jean-François Puget also told me that this paper attracted him to the area.

Other applications had similar effects. For instance, the cutting-stock applications presented at ICLP’88 used the `element` constraint, lexicographic ordering on the variables, and domain splitting. The microcode labelling application, published in the journal of logic programming (JLP) in 1990, introduced what is now called `table constraints`. The scheduling application, also in the JLP paper, taught us the value of flexible search procedures.
Just do it, then clean the mess: Alan Perlis once wrote that “simplicity does not precede complexity, but follows it”. It is indeed difficult to isolate the right concepts or abstractions immediately or to come up with the right design the first time. It is just how science works. We have to see the same patterns several times before recognizing them and determining how important they are. CHIP was sometimes criticized because it was growing in an ad-hoc fashion. But we could not see the proper abstractions or did not have the knowledge to simplify the concepts at the time. When I joined Brown in 1990, my goal was to clean the mess and I started working on the cc(FD) system. Ole Madsen, speaking about the BETA programming language said that “there were always two criteria for adding a construct to the language: it should be meaningful from a modeling standpoint as well as from technical standpoint.” Together with Yves Deville and Vijay Saraswat, I took the modeling road and try to find generic abstractions to build all the ad-hoc constraints of CHIP. It led to the cardinality operator, constructive disjunction, reification, and indexicals. Many of the ad-hoc constraints of CHIP were now easy to build in the language using logical or cardinality combinators. Nicolas Beldiceanu and Jean-Charles Régin took a more algorithmic road and studied how to increase the algorithmic power of constraints. Jean-Charles’ papers were especially elegant because they removed the ad-hoc aspects of algorithmically sophisticated constraints: Indeed, Jean-Charles showed how to enforce arc consistency on the alldifferent and global cardinality constraints and, as I said before, arc consistency is a declarative and natural concept. In preparing this talk, I read Jean-Charles’s paper on alldifferent again: It contains the observation that “only limited works have been carried out on the semantics of constraints”. It is stunning to see how such a fundamental idea was adopted so slowly, even about 10 years after the start of the CHIP project.

Car sequencing provides another example of the slow nature of progress and the necessity of the “just do it; then clean the mess” approach. Car sequencing introduced a cardinality constraint in 1988, which led to the cardinality operator in 1991 and to the global sequence constraint by Régin and Puget in 1997. In 2006, van Hoeve, Pesant, Rousseau, and Sabharwal won the best paper award for their work on the sequence constraint more than 18 years after the ECAI’88 paper.

Strive for the declarative ideal: I have mentioned several times already the importance of declarative abstractions. The success of constraint programming comes from its declarative aspects allowing users to be in the realm of modeling, more than programming. The filtering algorithms in constraint programming are typically specified declaratively. They are not just a bag of implementation tricks, but rather they specify the properties enforced when the propagation step converges. This property should ideally be arc consistency, but it has to be relaxed for some constraints whose feasibility problems are NP-complete, such as the edge finder. There are significant benefits in having these declarative specifications. On the one hand, they define an algorithmic problem for which algorithmic and complexity results can be derived. On the other hand, they allow for compiler optimizations and model transformations, as studied by
Christian Schulte and Peter Stuckey for instance. Search in constraint programming is also declarative and can be viewed as problem decomposition. As a result, constraint programming offers a compositionality and a separation of concerns that is so important in practical applications which typically features many idiosyncratic constraints that are revealed over time during the acquisition process.

**Work with good people:** I was lucky at ECRC to meet many researchers who had a lasting impact on my career. I mentioned several of them already but there were many others, too many to list in fact. Pierre Dufresne changed my life when he told me to view debugging as a game, man against machine, in which to surrender is not an option. I learned from Mireille Ducassé that a scientific talk is really a show. I saw her give the first talk, the day after a banquet, in which she said about a debugger: “and you press next” and then (really loudly) “splash, you get everything in your face” (except that she did not use “face” but its slang equivalent.) I was stunned but everyone else seemed to love it. Abder Aggoun and Jacques Noyé helped me understand that implementations are scientific objects of their own. Thomas Graf convinced me that academics should spent some time in the US.

I would like to come back to Alain Comerauer who I really wanted to meet at that point. Thomas Graf suggested to obtain a result Alain deeply care about. So we started working on disequations in linear constraints. The main issue in this problem is to identify all variables forced to take a single value (i.e., fixed variables or hidden constants). Many great scientists were working on the topic: Alain, Jean-Louis Lassez and Ken McAloon, and Peter Stuckey. Thomas thought that we could obtain a purely syntactic form and we eventually did: the first non-zero coefficient of a constraint must be greater than zero. It remained to show that this syntactic form could be preserved by pivoting. We started reading this book by Garfinkel and Nemhauser and it contained the lexicographic pivoting rule which, in fact, maintained our standard form. We had the result and I had no idea that, about 10 years later, I would meet George Nemhauser and start working on stochastic optimization with him.

I met Alain in Jerusalem and he then invited me to Marseilles several times. Meeting Alain was always a stimulating experience. I remembered once explaining some implementation details on the syntactic form and he did not believe the results. I went over the results that night, proving everything again. I waited for him to arrive the next morning. He was smiling when he came and told me: “It is correct but you explained these results very badly”. Alain then used our syntactic form in Prolog IV, which gave me great pleasure. Visiting Marseilles was always rejuvenating: Alain and Michel van Caneghem always maintained some sense of excitement about science and always looked at papers in a positive light. Alain also gave me a fundamental criterion for reviewing papers when he told me “if I can keep the paper, I accept it”. In 1994, Eugene Freuder wrote this paper about search as decomposition. It was short and I could not really see the contribution at the time. But Gene was passionate about
the paper, so I kept it. And, in 2001, this paper helped Laurent Michel and I solve a problem about search strategies for optimization problems.

3.2. The Numerica Project

Find the missing link: In the fall of 1993, I started a junior sabbatical, which I spent partly at MIT and at Marseilles. My host at MIT was David McAllister and we spent the first couple of meetings trying to find a joint project. David had received a couple of papers by Eldon Hansen on the interval Newton method and I had recently read papers in BNR-Prolog by Older and Vellino. We decided to study how to integrate these ideas and, in particular, how to use interval Newton methods to filter nonlinear constraints. David thinks very fast and the discussions on the white board were exciting. We ended up finding a way to prune the variable bounds by using interval Newton methods: the pruning operator was itself a search algorithm. David and I then started a race to implement the ideas, David in Scheme and I in {\texttt{cc(FD)}}. We tested the algorithms on the Broyden banded function and the results were amazing: we could solve very large instances without search: propagation alone was sufficient. Eventually we came to realize that constraint programming and numerical analysis methods were really orthogonal: constraint programming techniques were effective to reduce the search space when far from a solution, while interval Newton methods were excellent close to a solution.

Build systems: Once again, we built systems for testing the idea: first the {\texttt{NEWTON}} system and then {\texttt{NUMERICA}}, a modeling language for global optimization. Here is the Broyden banded function in {\texttt{NUMERICA}}:

```
Input: int n : "Number of variables";
Range: idx = [1..n];
Set: J[i in idx] = {j in [max(1,i-5)..min(n,i+1)] | i <> j};
Variable: x : array[idx] in [-10e8..10e8];
Body: solve system all
[i in idx]:
  0 = x[i] * (2 + 5*x[i]^-2)+1 - Sum(k in J[i]) x[k]*(1+x[k]);
```

These systems were key in isolating the limitations of the algorithms, discovering the synergies between constraint programming and interval analysis, and conveying the results.

Be driven by applications: Deepak Kapur was also visiting David that fall and joined the discussions. He introduced us to homotopy methods, the other class of global methods for nonlinear equations. Some papers in that area contained many interesting benchmarks in chemical engineering, robotics, and economics to name only a few
areas. Jean-François Puget also gave me a very challenging problem in circuit design, which helped drive the algorithmic developments further. This problem was solved orders of magnitude more efficiently than before and was described in a joint paper in the journal of global optimization. The university of Aachen also provided some extremely challenging distillation problems. Overall, this was an exciting time involving a lot of scientific progress. John Hooker once said in a panel that global optimization was one of the areas in which constraint programming had the most impact.

**Just do it, then clean the mess:** I spent the second part of my sabbatical in Marseille. I explained our results to Frédéric Benhamou and he was not happy about the description of the techniques: we were not formalizing them clearly enough. We worked together on resolving this issue and came up with the concept of box consistency. Once again, it was a declarative specification of the filtering and it helped subsequent algorithmic developments. Later on Yves Deville and I also capture the best such algorithms could hope for, which also improves our understanding of these techniques and the clarity of the NUMERICA book. Once again, I do not think we would have been capable of isolating these concepts initially: we had to experiment, try different algorithms, and find good trade-offs. Subsequently, it became important to formalize these concepts declaratively to foster progress.

**Work with good people:** I also met Laurent Michel during that period. Laurent spent four months at Brown working on our GAIA abstract interpreter. He really loved research and, after his stay at Brown, applied to the “fonds national de la recherche scientifique”, the Belgian counterpart to the French CNRS. I got lucky: he was rejected and had to choose exile and settle for a PhD at Brown University. Laurent and I have been working together on many projects since then: NUMERICA, LOCALIZER, OPL, and COMET. Laurent has no fear in trying new technologies, which has helped each of these projects substantially.

**Epilogue:** I do not want to leave the NUMERICA project without mentioning some of the subsequent developments. Many of the lessons I learned were in fact applied by the constraint group at the university of Nantes: they explored new links with symbolic computations, built new systems such as REALPAVER, branched out to new application areas such as vision, and designed new algorithms and filtering concepts. Very much the same can also be said about the group in Nice. I also worked extensively with Micha Janssen and Yves Deville on ordinary differentiable equations. We obtained surprising results by using constraints to address the wrapping effect, one of the main issues in interval methods for ODEs. Micha and I also designed a precisely A(α)-stable one-leg multistep methods, improving the stability of existing numerical analysis methods. It is amazing to see how constraint programming enabled us to take a fresh look at an area that had been investigated extensively.
3.3. The OPL Project

**Find the missing link:** The NUMERICA also opened another pandora box: modeling languages. Once I saw the beauty of a modeling language, I could not resist in designing a modeling language for constraint programming. The OPL system was thus a side-effect of NUMERICA and it played a fundamental role in exposing constraint programming to the operations research community. As a modeling language, OPL was innovative in several respects. It features a very high-level language for expressing search procedures, promoting a declarative style based on `forall` and `tryall` statements. It also provided default search procedures for constraint programming using some simple analyses of the model. Finally, it was the first modeling language integrating constraint and integer programming.

**Build systems:** OPL was also a huge engineering effort because I had the opportunity to collaborate with Ilog on this project. Jean-François Puget argued early on for the inclusion of linear and integer programming, while Irv Lustig wanted OPL to subsume AMPL. It was a challenge to design a language under these conditions but I had learned to try to keep things simple from my interactions with Alain. Jean-François also wanted OPL to interface with C++ and we pioneered code generation of models with OPL as well. Finally, I had learned my lessons from Helmut and I wanted to have OPL to support visualizations automatically. Once again, this was a very exciting time with exceptional researchers. OPL also had interesting side-effects: Jain and Grossman wrote a wonderful paper on logical Benders decomposition using both IP and CP models, demonstrating the potential of this technique. John Hooker also wrote more advanced models in OPL later on. Systems are enabling technologies!

**Strive for the declarative ideal:** For many years, the slogan for constraint programming has been

\[
\text{The Power of Constraint Programming} = \text{Constraints} + \text{Search.}
\]

In recent years, there has been a significant push to offer black-box CP systems. As I mentioned earlier, the ALICE system by Lauriere was a black-box system. But CHIP was instrumental in solving much more complex applications than ALICE by exploiting symmetries and dominance, writing specialized search procedures, using dedicated constraints or what is now called table constraints. Opening the language and exposing the search significantly enlarged the class of applications we could tackle. It does not mean that developing black-box optimization systems is not an important endeavor; it certainly is. But we are in a position to improve on ALICE now because we have built and experimented with open systems, contributed fundamental advances in modeling, filtering, and search, and delivered intermediate layers that dramatically improve productivity. We may want to remember the adage “He who can do more can do less”. A rich, open modeling and search language supports the building of sophisticated black-box systems. Observe also that the declarative ideal plays a fundamental
role here. Model transformations, the discovery of symmetries, and the derivation of search procedures are greatly facilitated by the declarative nature of constraint programming.

3.4. The Comet Project

Find the missing link: I will conclude the system descriptions with the COMET system. Once again, COMET fills a gap in the repertoire of tools for combinatorial optimization. It demonstrates the feasibility of the slogan

Local Search = Model + Search

allowing local search algorithms to be specified at a very high level of abstraction.

Build systems: It took us a long time to get there. Laurent developed LOCALIZER as a first step as part of his thesis and our paper at CP’97 introduced the concept of invariants. But it took us another five years and the contributions by many other researchers to articulate the above vision. The first paper on COMET was published at OOPSLA in 2002 but, once again, it was a first step which helped us discover many other abstractions and concepts.

Be driven by applications: COMET benefited from the fact that numerous applications of local search existed. Nevertheless, we spent considerable time reproducing them, sometimes discovering interesting variations on existing algorithms such as the iterative flattening algorithm of Cesta, Oddi, and Smith.

Strive for the declarative ideal: One of the most amazing features of COMET is that it allows local search algorithms to be presented using high-level declarative models. The model of the progressive party problem in COMET

1. var{int} boat[Guests,Periods](m,Hosts);
2. ConstraintSystem S(m);
3. forall(g in Guests)
4. S.post(2*alldifferent(all(p in Periods) boat[g,p]));
5. forall(p in Periods)
6. S.post(2*knapsack(all(g in Guests) boat[g,p],crew,carp));
7. forall(i in Guests, j in Guests : j > i)
8. S.post(atmost(1,all(p in Periods)(boat[i,p] == boat[j,p]) <= 1);

is almost identical to its constraint programming counterpart. The search procedure (omitting the tabu-search management) is also rather elegant:
50  Future and trends of Constraint Programming

Figure 3.1. Visualizing the Progressive Party Problem.

1. while(S.violations() > 0)
2.   selectMax(g in Guests,p in Periods)(S.decrease(boat[g,p]))
3.   selectMin(h in Hosts)(S.getAssignDelta(boat[g,p],h))
4.   boat[g,p] := h;

Recently, when looking at the COMET visualization on this example (see Figure 3.1),
I realized that the moves of this search procedure did not address directly the knapsack
constraints: assigning a new boat to a party would never reduce the violations in some
configurations. Consider guest 25 assigned to the third boat in Figure 3.1. Assigning
this guest to any other boat will only increase the violations. However, simple swaps
would remedy this limitation as guest 25 can be swapped with guest 6 on the fifth boat.
The resulting search procedure, once again omitting the tabu-search management, is
depicted in Figure 3.2. It is rather elegant in my opinion: It specifies what the neigh-
bors are (lines 3–9), while the neighborhood selector in line 1 specify how to select
the neighbor. The fact that this search procedure, combining two kinds of moves, can
be expressed in 11 lines is quite satisfying.

**Just do it, then clean the mess:** COMET has been evolving significantly over the last
5 years. Many of the extensions that once seem ad-hoc have been replaced by novel
abstractions such as constraint-based combinators for local search and differentiable
invariants. They probably look obvious retrospectively (which we always take as a
1. MinNeighborSelector N();
2. while (S.violations() &gt; 0) {
3.    selectMax(g in Guests, p in Periods)(S.violations(boat[g,p]))
4.    selectMin(h in Hosts, d=S.getAssignDelta(boat[g,p],h))(d)
5.    neighbor(d,N)
6.    boat[g,p] := h;
7.    selectMin(g1 in Guests, d=S.getSwapDelta(boat[g,p],boat[g1,p]))(d)
8.    neighbor(d,N)
9.    boat[g,p] := boat[g1,p];
10.   if (N.hasMove()) call (N.getMove());
11.}

Figure 3.2. The Search for the Progressive Party Problem.

compliment). Before we introduced these abstractions, either we could not recognize them or we did not see how to implement them in a reasonable fashion. It is often amazing to realize how slow we are.

3.5. The Future of Constraint Programming

I will conclude this talk with some reflections about the future of constraint programming. Obviously, there are many other possible directions and this book should give you a flavor about where the field is going. This section simply tries to apply the lessons learned to the future. It is important first to acknowledge that CP has great opportunities. The nature of optimization problems induces significant challenges that will not disappear any time soon. Industry needs effective solutions to large-scale and complex problems. Moreover, new telecommunication technologies open considerable opportunities, as optimization is increasingly being applied in operational setting in which decisions are taken online under uncertainty. Finally, CP has not even started branching yet and has been confined to a small number of application areas.

Find the missing link: CP is an integration technology: It is good at embedding algorithms from many areas and has room to include more advanced algorithms from theoretical computer science, transparently leveraging this wealth of algorithmic knowledge. CP may play an increasingly significant role in reasoning about uncertainty: this is a huge area since most problems are inherently stochastic. Finally, there are intriguing connections between CP and simulation and between CP and machine learning. Exploiting some of these connections would provide new exciting developments in CP. I hope that the community will embrace them, although they will likely lead to publications quite different from your typical CP paper. Hopefully the community will remain open and outward-looking, encompassing new ideas and technologies and a broad view of constraint programming.
Be driven by applications: CP has narrowed in terms of applications in the last decade and has focused on traditional combinatorial optimization. But CP has a role to play in many other areas. Luc Jaulin demonstrated a beautiful application in robotics at CP’07 and I would have loved to have such applications during our work on ODEs. Graphics and vision have a tradition of using constraints but they have little or no visibility at the conferences and in the community. Yet I believe that CP may be in a position to contribute to these areas once again. And obviously online applications offers significant challenges in which insights from CP may bring new directions. It seems desirable to have a much more substantial benchmark library to drive the research and to showcase the technology.

Build systems: CP should continue developing increasingly more advanced systems. Providing sophisticated visualization, debugging, and explanation tools, will help users understand complex behaviors and interactions between constraints that are likely to arise in practice. Automatic tuning of CP algorithms would dramatically reduce development time and parallel implementations will boost performance at the time when the speed of processors is leveling off. Artificial intelligence had significant impact on programming languages in the past thanks to the complexity of the problems it tackled. I believe CP is in a similar position now. Once again, it is important to recognize the need for a proper forum for such research and CP-Tools was introduced with that goal in mind.

Just do it, then clean the mess: Alan Kay said that “simple things should be simple; complex things should be possible”, which is a great guide for the next generation of CP systems. Simple problems should have small models or simple instances should be solved automatically and efficiently. And complex things should be possible by providing open languages with multiple abstraction layers. The regular constraint and differentiable invariants are illustrations of such abstraction layers, but more are needed. The work of Nicolas Beldiceanu on understanding the nature of global constraints is important in that respect.

Strive for the declarative ideal: Finally, although my first paper at AAAI’06 was about finite domains, I believe that CP should move on and study constraints over more complex objects such as sets, graphs, trees, and sequences to name only a few. Obviously, the underlying algorithms should exploit these specific structures and should be more than syntactic sugar. As Ole Madsen said: “There were always two criteria for adding a construct to the language: it should be meaningful from a modeling standpoint as well as from technical standpoint”.

Acknowledgments I would like to thank the ACP award committee for this recognition and Frédéric Benhamou for inviting me to contribute this chapter. I also would like to thank all my research collaborators who are not acknowledged in the text and
all my undergraduate and graduate students. Last but not least, special thanks to Baudouin Le Charlier whose class on programming methodology, although taught with flow charts, showed me the beauty of the declarative ideal early in my studies.