Chapter 4: Basic Constraint Reasoning (SEND+MORE=MONEY)

Helmut Simonis

email: helmut.simonis@insight-centre.org
homepage: http://http://insight-centre.org/

Insight SFI Centre for Data Analytics
School of Computer Science and Information Technology
University College Cork
Ireland

CRT-AI CP Week 2024

Acknowledgments

This publication has emanated from research conducted with the financial support of Science Foundation Ireland under Grant number 12/RC/2289-P2 at Insight the SFI Research Centre for Data Analytics at UCC, which is co-funded under the European Regional Development Fund.

A version of this material was developed as part of the ECLiPSe ELearning course: https://eclipseclp.org/ELearning/index.html. Support from Cisco Systems and the Silicon Valley Community Foundation is gratefully acknowledged.

Example 1: SEND+MORE=MONEY

- Example of Finite Domain Constraint Problem
- Models and Programs
- Constraint Propagation and Search
- Some Basic Constraints: linear arithmetic, alldifferent, disequality
- A Built-in search
- Visualizers for variables, constraints and search

1 Problem

Let's start with the problem.

Figure 1: Problem Definition

$$\begin{array}{cccccc} & S & E & N & D \\ + & M & O & R & E \\ \hline M & O & N & E & Y \end{array}$$

SEND+MORE=MONEY is a crypt-arithmetic puzzle, where characters encode numbers and we have to deduce which character stands for which digit. It is one of the classical, early examples of constraint programming, showing how the constraint reasoning mimics the human reasoning when solving this type of problem. The arithmetic problem is usually shown in the form of a handwritten addition. The puzzle is defined by the following rules:

Rules

- Each character stands for a digit from 0 to 9.
- Numbers are built from digits in the usual, positional notation.
- Repeated occurrence of the same character denote the same digit.
- Different characters denote different digits.
- Numbers do not start with a zero.
- The equation must hold.

2 Program

If we want to model this problem as a finite domain constraint problem, we have to define variables and constraints. *Variables* range over *domains*, which describe the possible values they can take. In this section we consider only finite domains, indeed only finite subsets of natural numbers. We will consider constraint problems with non-integral domains in a later chapter.

An *assignment* is a mapping from the variables to values in their respective domains.

A *constraint* ranges over one or multiple variables and expresses a condition which must hold between these variables. Constraints can take many forms, they may be explicit, describing combinations of values that are allowed or excluded, or symbolic, for example in the form of arithmetic constraints. Formally, we can see a constraint as a subset of the Cartesian product of the domains of its variables. An assignment *satisfies* a constraint if its projection on the variables of the constraint belong to this constrained subset.

An assignment of values to variables is a *solution* if all constraints are satisfied.

For many problems, we have considerable choices in selecting the variables and constraints. Often, models differ dramatically in the number of variables needed, the complexity and number of constraints used to describe the problem, and the speed by which a constraint solver can find a solution.

For the small puzzle considered here, the choice of the variables is fairly obvious: Each character is a variable, which ranges over the values 0 to 9.

For the constraints, a natural description would be:

- An *alldifferent* constraint between all variables, which states that two different variables must have different values. This is a very common constraint, which we will encounter in many other problems later on.
- Two *disequality constraints* (variable X must be different from value V) stating that the variables at the beginning of a number can not take the value 0.
- An arithmetic *equality constraint* linking all variables with the proper coefficients and stating that the equation must hold.

Remember that this is not the only model of this problem, we will discuss some alternatives later on.

SEND+MORE=MONEY Models

- ECLiPSe Show
- MiniZinc Show
- NumberJack Show
- CPMpy Show
- Choco-solver Show

ECLiPSe Model

```
:- lib(ic).

sendmore(Digits) :-
    Digits = [S,E,N,D,M,O,R,Y],
    Digits :: [0..9],
    alldifferent(Digits),
    S #\= 0,
    M #\= 0,
                1000*S + 100*E + 10*N + D
              + 1000*M + 100*O + 10*R + E
    #= 10000*M + 1000*O + 100*N + 10*E + Y,
    labeling(Digits).
```

Continue

MiniZinc Model

```
include "alldifferent.mzn";
var 0..9: S;
var 0..9: E;
var 0..9: N;
var 0..9: D;
var 0..9: M;
var 0..9: O;
var 0..9: R;
var 0..9: Y;
constraint S != 0;
constraint M != 0;
constraint           1000 * S + 100 * E + 10 * N + D
                   + 1000 * M + 100 * O + 10 * R + E
       = 10000 * M + 1000 * O + 100 * N + 10 * E + Y;
constraint alldifferent([S,E,N,D,M,O,R,Y]);

solve satisfy;
```

Continue

NumberJack Model (from https://github.com/eomahony/Numberjack/)

```
from Numberjack import *

def get_model():
    model = Model()
    s, m = VarArray(2, 1, 9)
    e, n, d, o, r, y = VarArray(6, 0, 9)
    model.add(      s*1000 + e*100 + n*10 + d +
                    m*1000 + o*100 + r*10 + e ==
          m*10000 + o*1000 + n*100 + e*10 + y)
    model.add(AllDiff((s, e, n, d, m, o, r, y)))
    return s, e, n, d, m, o, r, y, model

def solve(param):
    s, e, n, d, m, o, r, y, model = get_model()
    solver = model.load(param['solver'])
    solver.setVerbosity(param['verbose'])
    solver.solve()
```

Continue

CPMpy Model (from https://github.com/CPMpy/)

```
from cpmpy import *
import numpy as np

s,e,n,d,m,o,r,y = intvar(0,9, shape=8)
model = Model(
    AllDifferent([s,e,n,d,m,o,r,y]),
    (    sum(   [s,e,n,d] * np.array([       1000, 100, 10, 1]) ) \
       + sum(   [m,o,r,e] * np.array([       1000, 100, 10, 1]) ) \
      == sum( [m,o,n,e,y] * np.array([10000, 1000, 100, 10, 1]) ) ),
    s > 0,
    m > 0,
)

model.solve()
```

Continue

Choco-solver Model (from https://choco-solver.org/)

```
Model model = new Model("SEND+MORE=MONEY");
IntVar S = model.intVar("S", 1, 9, false);
IntVar E = model.intVar("E", 0, 9, false);
IntVar N = model.intVar("N", 0, 9, false);
IntVar D = model.intVar("D", 0, 9, false);
IntVar M = model.intVar("M", 1, 9, false);
IntVar O = model.intVar("0", 0, 9, false);
IntVar R = model.intVar("R", 0, 9, false);
IntVar Y = model.intVar("Y", 0, 9, false);

model.allDifferent(new IntVar[]{S, E, N, D, M, O, R, Y}).post();

IntVar[] ALL = new IntVar[]{
    S, E, N, D,
    M, O, R, E,
    M, O, N, E, Y};
int[] COEFFS = new int[]{
    1000, 100, 10, 1,
    1000, 100, 10, 1,
    -10000, -1000, -100, -10, -1};
model.scalar(ALL, COEFFS, "=", 0).post();

Solver solver = model.getSolver();
solver.showStatistics();
solver.showSolutions();
solver.findSolution();
```

Continue

Line **??** shows the module declaration. Each file should be a module, and there should be one module per file. All predicates defined in a module are only visible inside the module, unless they are exported.

Line **??** shows an export directive. It states that the predicate `sendmory` with *arity* one (with one argument) should be exported from this module.

Line **??** contains a library directive. We are loading the `ic` library, which defines the *I*nterval *C*onstraints in the ECLiPSe language.

Lines **??**-**??** define the `sendmory` predicate which is the core of our model. It consists of a single clause with the clause head `sendmory(L)`. The clause has a single argument L, which is a list of variables as defined in line **??**. The square brackets denote lists in Prolog.

We next define the domain of the variables in line **??**. The infix operator `::` takes a variable or list of variables as the first (left) argument, and a domain (here `0..9`) as the second (right) argument.

We then express the `alldifferent` constraint in line **??**. This constraint is a built-in constraint in the `ic` library, and is thus shown in bold.

Lines **??**-**??** express the two disequality constraints. The `ic` library defines the operator `#\=` for this purpose.

The big linear equality constraint is expressed in line **??**. The operator `#=` is used in the `ic` library.

Finally, we call the builtin search routine `labeling` to assign values to the variables.

A Note on Syntax

- Some formulations may seem simpler than others
- This largely is an artifact of a very simple problem
- In most models, you do not write down constraints one by one
- You create constraints based on data
- Ease of integration becomes more important than syntax
- Debugging tools for those who need a debugger :-)

Choice of Model

- This is *one* model, not *the* model of the problem
- Many possible alternatives
- Choice often depends on your constraint system
  - Constraints available
  - Reasoning attached to constraints
- Not always clear which is the *best* model
- Often: Not clear what is the *problem*

The model we have presented here is *one* model of the problem, it is not *the* model of the puzzle. Quite often there is a natural way of expressing the problem, but typically there are still many possible alternatives for expressing particular aspects of a model.

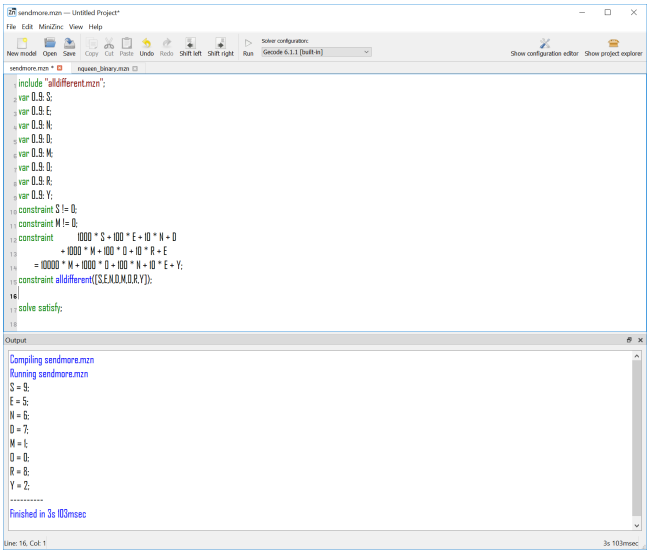
The choice which model to use often depends on your constraint system. A system may or may not contain particular constraints, some constraint system have constraint that others don't have and which are difficult to implement in another system. Some constraint systems have special reasoning attached to some constraints, so that they are much more powerful solving some problems than other systems.

Unfortunately, it is not always clear which is the *best model*, or if there is indeed a best model for a given problem class. Quite often, we don't even know what exactly the problem is, and we can decide which constraints we want to include and which aspects of a problem we can ignore.

One important aspect of constraint programming is that we want to play with the model, experiment by adding or removing some constraints, to really find out which problem we are going to solve and how we are going to do it.

Indeed, for the puzzle here we have developed two alternative models, which are described at the end of this chapter. There are hyperlinks on the slides to look at these alternatives, or you can use the table of contents of the video to look at them now.

Running the Program (MiniZinc IDE)



If we want to run the program, we have to enter a query `sendmory:sendmory(L).` which calls the predicate `sendmory` in the module `sendmory` with a single argument, a free variable L.

The system then returns the answer as a binding for the variable L, and it says `yes`, indicating that it has found a first solution. There may be additional solutions, which can be obtained by backtracking.

Question

- But how did the program come up with this solution?
- We show solution with ECLiPse, other solvers vary slightly

How did the program actually find this solution, what happened in the constraint system before it printed out this message? That is the question we are going to answer in the following sections.

Usually, we are not going to analyze the behaviour of a program in such great detail, but for an introduction I think it is useful to understand how much work can happen inside the constraint engine and how variables, constraints and search interact to find a solution.

3 Constraint Setup

We will now go through the constraint setup, where the constraints are created and perform their initial propagation before the search is started. We are going to use different visualization tools to understand what is happening, this gives a much better conceptual model than tracing through the execution with the line debugger.

3.1 Domain Definition

Domain Definition

```
var 0..9: S;
var 0..9: E;
var 0..9: N;
var 0..9: D;
var 0..9: M;
var 0..9: O;
var 0..9: R;
var 0..9: Y;
```

Figure 2: Domain Visualization

	0	1	2	3	4	5	6	7	8	9
S										
E										
N										
D										
M										
O										
R										
Y										

When we first encounter our list of variables, we define them to be domain variables with a domain from 0 to 9.

We can represent the variables with a *domain visualizer* (Figure 2). It shows the state of the domain variables at this point. Each line shows one variable, each column one value. Initially, all values are allowed for all variables, this is indicated by white cells.

3.2 Alldifferent Constraint

Alldifferent Constraint

```
include "alldifferent.mzn";

constraint alldifferent([S,E,N,D,M,O,R,Y]);
```

- Built-in alldifferent predicate included
- No initial propagation possible
- *Suspends*, waits until variables are changed
- When variable is fixed, remove value from domain of other variables
- *Forward checking*

3.3 Disequality Constraints

3.4 Equality Constraint

	1000*S+	100*E+	10*N+	D
	+1000*M+	100*O+	10*R+	E
10000*M+	1000*O+	100*N+	10*E+	Y

is transformed into

	1000*S+	91*E+		D
		+	10*R	
9000*M+	900*O+	90*N+		Y

Simplified Equation

$$1000 * S + 91 * E + 10 * R + D = 9000 * M + 900 * O + 90 * N + Y$$

We now look at the initial propagation of the equality constraint, while noting that the `alldifferent` constraint is suspended, but will be woken when variables are fixed to values.

Propagation

We consider the two sides of the equation, taking the domains of the variables into account.

$$1000 * S^{1..9} + 91 * E^{0..9} + 10 * R^{0..9} + D^{0..9} = \\ 9000 * M^{1..9} + 900 * O^{0..9} + 90 * N^{0..9} + Y^{0..9}$$

The left hand side ranges from 1000 to 9918, the right hand side from 9000 to 89919.

$$\underbrace{1000 * S^{1..9} + 91 * E^{0..9} + 10 * R^{0..9} + D^{0..9}}_{1000..9918} = \\ \underbrace{9000 * M^{1..9} + 900 * O^{0..9} + 90 * N^{0..9} + Y^{0..9}}_{9000..89919}$$

If the equation holds, then both sides must be equal. We can therefore consider the intersection of the ranges for left and right hand side.

$$\underbrace{1000 * S^{1..9} + 91 * E^{0..9} + 10 * R^{0..9} + D^{0..9}}_{9000..9918} = \\ \underbrace{9000 * M^{1..9} + 900 * O^{0..9} + 90 * N^{0..9} + Y^{0..9}}_{9000..9918}$$

From this we can deduce the following domain restrictions:

Deduction:

$$M = 1, S = 9, O \in \{0..1\}$$

Consider lower bound for S

- Lower bound of equation is 9000
- Rest of lhs (left hand side) $(91 * E^{0..9} + 10 * R^{0..9} + D^{0..9})$ is atmost 918
- S must be greater or equal to $\frac{9000-918}{1000} = 8.082$
  - otherwise lower bound of equation not reached by lhs
- S is integer, therefore $S \geq \lceil \frac{9000-918}{1000} \rceil = 9$
- S has upper bound of 9, so $S = 9$

Consider upper bound of M

- Upper bound of equation is 9918
- Rest of rhs (right hand side) $900 * O^{0..9} + 90 * N^{0..9} + Y^{0..9}$ is at least 0
- M must be smaller or equal to $\frac{9918-0}{9000} = 1.102$
- M must be integer, therefore $M \leq \lfloor \frac{9918-0}{9000} \rfloor = 1$
- M has lower bound of 1, so $M = 1$

Figure 3: Propagation of equality: Result

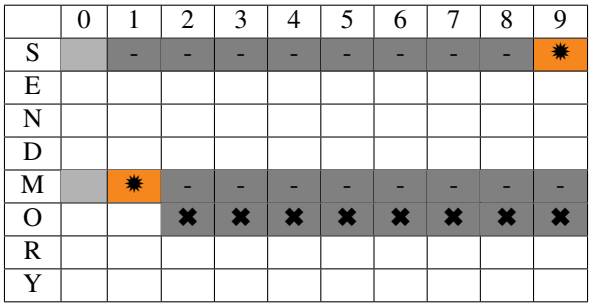
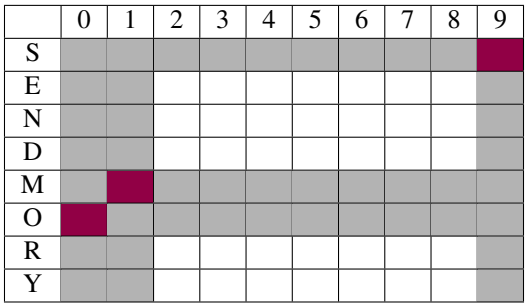


Figure 4: Propagation of `alldifferent`



Consider upper bound of O

- Upper bound of equation is 9918
- Rest of rhs (right hand side) $9000 * 1 + 90 * N^{0..9} + Y^{0..9}$ is at least 9000
- O must be smaller or equal to $\frac{9918-9000}{900} = 1.02$
- O must be integer, therefore $O \leq \lfloor \frac{9918-9000}{900} \rfloor = 1$
- O has lower bound of 0, so $O \in \{0..1\}$

So after the initial reasoning of the equality constraint (Figure 3), we have deduced that two variables (S and M) must have fixed values and that another variable O can only range between 0 and 1.

The assignment of the variables will wake up the `alldifferent` constraint. This removes the assigned values 1 and 9 from the domain of the unassigned variables. But variable O could only take values 0 or 1. If we remove value 1, then it must take value 0. This assignment will be propagated through the `alldifferent` constraint and removes the value 0 from the remaining variables (Figure 4.

At this point the equality constraint will be checked again, since some variable bounds have been updated.

For ease of presentation, we first remove the constant parts of the constraint, and obtain a simplified form:

Removal of constants

$$1000 * 9 + 91 * E^{2..8} + 10 * R^{2..8} + D^{2..8} = 9000 * 1 + 900 * 0 + 90 * N^{2..8} + Y^{2..8}$$

$$\mathbf{1000} * \mathbf{9} + 91 * E^{2..8} + 10 * R^{2..8} + D^{2..8} = \mathbf{9000} * \mathbf{1} + \mathbf{900} * \mathbf{0} + 90 * N^{2..8} + Y^{2..8}$$

$$91 * E^{2..8} + 10 * R^{2..8} + D^{2..8} = 90 * N^{2..8} + Y^{2..8}$$

We now go though several steps of updating variable bounds, each triggering a renewed check of the equality constraint.

Propagation of equality (Iteration 1)

$$\underbrace{91 * E^{2..8} + 10 * R^{2..8} + D^{2..8}}_{204..816} = \underbrace{90 * N^{2..8} + Y^{2..8}}_{182..728}$$

$$\underbrace{91 * E^{2..8} + 10 * R^{2..8} + D^{2..8} = 90 * N^{2..8} + Y^{2..8}}_{204..728}$$

$$N \geq 3 = \lceil \frac{204 - 8}{90} \rceil, E \leq 7 = \lfloor \frac{728 - 22}{91} \rfloor$$

Propagation of equality (Iteration 2)

$$91 * E^{2..7} + 10 * R^{2..8} + D^{2..8} = 90 * N^{3..8} + Y^{2..8}$$

$$\underbrace{91 * E^{2..7} + 10 * R^{2..8} + D^{2..8}}_{204..725} = \underbrace{90 * N^{3..8} + Y^{2..8}}_{272..728}$$

$$\underbrace{91 * E^{2..7} + 10 * R^{2..8} + D^{2..8} = 90 * N^{3..8} + Y^{2..8}}_{272..725}$$

$$E \geq 3 = \lceil \frac{272 - 88}{91} \rceil$$

Propagation of equality (Iteration 3)

$$91 * E^{3..7} + 10 * R^{2..8} + D^{2..8} = 90 * N^{3..8} + Y^{2..8}$$

$$\underbrace{91 * E^{3..7} + 10 * R^{2..8} + D^{2..8}}_{295..725} = \underbrace{90 * N^{3..8} + Y^{2..8}}_{272..728}$$

$$\underbrace{91 * E^{3..7} + 10 * R^{2..8} + D^{2..8} = 90 * N^{3..8} + Y^{2..8}}_{295..725}$$

$$N \geq 4 = \lceil \frac{295 - 8}{90} \rceil$$

Propagation of equality (Iteration 4)

$$91 * E^{3..7} + 10 * R^{2..8} + D^{2..8} = 90 * N^{4..8} + Y^{2..8}$$

$$\underbrace{91 * E^{3..7} + 10 * R^{2..8} + D^{2..8}}_{295..725} = \underbrace{90 * N^{4..8} + Y^{2..8}}_{362..728}$$

$$\underbrace{91 * E^{3..7} + 10 * R^{2..8} + D^{2..8} = 90 * N^{4..8} + Y^{2..8}}_{362..725}$$

$$E \geq 4 = \lceil \frac{362 - 88}{91} \rceil$$

Propagation of equality (Iteration 5)

$$91 * E^{4..7} + 10 * R^{2..8} + D^{2..8} = 90 * N^{4..8} + Y^{2..8}$$

$$\underbrace{91 * E^{4..7} + 10 * R^{2..8} + D^{2..8}}_{386..725} = \underbrace{90 * N^{4..8} + Y^{2..8}}_{362..728}$$

$$\underbrace{91 * E^{4..7} + 10 * R^{2..8} + D^{2..8} = 90 * N^{4..8} + Y^{2..8}}_{386..725}$$

$$N \geq 5 = \lceil \frac{386 - 8}{90} \rceil$$

Propagation of equality (Iteration 6)

$$91 * E^{4..7} + 10 * R^{2..8} + D^{2..8} = 90 * N^{5..8} + Y^{2..8}$$

$$\underbrace{91 * E^{4..7} + 10 * R^{2..8} + D^{2..8}}_{386..725} = \underbrace{90 * N^{5..8} + Y^{2..8}}_{452..728}$$

$$\underbrace{91 * E^{4..7} + 10 * R^{2..8} + D^{2..8} = 90 * N^{5..8} + Y^{2..8}}_{452..725}$$

$$N \geq 5 = \lceil \frac{452 - 8}{90} \rceil, E \geq 4 = \lceil \frac{452 - 88}{91} \rceil$$

No further propagation at this point We have reached a fixed-point of the constraint propagation, all constraints have been checked, and no further domain reduction is possible.

Figure 5: Domains after setup

	0	1	2	3	4	5	6	7	8	9
S										
E										
N										
D										
M										
O										
R										
Y										

Figure 5 shows the state of the variables after the constraint setup, three variables have been assigned, and the domains of the unassigned variables have been reduced.

4 Search

We now call the `labeling` routine to assign values to the unassigned variables.

The predicate call `labeling([S,E,N,D,M,O,R,Y])` in line **??** on page **??** will enumerate the variables in the given order, each time starting with the smallest value in the domain. If the assignment of that value fails, the next value will be tried, and so on. If no more values remain, the search will backtrack to the last previous choice and try its next alternative. This will continue until either

- a valid assignment is found for all variables, and the search *succeeds*
- no more untested choices remain, the search *fails*

4.1 Step 1

Figure 6: Search Tree Step 1

Figure 6 shows the search tree at this point. The top node shows the variable being assigned (here S), the label on the edge shows the value that is taken (here 9). The link is dotted, as the variable is already assigned, and no new choice is required.

4.2 Step 2

Figure 7 shows the search tree at this point. We are currently assigning variable E to value 4. As this is a proper choice, a solid link is used.

Assigning E to 4 (Figure 8) will wake the equality constraint

Propagation of $E = 4$, equality constraint

$$91 * 4 + 10 * R^{2..8} + D^{2..8} = 90 * N^{5..8} + Y^{2..8}$$

$$\underbrace{91 * 4 + 10 * R^{2..8} + D^{2..8}}_{386..452} = \underbrace{90 * N^{5..8} + Y^{2..8}}_{452..728}$$

$$\underbrace{91 * 4 + 10 * R^{2..8} + D^{2..8} = 90 * N^{5..8} + Y^{2..8}}_{452}$$

$$N = 5, Y = 2, R = 8, D = 8$$

The state after propagation the equality constraint is shown in Figure 9.

Figure 7: Step 2, Alternative $E = 4$

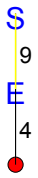


Figure 8: Assignment $E = 4$

	0	1	2	3	4	5	6	7	8	9
S										
E					✹	-	-	-		
N										
D										
M										
O										
R										
Y										

The assignments will trigger the `alldifferent` constraint, but the constraint detects that two variables (D and R) have the same value 8. This is not allowed and the `alldifferent` constraint fails (Figure 10).

We backtrack to the last choice (E), and then try the next value ($E = 5$). This situation is depicted in figure 11. The left child of node E is marked as a failure, we are currently testing value 5, the second alternative. The assignment of value ($E = 5$) will trigger both the `alldifferent` and the equality constraint. Assume that the `alldifferent` is woken first[1].

The propagation of the assignment $E = 5$ in the `alldifferent` constraint will remove value 5 from the variable N (Figure 13). At this point, since the lower bound of variable N is updated to 6, the equality constraint is triggered.

Propagation of equality

$$91 * 5 + 10 * R^{2..8} + D^{2..8} = 90 * N^{6..8} + Y^{2..8}$$

$$\underbrace{91 * 5 + 10 * R^{2..8} + D^{2..8}}_{477..543} = \underbrace{90 * N^{6..8} + Y^{2..8}}_{542..728}$$

$$\underbrace{91 * 5 + 10 * R^{2..8} + D^{2..8} = 90 * N^{6..8} + Y^{2..8}}_{542..543}$$

$$N = 6, Y \in \{2, 3\}, R = 8, D \in \{7..8\}$$

Figure 9: Result of equality propagation

	0	1	2	3	4	5	6	7	8	9
S										
E										
N						✹	-	-	-	
D			-	-	-	-	-	-	✹	
M										
O										
R			-	-	-	-	-	-	✹	
Y			✹	-	-	-	-	-	-	

Figure 10: Propagation of `alldifferent` fails!

	0	1	2	3	4	5	6	7	8	9
S										
E										
N						✹	-	-		
D			-	-	-	-	-	-	✹	
M										
O										
R			-	-	-	-	-	-	✹	
Y			✹	-	-	-	-	-		

The `alldifferent` constraint (figure 15) has fixed variable D to 7. We again wake the equality constraint.

Propagation of equality

$$91 * 5 + 10 * 8 + 7 = 90 * 6 + Y^{2..3}$$

$$\underbrace{91 * 5 + 10 * 8 + 7}_{542} = \underbrace{90 * 6 + Y^{2..3}}_{542..543}$$

$$\underbrace{91 * 5 + 10 * 8 + 7 = 90 * 6 + Y^{2..3}}_{542}$$

$$Y = 2$$

This finally assigns the last variable Y to 2 (Figure 16).

4.3 Further Steps

Search Tree with Gecode/GIST

[1] Which of the constraints is woken up first can be difficult to predict without understanding the details of the implementation of the solver

Figure 11: Step 2, Alternative $E = 5$

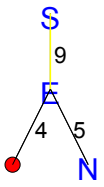
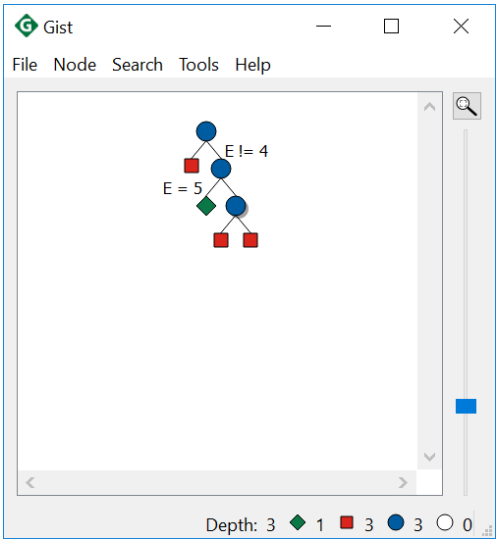


Figure 12: Assignment $E = 5$

	0	1	2	3	4	5	6	7	8	9
S										
E					-	✹	-	-		
N										
D										
M										
O										
R										
Y										



Some Differences

- Uses binary branching
  - var equal value, var not equal value

Figure 13: Propagation of `alldifferent`

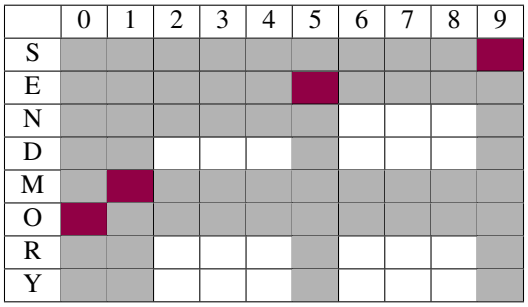
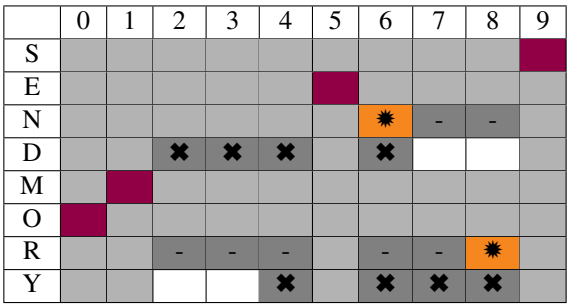


Figure 14: Result of equality propagation



- Solutions in green, failure leafs in red, internal nodes in blue
- By default, shows all failed sub trees collapsed
- By default, uses different search strategy

All variables have been assigned, the search succeeds. Figure 17 shows the resulting search tree. There is only a single variable (E) for which an actual choice had to be made, all other variables are assigned by constraint propagation. The last node at the bottom of the tree is a green success node, indicating that a solution was found.

4.4 Solution

We finally reached a solution (Figure 18), it is easy to check that this indeed solves the puzzle.

Figure 15: Propagation of `alldifferent`

	0	1	2	3	4	5	6	7	8	9
S										
E										
N										
D										
M										
O										
R										
Y										

Figure 16: Last propagation step

	0	1	2	3	4	5	6	7	8	9
S										
E										
N										
D										
M										
O										
R										
Y			✸	-						

Figure 17: Complete Search Tree

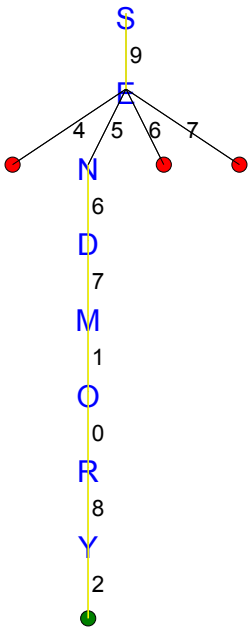


Figure 18: Solution

	9	5	6	7
+	1	0	8	5
1	0	6	5	2

5 Points to Remember

Points to Remember

- Constraint models are expressed by variables and constraints.
- Problems can have many different models, which can behave quite differently. Choosing the best model is an art.
- Constraints can take many different forms.
- Propagation deals with the interaction of variables and constraints.
- It removes some values that are inconsistent with a constraint from the domain of a variable.
- Constraints only communicate via shared variables.

Points to Remember

- Propagation usually is not sufficient, search may be required to find a solution.
- Propagation is data driven, and can be quite complex even for small examples.
- The default search uses chronological depth-first backtracking, systematically exploring the complete search space.
- The search choices and propagation are interleaved, after every choice some more propagation may further reduce the problem.

For Puzzle Purists Only

- We did not follow the puzzler ethos!
- We should solve the puzzle without making choices
- Even a case analysis should be avoided
- The puzzle has a single solution, we should be able to deduce the solution
- In the process shown, we are limited by the underlying assumptions
  - Treat each constraint on its own, they interact only by domains of variables
  - We only use the constraints that we stated in the model
- Can we do better than that?

Skip

Better Reasoning

- Three possible approaches (possibly many more)
  - Full domain reasoning for arithmetic, not just bound reasoning
  - Interaction of *sum* and *alldifferent* constraints
  - Deduced implied constraint

Looking at more than just bounds

- We only considered the smallest and largest values that can be achieved in the sum constraint
- We can do more
  - Can any of the values between be expressed as the sum of the terms
  - Consider holes in the domains, and in the range of possible values for LHS and RHS
- Usually not done in actual solvers for arithmetic constraints
- Easy to do with Dynamic Programming

Consider the interaction of multiple constraints

- Usually ignored, as only interaction is via domains of shared variables
- Here: Sum and *alldifferent* interact
  - When considering the bounds, we cannot assume that each variable takes its smallest/largest value independently
  - Find feasible assignment that minimizes/maximizes the total weight
  - To do this properly, we need some non-trivial reasoning
- Do we do this combined reasoning automatically, or only when prompted by the modeler?

Deduced Implied Constraints

- Look at the partially solved puzzle

```
  9END
 +10RE
  ----
```

- `10NEY`
- In the hundreds position, we have $E + 0 + C_{10} = N + 10 * C_{100}$, with C_{10} the 0/1 carry from the tens position
- NB: No carry C_{100} into the thousands, C_{100} = 0
- N must be equal to $E + 1$ with $C_{10} = 1$
- If $C_{10} = 0$, then $N = E$, not possible
- We can substitute $N = E + 1$ into our main equation, but keep $N = E + 1$ as well

Expert Mode Reasoning

Starting with

$$91 * E^{4..7} + 10 * R^{2..8} + D^{2..8} = 90 * N^{5..8} + Y^{2..8}$$

we get

$$91 * E^{4..7} + 10 * R^{2..8} + D^{2..8} = 90 * E^{4..7} + 90 + Y^{2..8}$$

Eliminating duplicate occurrences of E

$$\underbrace{E^{4..7} + 10 * R^{2..8} + D^{2..8}}_{26..95} = \underbrace{90 + Y^{2..8}}_{92..98}$$

shared range 92..95 To reach 92, R must be equal to 8, therefore N, E, D, Y must be less than 8

As $N = E + 1$, E must be less than 7

$$E^{4..6} + 10 * 8 + D^{2..7} = 90 + Y^{2..7}$$

Simplification yields

$$\underbrace{E^{4..6} + D^{2..7}}_{6..13} = \underbrace{10 + Y^{2..7}}_{12..17}$$

shared range 12..13 To reach 12 on LHS, E must be greater than 4, D must be greater than 5
To reach 13 in RHS, Y must be smaller than 4

$$E^{5..6} + D^{6..7} = 10 + Y^{2..3}$$

As both N and D must be in 6..7, no other variable can use those values
NB: This requires better reasoning than *forward checking* on the *alldifferent* constraint
If we remove 6 from the domain of E, E must be $E = 5$, and thus $N = 6$, due to $N = E + 1$
Alldifferent forces $D = 7$, leaving

$$5 + 7 = 10 + Y^{2..4}$$

This only leaves $Y = 2$

$$5 + 7 = 10 + 2$$

That is a solution, the constraint is satisfied

Expert Mode Summary

- Often there is more propagation that can be done
- Can be difficult/expensive to do
- Balancing
  - How much work it done at each step of search?
  - How many steps of search you need?
- For hard problems, doing all possible propagation may be exponential
- Not aware that any CP system does the full reasoning shown here

This is how people solve the puzzle by hand

- When writing the first version of this puzzle for CHIP (in 1986), we wanted to mimic the way we solve the puzzle by hand