Bull. Nov. Comp. Center, Comp. Science, 24 (2006), 45–55
© 2006 NCC Publisher

A reliable linear constraint solver for the UniCalc system

E. Botoeva, Yu. Kostov, E. Petrov

Abstract. In this paper we present a linear constraint solver for the UniCalc system, an environment for reliable solution of mathematical modeling problems.

1. Introduction

The UniCalc system is an environment for reliable solution of mathematical modeling problems. The UniCalc system is based on the constraint programming technology. The constraint solvers which make up the UniCalc system have been used by the Russian Research Institute of Artificial Intelligence to efficiently solve design and modeling problems in various areas of industry and science. The first version of the UniCalc system was released in 1990. First of all, the UniCalc system is intended for scientific staff, students, and engineers.

The core solver of the UniCalc system implements subdefinite calculations [4]. For an arbitrary system of constraints (equations, inequalities, Boolean statements), subdefinite calculations find a set containing all solutions (and, possibly, some non-solutions) to these constraints. Besides this core solver, the UniCalc system currently contains a symbolic solver handling the dependency problem which is well-known in the constraint programming community, a symbolic solver for linear equations, and a set of hard-coded strategies describing the typical ways of using these 3 solvers. The user interacts with the UniCalc system through a friendly interface.

Among software for solution of mathematical modeling problems, the UniCalc system is identified by 3 features: reliable calculations, a high-level modeling language, a user-friendly graphical interface. The term "reliable" means here "correct despite rounding errors and inaccuracy in the numerical input data". According to a recent survey [5], there are 4 more systems which, to some extent, have these 3 features: the Numerica system [8] (support was discontinued in 2003), the Premium Solver [1] (a plug-in for Microsoft Excel), the GlobSol solver [2] (no graphical interface, needs a FORTRAN-90 compiler), the ICOS solver [3] (no graphical interface, needs an external linear solver).

Like any constraint propagation algorithm that processes real numbers, subdefinite calculations converge slowly and/or produce excessively wide

enclosures on some problems. That undesirable behavior is observed when such an algorithm is applied to a special class of linear constraints [7]. On one hand, this fact impeded usage of the UniCalc system for economical modeling because linear constraints are very common in this domain. But on the other hand, this fact motivated us for working on a linear constraint solver for the UniCalc system.

Our work resulted in a reliable (in the same sense as above) solver for linear constraints, called UniCalc.LIN. This solver is integrated into the latest version of the UniCalc system.

In this paper we present the UniCalc.LIN solver (Section 2), compare it with the symbolic solver for linear equalities and describe a strategy of using the UniCalc linear solver (Section 3), present the results of experiments with the linear solvers (Section 4) and discuss our plans concerning solution of linear constraints in the UniCalc system (Section 5).

2. The UniCalc.LIN solver

In this section we present the UniCalc.LIN solver for the UniCalc system.

All solvers making up the UniCalc system are required to allow arbitrary mathematical models as the input data and return some mathematical model as the output data. To meet this requirement, the UniCalc.LIN solver performs the following conceptual steps: linear relaxation of the input model, non-reliable calculation of an enclosure for the solutions to a linear model thus obtained, adjustment of this enclosure which make it reliable. This reliable enclosure is returned to the UniCalc system in the form of linear inequalities each of which contains exactly one variable from the input model.

UniCalc.LIN calculates non-reliable enclosures using a 2-phase lexicographical simplex algorithm which is out of focus of this paper. We focus on linear relaxation and adjustment of non-reliable enclosures. These two topics are discussed in the following subsections. Note that, for a better performance, these two conceptual steps are merged in the actual implementation.

2.1. Linear relaxation

In this section we explain how the UniCalc.LIN solver constructs a linear program which logically follows from the input model. This step is called linear relaxation.

The UniCalc.LIN solver receives the input model in the form of a directed acyclic graph, called a syntactic DAG, whose nodes are expressions from the input model. In this DAG, the arcs connect expressions to their immediate subexpressions. Each node stores an enclosure for the range of

the corresponding expression. These enclosure are not requiered to be reliable to let syntactic DAGs represent any intermediate results. There are two types of enclosures: intervals for numerical expressions and subsets of {TRUE, FALSE} for Boolean expresions (like $\sin(x+y) = [-1/2, 1/2]$, etc.)

During linear relaxation, the UniCalc.LIN solver constructs an interval linear program which logically follows from the input model. For each expression e from this model, this program contains the following constraints:

- e, if e is a linear constraint;
- $e \in I$, if e is a linear expression with a reliable enclosure I;
- $0 = 0$ (no constraint) otherwise.

The constraints of the form "$e \in I$" are rewritten as inequalities; one of the bounds of I may be infinite.

Internally the UniCalc.LIN solver stores the bounds of interval enclosures in the floating point format. Because floating point numbers are of the form $m \cdot 2^e$ where m, e are some integer numbers, constants like 1/10, 1/3, etc. are represented by thin intervals to ensure that linear constraints do follow from the input model.

Consider the following input model:

$$\sin(x + y) = [-1/2, 1/2]; \sin(x - y) = [-1/2, 1/2].$$

Its syntactic DAG consists of the nodes x, y, $x+y$, $\sin(x+y)$, $x+y$, $\sin(x-y)$, $[-1/2, 1/2]$, $\sin(x + y) = [-1/2, 1/2]$, $\sin(x - y) = [-1/2, 1/2]$.

Let I, J, K, L be reliable enclosures stored by x, y, $x + y$, $x - y$ in the syntactic DAG. Linear relaxation produces the following 4 linear constraints: $x \in I$, $y \in J$, $x + y \in K$, $x - y \in L$. For any I, J, K, L, the set of solutions to these linear constraints includes the set of solutions to the input model. Moreover, if sine is monotonic on K and L, then the above solution sets are equal.

2.2. Reliable enclosure

In this section we explain how the UniCalc.LIN solver finds reliable enclosures for solutions to linear constraints.

Let C be the linear constraints produced by linear relaxation of the input model. If all calculations were exact, it would suffice to solve 2 linear programs for each variable x from C: $\min x$ s.t. C and $\max x$ s.t. C. The Cartesian product of the intervals of the form $[\min\{x|C\}, \max\{x|C\}]$ would be the wanted reliable enclosure.

Because the exact arithmetic operations on rational numbers may consume considerable amounts of memory (and CPU time), the UniCalc.LIN solver uses approximate arithmetic operations on floating point numbers

and the post-processing technique proposed in [6] and adjusted by us for the case of interval linear constraints generated during linear relaxation.

Given a linear program, any reliable enclosure for its solutions and any approximate linear programming solver, this post-processing technique produces another reliable enclosure (hopefully, a smaller one) for the solutions to the same linear program. The diameter of this output enclosure is proportional to the diameter of the input enclosure. The proportionality factor is determined by the accuracy of the approximate linear programming solver: the more accurate solver (smaller residual), the smaller factor. To weaken this limitation, the UniCalc system uses the UniCalc.LIN solver in cooperation with the symbolic solver for linear constraints. This topic is discussed in the next section.

3. The UniCalc.LIN solver vs. the symbolic solver for linear equations

In this section we give examples which illustrate weak and strong points of the UniCalc.LIN solver and the symbolic solver for linear equations. We also describe a strategy which increases the chances of using the right linear solver.

The symbolic solver for linear equations is a reliable implementation of Gaussian elimination using interval arithmetic. It solves linear equations very efficiently but may produce excessively wide enclosures if the equations have wide intervals on the right-hand side. Such equations emerge, for example, when inequalities are transformed into equalities by the symbolic solver for the dependency problem.

Consider the following example:

$$0 \le x + y \le 1$$
$$0 \le x - y \le 1$$

The UniCalc.LIN solver alone finds the smallest possible enclosure for the solutions to these inequalities: $x \in [0, 1]$, $y \in [-1/2, 1/2]$.

Cooperation of the symbolic solvers results in a non-trivial enclosure, but not the smallest one. First, the symbolic solver for the dependency problem transforms these inequalities into the following equations:

$$x + y = [0, 1]$$
$$x - y = [0, 1]$$

After that, depending on the order of elimination of variables, the symbolic solver for linear equations produces $x \in [-1/2, 3/2]$, $y \in [-1/2, 1/2]$ (x is eliminated first), or $x \in [0, 1]$, $y \in [-1, 1]$ (otherwise).

Like the symbolic solver for linear equations, the UniCalc.LIN solver may output wider enclosures than possible. This thing happens, if linear equations are ill-conditioned and the enclosure fed into the UniCalc.LIN solver initially is too loose. Under these conditions the symbolic solver for linear equations still finds relatively small enclosures, while the UniCalc.LIN solver cannot make the diameter of the initial enclosure small enough because of a considerable residual in the ill-conditioned equations.

Consider the following example:

$$\begin{aligned} &x + y = 3 \cdot 10^{-7} \\ &x + (1 + 10^{-7}) \cdot y = 10^{-7} \\ &x = y = [-10^7, 10^7] \end{aligned}$$

The solution to these equations is $x = 2 + 3 \cdot 10^{-7}$, $y = -2$. Because of the large condition number of this linear problem (about 10^7 in the sup norm), the UniCalc.LIN solver finds a relatively large enclosure for this solution: $x \in [1.94, 2.04]$, $y \in [-2.04, -1.96]$ (the bounds rounded outward; 2 decimal digits kept). The symbolic solver for linear equations finds an almost optimal enclosure: $x \in [2.00000029, 2.00000031]$, $y \in [-2.00000001, -1.99999999]$.

The above considerations lead us to the following strategy of using the solvers for linear constraints in the UniCalc system:

- divide the linear constraints C obtained by linear relaxation of the input model into linear equations E with thin intervals on the right-hand side and other linear constraints $C \backslash E$;
- solve E by the symbolic solver for linear equations; denote the output linear constraints by $\mathrm{Gauss}(E)$;
- solve $C \cup \mathrm{Gauss}(E)$ by the UniCalc.LIN solver.

4. Experimental results

In this section we compare the pure Simplex method, the pure Gaussian elimination method, and the strategy from the previous section.

We use the linear test-cases for the CoConUT framework. Those test-cases are minimization problems. For each test-case, we give the reliable lower bound on the objective found by the solvers and the best such bound known to the authors.

The test-cases ran on an Intel Celeron 1.5GHz CPU with 512Mb of RAM. The running time is less than 0.01 second for all the solvers and all the test-cases but reading2. On that latter test-case, the Gaussian elimination method runs in less than 0.01 second; the Simplex method and the strategy run in less than 0.22 second. The experimental results are provided in Figure 1. For the test-cases themselves, see Appendix A. Note that all variables were artificially bounded by the interval $[-10^{19}, 10^{19}]$.

Test-case	G.E. obj.	Sim. obj.	Strat. obj.	Best
degenpla	-10^{19}	-10^{19}	-10^{19}	-10^{19}
degenplb	-10^{19}	-10^{19}	-10^{19}	-10^{19}
extrasim	1	1	1	1
oet1	$-1.3533 \cdot 10^{18}$	-9641.0	-9641.0	0.53667
oet3	-10^{19}	$-0.9 \cdot 10^{-323}$	$-0.9 \cdot 10^{-323}$	0.00437
pt	$-1.1111 \cdot 10^{18}$	-9894.2	-9894.2	0.17834
reading2	-0.047947	-0.01302	-0.01302	-0.01302
simplpla	1	0	1	1
simplplb	1	1.0999	1.0999	1.0999
sipow1	-1.37639	-1.0515	-1.0515	-1.0515
sipow1m	-1.0001	-1.0001	-1.0001	-1.0001
sipow2	-1.9021	-1.1756	-1.1756	-1.1756
sipow2m	-1.3764	-1.1756	-1.1756	-1.1756
sipow3	0	0	0	0.41025
sipow4	-10^{19}	-22031	-22031	0.12952
supersim	0.66666	0.66666	0.66666	0.66666
tfi2	$-8.3334 \cdot 10^{17}$	-5273.0	-5273.0	0.64791
hilbert8	$-2.9248 \cdot 10^{-11}$	$-1.5462 \cdot 10^{12}$	$-2.9248 \cdot 10^{-12}$	0

Figure 1. Interval bounds found for each test-case by the linear solvers and the best lower bound known to the authors. G.E., Sim., and Strat. stand respectively for Gaussian elimination, Simplex method, and Strategy

5. Conclusion

In this paper we presented the UniCalc.LIN solver for reliable solution of linear constraints in the UniCalc system. In the constraint programming literature, most attention is paid to exhaustive search and solution of non-linear constraints rather than to reliable solution of linear constraints. However, during exhaustive search, any reliable solver for non-linear constraints "looks" at constraints very locally. Because of this fact, even complicated non-linear constraints "seem" almost linear to the solver; in fact, it is linear constraints that the solver has to handle most of the time when it solves hard non-linear constraints. This observation is the fundamental motivation for our work.

Currently, our work on reliable solution of linear models in the UniCalc system continues in the following areas:

- improvement of the linear relaxation step (doing linearization of non-linear constraints, taking into account the min and max operations, products and ratios of known signs);
- experiments with linear relaxations of a limited dimension to reduce

the CPU time consumed by the UniCalc.LIN solver on large linear models.

We thank the Russian Research Institute of Artificial Intelligence and A.P. Ershov Institute of Informatics Systems of the Russian Academy of Science for a financial support of this research.

References

[1] Frontline Systems and Spreadsheet Optimization. The premium solver web-site. — http://www.solver.com.

[2] Kearfott R. B. Rigorous Global Search: Continuous Problems. — Kluwer Academic Publishers, 1996.

[3] Lebbah Y., Michel C., Rueher M. Global filtering algorithms based on linear relaxations // Notes of the 2nd Intrnat. Workshop on Global Constrained Optimization and Constraint Satisfaction (Cocos'03), Lausanne, Switzerland, Nov. 2003. — `citeseer.ist.psu.edu/article/lebbah03global.html`

[4] Narinyani A. S. Subdefiniteness and basic means of knowledge representation // Computers and Artificial Intelligence. — 1983. — Vol. 2(5). — P. 443–452.

[5] Neumaier A. Complete search in continuous global optimization and constraint satisfaction // Acta Numerica. — Cambridge University Press, 2004. — P. 271–369.

[6] Neumaier A., Shcherbina O. Safe bounds in linear and mixed-integer programming // J. of Mathematical Programming. — 2004. — Vol. 99. — P. 283–296.

[7] Ushakov D. On local consistency of linear constraints (in Russian) // J. of Computation Technology. — 1999. — Vol. 4(4). — P. 76–79.

[8] Van Hentenryck P., Michel L., Deville Y. Numerica: a Modelling Language for Global Optimization. — The MIT Press, Cambridge, MA, 1997.

A. The linear test-cases

degenpla

```
//minimize f:
f=0.01*x[2]+33.333*x[3]+100.0*x[4]+0.01*x[5]
  +33.343*x[6]+100.01*x[7]+33.333*x[8]+133.33*x[9]+100.0*x[10];
-0.70785+x[1]+2*x[2]+2*x[3]+2*x[4]+x[5]+2*x[6]+2*x[7]+x[8]+2*x[9]
  +x[10] = 0;
0.326*x[1]-101*x[2]+200*x[5]+0.06*x[6]+0.02*x[7] = 0;
0.0066667*x[1]-1.03*x[3]+200*x[6]+0.06*x[8]+0.02*x[9] = 0;
0.00066667*x[1]-1.01*x[4]+200*x[7]+0.06*x[9]+0.02*x[10] = 0;
0.978*x[2]-201*x[5]+100*x[11]+0.03*x[12]+0.01*x[13] = 0;
0.01*x[2]+0.489*x[3]-101.03*x[6]+100*x[12]+0.03*x[14]+0.01*x[15] = 0;
0.001*x[2]+0.489*x[4]-101.03*x[7]+100*x[13]+0.03*x[15]+0.01*x[16] = 0;
```

```
0.001*x[3]+0.01*x[4]-1.04*x[9]+100*x[15]+0.03*x[18]+0.01*x[19] = 0;
0.02*x[3]-1.06*x[8]+100*x[14]+0.03*x[17]+0.01*x[19] = 0;
0.002*x[4]-1.02*x[10]+100*x[16]+0.03*x[19]+0.01*x[20] = 0;
-2.5742-6*x[11]+0.00252*x[13]-0.61975*x[16]+1.03*x[20] = 0;
-0.00257*x[11]+0.25221*x[12]-6.2*x[14]+1.09*x[17] = 0;
0.00629*x[11]-0.20555*x[12]-4.1106*x[13]+101.04*x[15]+505.1*x[16]
  -256.72*x[19] = 0;
0.00841*x[12]-0.08406*x[13]-0.20667*x[14]+20.658*x[16]+1.07*x[18]
  -10.5*x[19] = 0;
```

degenplb

```
//minimize f:
f=-1*(0.01*x[2]+33.333*x[3]+100.0*x[4]+0.01*x[5]
  +33.343*x[6]+100.01*x[7]+33.333*x[8]+133.33*x[9]+100.0*x[10]);
-0.70785+x[1]+2*x[2]+2*x[3]+2*x[4]+x[5]+2*x[6]+2*x[7]+x[8]+2*x[9]
  +x[10] = 0;
0.326*x[1]-101*x[2]+200*x[5]+0.06*x[6]+0.02*x[7] = 0;
0.0066667*x[1]-1.03*x[3]+200*x[6]+0.06*x[8]+0.02*x[9] = 0;
0.00066667*x[1]-1.01*x[4]+200*x[7]+0.06*x[9]+0.02*x[10] = 0;
0.978*x[2]-201*x[5]+100*x[11]+0.03*x[12]+0.01*x[13] = 0;
0.01*x[2]+0.489*x[3]-101.03*x[6]+100*x[12]+0.03*x[14]+0.01*x[15] = 0;
0.001*x[2]+0.489*x[4]-101.03*x[7]+100*x[13]+0.03*x[15]+0.01*x[16] = 0;
0.001*x[3]+0.01*x[4]-1.04*x[9]+100*x[15]+0.03*x[18]+0.01*x[19] = 0;
0.02*x[3]-1.06*x[8]+100*x[14]+0.03*x[17]+0.01*x[19] = 0;
0.002*x[4]-1.02*x[10]+100*x[16]+0.03*x[19]+0.01*x[20] = 0;
-2.5742-6*x[11]+0.00252*x[13]-0.61975*x[16]+1.03*x[20] = 0;
-0.00257*x[11]+0.25221*x[12]-6.2*x[14]+1.09*x[17] = 0;
0.00629*x[11]-0.20555*x[12]-4.1106*x[13]+101.04*x[15]+505.1*x[16]
  -256.72*x[19] = 0;
0.00841*x[12]-0.08406*x[13]-0.20667*x[14]+20.658*x[16]+1.07*x[18]
  -10.5*x[19] = 0;
-x[1]+300*x[2]+0.09*x[3]+0.03*x[4] = 0;
```

extrasim

```
x>=0;
//minimize f:
f = x+1;
x+2*y-2.0 = 0;
```

oet1

```
const int M=5;
const real lower = 0.0;
const real upper = 2.0;
const real diff = upper-lower;
const real h =diff/M;
//minimize f:
f=u;
all(i=0,1,M;
  u-(i*h+lower)*x[1]-exp(i*h+lower)*x[2]-(i*h+lower)^2 >= 0;
  u+(i*h+lower)*x[1]+exp(i*h+lower)*x[2]+(i*h+lower)^2 >= 0);
```

oet3

```
const int M=10;
const real lower = 0.0;
const real upper = 1.0;
const real diff = upper-lower;
const real h=diff/M;
all(i=0,1,M; c[i+1]=sin(i*h+lower));
//minimize f:
f=u;
all(i=0,1,M;
  u-c[i+1]-x[1]-(i*h+lower)*x[2]-(i*h+lower)^2*x[3] >= 0;
  u+c[i+1]+x[1]+(i*h+lower)*x[2]+(i*h+lower)^2*x[3] >= 0);
```

pt

```
const int M=5;
const real lower = 0.0;
const real upper = 1.0;
const real diff = upper-lower;
const real h = diff/M;
//minimize f:
f=u;
all(i=0,1,M;
  -(i*h+lower)+(i*h+lower)^2+u+((i*h+lower)-3*(i*h+lower)^2+1)*x >= 0);
```

reading2

```
const int N=5;
const real A=0.07716;
const real H=1/N;
all(i=1,1,N+1;x2[i]=[-0.125,0.125]);
all(i=1,1,N+1;u[i]=[-1,1]);
const real c1 = H/(8*PI^2);
all(i=1,1,N+1;c2[i]=-0.5*H*cos(2*PI*(i-1)*H));
f=sum (i=1,1,N; c2[i+1]*x1[i+1]-c2[i]*x1[i+1]+c1*(u[i+1]+u[i]));
all(i=1,1,N;
  (x1[i+1]-x1[i])/H-0.5*(x2[i+1]+x2[i]) = 0;
  (x2[i+1]-x2[i])/H-0.5*(u[i+1]+u[i]) = 0);
x1[1] = 0.0;
x2[1] = 0.0;
```

simplpla

```
x[1]>=0;
x[2]>=0;
//minimize f:
f=2*x[1]+x[2];
x[1]+x[2]-1.0 = [0,1e19];
x[1]+2*x[2]-1.5 = [0,1e19];
```

simplplb

```
x[1]>=0;
x[2]>=0;
//minimize f:
f=1.5*x[1]+x[2];
x[1]+x[2]-1.0 >= 0;
x[1]+2*x[2]-1.2 >= 0;
2*x[1]+x[2]-1.2 >= 0;
```

sipow1

```
const int m = 10;
//minimize f:
f = x[2];
all(j=1,1,m; x[1]*cos(2*PI*j/m)+x[2]*sin(2*PI*j/m)+1.0 >= 0);
```

sipow1m

```
const int m = 10;
//minimize f:
f=x[2];
all(j=1,1,m;
  x[1]*cos(2*PI*(j+0.5)/m)+x[2]*sin(2*PI*(j+0.5)/m)+1.0 >= 0);
```

sipow2

```
const int M=10;
//minimize f:
f=x[2];
all(i=1,1,M/2;
    1+x[1]*cos(4*PI*i/M)+x[2]*sin(4*PI*i/M) >= 0);
x[1]+1 >= 0;
```

sipow2m

```
const int M=10;
//minimize f:
f=x[2];
all(i=1,1,M/2;
  1+x[1]*cos(4*PI*(i+0.5)/M)+x[2]*sin(4*PI*(i+0.5)/M) >= 0);
x[1]+1 >= 0;
```

sipow3

```
const int M = 24;
const real STEP = 8/M;
all(j=1,1,M/8; xi[j]=0);
all(j=M/8+1,1,M/4; xi[j]=(j-1)*STEP-1);
all(j=M/4+1,1,3*M/8; xi[j]=1);
all(j=3*M/8+1,1,M/2; xi[j]=(j-1)*STEP-3);
all(j=1,1,M/8; eta[j]=(j-1)*STEP);
all(j=M/8+1,1,M/4; eta[j]=1);
```

```
all(j=M/4+1,1,3*M/8; eta[j]=(j-1)*STEP-2);
all(j=3*M/8+1,1,M/2; eta[j]=0);
//minimize f:
f=x[4];
all(j=1,1,M/2;
  x[1]+x[4]+xi[j]*x[2]+eta[j]*x[3]-xi[j]^2*eta[j] >= 0;
  x[1]+xi[j]*x[2]+eta[j]*x[3]-xi[j]^2*eta[j] <= 0);
```

sipow4

```
const int M=10;
all(j=1,1,M/2;
  xi[j] = 0.5-sqrt(0.5)*cos((PI/4)-j*(2*PI/M));
  eta[j] = 0.5-sqrt(0.5)*sin((PI/4)-j*(2*PI/M)));
//minimize f:
f=x[4];
all(j=1,1,M/2;
  x[1]+x[4]+xi[j]*x[2]+eta[j]*x[3]-xi[j]^2*eta[j] >= 0;
  x[1]+xi[j]*x[2]+eta[j]*x[3]-xi[j]^2*eta[j] <= 0);
```

supersim

```
x >= 0;
//minimize f:
f=x;
x+2*y-2 = 0;
2*x+y-2 = 0;
```

tfi2

```
const int M = 10;
const real h = 1/M;
//minimize f:
f=x[1]+0.5*x[2]+x[3]/3;
all(i=0,1,M;-x[1]-i*h*x[2]-(i*h)^2*x[3]+tg(i*h) <= 0);
```

hilbert8

```
//minimize f:
f=x1+x2+x3+x4+x5+x6+x7+x8;
+x1/1+x2/2+x3/3+x4/4+x5/5+x6/6+x7/7+x8/8 = 0;
+x1/2+x2/3+x3/4+x4/5+x5/6+x6/7+x7/8+x8/9 = 0;
+x1/3+x2/4+x3/5+x4/6+x5/7+x6/8+x7/9+x8/10 = 0;
+x1/4+x2/5+x3/6+x4/7+x5/8+x6/9+x7/10+x8/11 = 0;
+x1/5+x2/6+x3/7+x4/8+x5/9+x6/10+x7/11+x8/12 = 0;
+x1/6+x2/7+x3/8+x4/9+x5/10+x6/11+x7/12+x8/13 = 0;
+x1/7+x2/8+x3/9+x4/10+x5/11+x6/12+x7/13+x8/14 = 0;
+x1/8+x2/9+x3/10+x4/11+x5/12+x6/13+x7/14+x8/15 = 0;
```