Bull. Nov. Comp. Center, Comp. Science, 29 (2009), 79–88
© 2009 NCC Publisher

The uniform constraint solving API for UniCalc

E. S. Petrov

Abstract. The paper describes a programming interface that we have developed to separate constraints, solvers, and cooperation strategies in a constraint programming toolkit UniCalc. The interface is simple but it allows us to express such constraint solving techniques as increasing the level of consistency, various strategies for exhaustive search, and cooperation of constraint, symbolic and linear interval solvers.

1. Introduction

Constraint programming is a highly successful technology for solving the combinatorial problems (scheduling, staff allocation, assignment, routing, design, etc.) and non-linear constraints. Constraint programming toolkits are used by companies such as Amazon.com, British Airways, Chevron, Cisco, Ford, HP, KLM, Lockheed Martin, Nestle, Oracle, Proctor & Gamble, Renault, SNCF, UPS, and Volvo [1].

The main purpose of constraint programming toolkits is efficient solution of as wide class of constraints as possible. Because an efficient "solver of everything" hardly exists, the need for combination and interaction of constraint solvers is widely recognized. This area of constraint programming is called cooperative constraint solving.

UniCalc is a constraint programming toolkit developed by the Russian Research Institute of Artificial Intelligence and A.P. Ershov Institute of Informatics Systems to answer the research requests from customers in industry, science and Russian government. Most of such requests are related to reliable numerical solution of non-linear design and mathematical modeling problems.

Currently UniCalc consists of 4 constraint solvers and a number of cooperation strategies hard-coded for those solvers. The paper describes the programming interface that we have developed to separate constraints, solvers, and cooperation strategies in UniCalc. We refer to this interface as Uniform constraint solving application programming interface (UCS API). The basic UCS API is simple but it allows us to express such constraint solving techniques as increasing the level of consistency, various strategies for exhaustive search, and cooperation of constraint, symbolic and linear interval solvers.

The paper is organized as follows. Section 2 gives an overview of existent software that exploits/enables interaction between solvers. Section 3 describes the architecture of the UniCalc system. Section 4 defines the basic

UCS API. Section 5 gives usage examples for the basic UCS API. Section 6 concludes the paper.

2. Related work

Solution of non-linear constraints is a popular tool of analysis and modeling in such areas as finance, oil and gas, energy, and industry. The corresponding constraint programming toolkits are numerous but each solves efficiently only the constraints of a certain small class (linear, geometric, etc.).

The need for combination and interaction of constraint solvers is widely recognized. There have been developed many systems for cooperative constraint solving like GMACS [2], BALI [3], Eclipse [6], Prolog IV [7], Meta-S [4], and CFLP [5].

Though the need for combination and interaction of constraint solvers is widely recognized, there is no widely accepted API for cooperative constraint solving. In the above systems, constraint solvers cooperate through ad hoc APIs that were never released to wider public. In the existent public APIs (see below), constraint solvers "cooperate" if they use a common format to store and exchange data, such as constraints, vectors, matrices, etc.

The most constraint-oriented APIs of this type are the COCONUT API for continuous computer math in C++ [10], the GAMS and AMPL APIs for non-linear programming in C and FORTRAN [8, 9]. Less specific to constraint solving are the Sage API for computer math in the Python language [11], the Mathematica and MatLab APIs that can be used to do computer math in C and FORTRAN [12, 13].

The point of this paper is that an API for interaction of constraint solvers can capture more than sharing a format to exchange data.

3. UniCalc

The constraint programming toolkit UniCalc has been first released by the Russian Research Institute of Artificial Intelligence (RRIAI) in 1990. Since that time 5 major releases of UniCalc have been published (the most recent in 2007). UniCalc has evolved from a single solver called subdefinite calculations [14] to a system of 4 solvers and a number of efficient cooperation and search strategies [18].

The first version of UniCalc was released for UNIX in 1987. Actually, this version was an extended demo for the capabilities of subdefinite calculations.

The next version of UniCalc was released for MS DOS in 1991. This version could solve only the medium size sets of constraints because of MS DOS limitations. However, a number of Russian companies were satisfied the functionality provided by UniCalc and bought this version.

UniCalc 3.14 was released for MS Windows 3.1 in 1994. The user interface was reworked and computing capabilities were extended, e.g. the size limitation on the set of constraints was relaxed. In 1995-1996, this version was ported to MS Windows 95/NT and UNIX dialects for Sun, IBM, HP, and SGI workstations.

UniCalc 4 was released in 2001. This version was less successful than UniCalc 3.14 because of lower product quality that was caused by a significant rotation of the UniCalc developers in 2000-2001. Many developers moved from RRIAI to other software companies.

UniCalc 5 was released in 2002 and since that time a number of minor releases followed in 2004-2007. The user interface was reworked, the parser of constraints was sped up 100-1000x for large sets of constraints, new solvers were implemented for linear constraints, and cooperation and search strategies were also implemented.

UniCalc has been used by many Russian companies, such as AvtoVAZ, Novosibirsk Chemical Concentrates Plant, Beriev Aircraft Company, and Kamov Design Bureau. UniCalc has also been used by French companies Dassault Aviation and Dassault Systemes. The library for interval calculations from UniCalc has been integrated into the constraint logic programming system Eclipse [6].

As of 2009, UniCalc is the only constraint programming toolkit for solution of non-linear constraints that is 100% designed and developed in Russian Federation.

Figure 1 shows the architecture of UniCalc. Boxes show the components of UniCalc, the text explains which data are passed between components, the arrows show in which direction the data are passed. The solvers are subdefinite calculations, the interval Gauss-Seidel solver, interval Simplex solver, and symbolic occurrence reducer. Subdefinite calculations are a propagation-based solver for non-linear constraints. The interval Gauss-Seidel solver and interval Simplex solver implement the corresponding methods of interval linear analysis [16, 15].

The strategies are 3B consistency [17], pre-processing with occurrence reducer, subdefinite calculations mixed with exhaustive search, fixed point for any combination of interval Gauss-Seidel, interval Simplex, and subdefinite calculations.

4. The Basic UCS API

In UniCalc, we use the UCS API to separate constraint solving from constraint solving strategies (constraint solving applications in general). The strategies work with variables, constraints, sets of constraints, and constraint solvers. These objects are represented by handles of type UCS_HANDLE. A few of these handles are implementation specific named constants; most

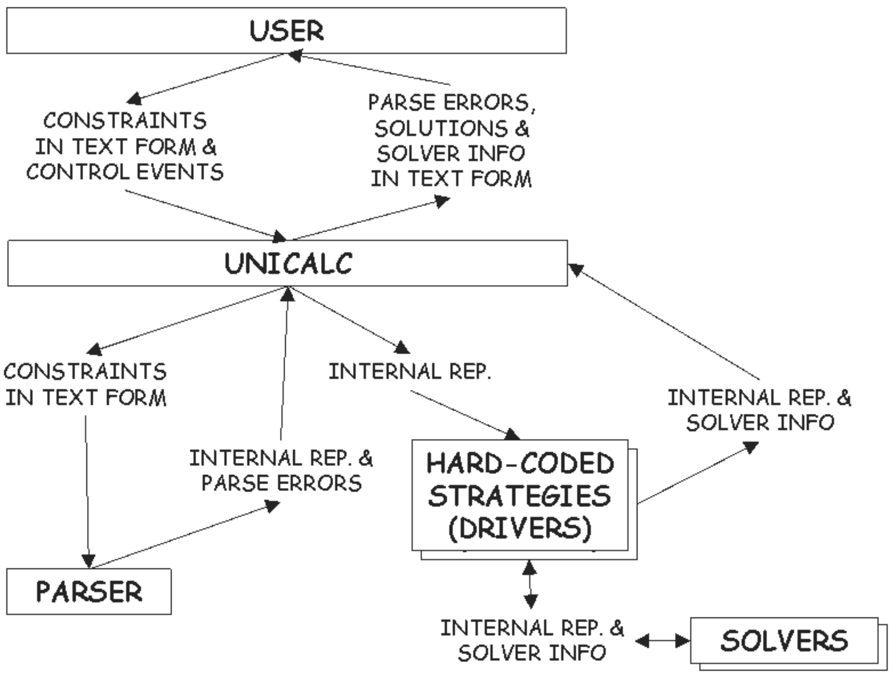
USER
CONSTRAINTS IN TEXT FORM & CONTROL EVENTS
PARSE ERRORS, SOLUTIONS & SOLVER INFO IN TEXT FORM
UNICALC
CONSTRAINTS IN TEXT FORM
INTERNAL REP.
INTERNAL REP. & SOLVER INFO
INTERNAL REP. & PARSE ERRORS
HARD-CODED STRATEGIES (DRIVERS)
PARSER
INTERNAL REP. & SOLVER INFO
SOLVERS

Figure 1. The architecture of UniCalc

are created by the strategy.

The strategies can do around 20 basic actions that correspond to the basic UCS API entries. Normally the actions produce a handle, or a UCS_ENUM value (UCS_YES, UCS_NO, UCS_UNCERTAIN, or UCS_ERROR), or an index (non-negative integer).

The actions on invalid combinations of objects produce either UCS_NO_OBJECT (if the action produces a handle), or UCS_ERROR (otherwise). The strategies can use a service function to get a human-readable message that explains the error.

A solver S can be used on a set C of constraints in 3 ways.

- Check whether S is relevant to solving C with sCheckRelevant(S, C)
- Apply S to C with sApply(S, C)
- Ask S to indicate the “freest” variable in C with sFindFreest(S, C)

The solvers are free to change C in any way and indicate any variable as the freest. The expected behavior is however to preserve the solution space of C and to indicate the least controlled variable as the 4 solvers available in UniCalc do (see below).

The strategies can create new solvers with sNew(Relevant, Apply, Freest) given the pointers to the functions Relevant, Apply, Freest that take a set of constraints and return a value of the corresponding type. If Relevant is NULL, the new solver is relevant to any set of constraints. If Apply is NULL, the new solver always returns UCS_ERROR. If Freest is NULL, the new solver always returns UCS_NO_OBJECT as the freest variable.

The solvers available in UniCalc are represented by 4 named handles. These handles are specific to our implementation of the UCS API.

- Subdefinite calculations UCS_SUBDEFINITE_CALC
- Interval Gauss-Seidel solver UCS_GAUSS_SEIDEL
- Interval Simplex solver UCS_SIMPLEX
- Symbolic occurrence reducer UCS_OCCURENCE_REDUCER

Usually, a set of constraints is passed to a strategy as a parameter. A set of constraints can also be created from a zero-terminated string text with pCreate(text). If text starts with "file://", the rest of it must point to an existent file, and the set of constraints is read from there. Otherwise text must be a set of constraints in the UniCalc language, e.g. pCreate("x=cos(x+y); y=sin(x-y);"). The empty set of constraints is created with pCreate(""). See [18] and the references therein for a description of the UniCalc language.

The strategies can test whether the set C of constraints has a unique solution with pCheckUnique(C), or has no solution with pCheckInconsistent(C), or has a small ("small" is configured outside the UCS API) solution space with pCheckSmallSpace(C). The first two tests may return UCS_UNCERTAIN if answering definitely would take much resource.

The strategies can query the number of variables in a set C of constraints with vNumberOf(C), get the i-th variable in C with vGet(C, i), make the bounds on variables V and W disjoint with vMakeDisjoint(V, W), assign the bounds on a variable W to a variable V with vAssignBounds(V, W).

The strategies can query the number of constraints in a set C of constraints with cNumberOf(C), get the i-th constraint in C with cGet(C, i), insert and delete a constraint c into/from a set C of constraints with cInsert(C, c) and cDelete(C, c).

The strategies can get the index of a variable or constraint X in a set C of constraints with pFind(C, X). The strategies can copy an object O (except solvers) with oCopy(O) and close a handle to an object O with oClose(O).

Memory and concurrency management for the UCS API objects are implementation-specific. UniCalc implements a reference counting mechanism to decide when the memory occupied by a particular object can be freed. UniCalc also use handles to serialize the concurrent write access to the corresponding objects.

5. Examples of using the UCS API

The basic UCS API is simple but allows us to express increment of the level of consistency (so called “3B consistency”), pre-processing with occurrence reducer, subdefinite calculations mixed with exhaustive search, fixed point for any combination of interval Gauss-Seidel, interval Simplex, and subdefinite calculations. In this section, we give a simplified code for these strategies.

5.1. Symbolic pre-processing

Symbolic pre-processing is a sequence of the symbolic occurrence reducer and a solver. Symbolic pre-processing improves the bounds calculated by propagation-based solvers like UCS_SUBDEFINITE_CALC and enlarges the set of constraints available to linear solvers like UCS_GAUSS_SEIDEL and UCS_SIMPLEX.

The corresponding function symbolic_prep is straightforward. To keep the error message, we return after an error immediately.

UniCalc uses this strategy for S = UCS_SUBDEFINITE_CALC, S = UCS_GAUSS_SEIDEL, and S = UCS_SIMPLEX.

```
void symbolic_prep(UCS_HANDLE S, UCS_HANDLE C)
{
    if (sApply(UCS_OCCURENCE_REDUCER, C) == UCS_ERROR)
        return;
    sApply(S, C);
}
```

5.2. 3B consistency

3B consistency is a constraint solving technique that tries to “trim” the bounding box found by a constraint solver with the help of reasoning by contradiction.

The function refuteBounds(S, C, V, i) checks, with the help of the solver S, whether the bounds on the variable V are consistent with the bounds on the i-th variable in C. The function _3b(S, C) enforces 3B consistency with the help of the solver S. For simplicity, we omit error handling.

UniCalc uses this strategy for S = UCS_SUBDEFINITE_CALC.

```
int refuteBounds(
    UCS_HANDLE S, UCS_HANDLE C, UCS_HANDLE V, int i)
{
    int refute = 0;
    UCS_HANDLE copy = oCopy(C);
    vAssignBounds(vGet(copy, i), V);
    sApply(S, copy);
    refute = pCheckInconsistent(copy) == UCS_YES;
    oClose(copy); // also closes the handle returned by vGet
    return refute;
}

void _3b(UCS_HANDLE S, UCS_HANDLE C)
{
    while (1) {
        int i, loop = 0;
        for (i = 0; i != vNumberOf(C); i++) {
            UCS_HANDLE vari = vGet(C, i);
            UCS_HANDLE V = oCopy(vari), W = oCopy(vari);
            vMakeDisjoint(V, W);
            if (refuteBounds(S, C, V, i)) {
                vAssignBounds(vari, W);
                sApply(S, C);
                loop = 1;
            } else if (refuteBounds(S, C, W, i)) {
                vAssignBounds(vari, V);
                sApply(S, C);
                loop = 1;
            }
            oClose(vari);
            oClose(V);
            oClose(W);
        }
        if (!loop) break;
}}
```

5.3. Exhaustive search and subdefinite calculations

Exhaustive search is a common-knowledge constraint solving technique that applies reasoning by case until all solutions to the set of constraints are isolated.

The function exhaustiveSearch(S, C) implements the breadth-first search for solutions to the set C of constraints using the solver S to discard boxes that do not contain solutions to C. The strategy assumes that S indicates the same variable as the freest in C and its copy. For simplicity, we omit error handling.

UniCalc uses exhaustive search for S = UCS_SUBDEFINITE_CALC.

```
#include <list> // use STL template list
void exhaustiveSearch(UCS_HANDLE S, UCS_HANDLE C)
```

```
{
    std::list<UCS_HANDLE> space, solutions, invariant;
    space.push_back(C);
    while (!space.empty()) {
        UCS_HANDLE C = space.front();
        space.pop_front();
        if (sCheckRelevant(S, C) != UCS_YES) {
            invariant.push_back(C);
            continue;
        }
        sApply(S, C);
        if (pCheckUnique(C) == UCS_YES) {
            solutions.push_back(C);
        } else if (pCheckInconsistent(C) == UCS_YES) {
            oClose(C);
            continue;
        } else {
            UCS_HANDLE copy = oCopy(C);
            UCS_HANDLE freestC = sFindFreest(S, C);
            UCS_HANDLE freestCopy = sFindFreest(S, copy);
            vMakeDisjoint(freestC, freestCopy);
            oClose(freestC);
            oClose(freestCopy);
            space.push_back(C);
            space.push_back(copy);
}}}
```

5.4. Fixed point of linear solvers and subdefinite calculations

A fixed point of a set of solvers is another common-knowledge constraint solving technique for solvers that share the same representation of constraints.

The function fixedPoint(S, nS, nLoop, C) tries to calculate the fixed point for the set S of nS solvers. If the fixed point is not reached in nLoop steps, then the strategy returns the approximation obtained after the nLoop-th step. For simplicity, we omit error handling.

UniCalc uses fixed point
S[] = {UCS_SUBDEFINITE_CALC, UCS_GAUSS_SEIDEL,
UCS_SIMPLEX},
S[] = {UCS_SUBDEFINITE_CALC, UCS_GAUSS_SEIDEL}, and
S[] = {UCS_SUBDEFINITE_CALC, UCS_SIMPLEX}.

```
void fixedPoint(
    UCS_HANDLE S[], size_t nS, size_t nLoop, UCS_HANDLE C)
```

```
{
    while (nLoop) {
        int i;
        for (i = 0; i != n; i++) {
            if (sCheckRelevant(S[i], C) == UCS_YES) {
                sApply(S[i], C);
                nLoop = 0;
}}}}
```

6. Conclusion

In this paper, we introduced the Uniform constraint solving application programming interface (the UCS API). The basic UCS API consists of around 20 entries but allows us to express such constraint solving techniques as 3B consistency, various strategies for exhaustive search, cooperation of constraint, symbolic and linear interval solvers.

We would like that, having read this paper, the reader keep the following two points about API's for the real world cooperative constraint solving:

- An API for cooperative constraint solving can be simple.
- An API for cooperative constraint solving can express more than sharing data format.

We will focus our future work on a language that will allow the users of UniCalc to specify constraint solving strategies along with the constraints to be solved. This language will give to the users of UniCalc more freedom than just selection from the set of hard-coded constraint solving strategies.

We thank the Russian Research Institute of Artificial Intelligence and A.P. Ershov Institute of Informatics Systems for supporting this work.

References

[1] van Beek P., Walsh T. Principles of Constraint Programming and Constraint Processing: A Review // AI Magazine. – 2004. – Vol. 25, N 4.

[2] Kleymenov A., Semenov A. Using a Cooperative Solving Approach to Global Optimization Problems // Lect. Notes Comput. Sci. – Berlin: Springer, 2005. – Vol. 3478. – P. 86–100.

[3] Monfroy E. A Solver Collaboration in BALI // Proc. IJCSLP 1998. – The MIT Press, 1998. – P. 349–350.

[4] Frank S., Hofstedt P., Mai P.R. A Flexible Meta-solver Framework for Constraint Solver Collaboration // Lect. Notes Comput. Sci. – Berlin: Springer, 2003. – Vol. 2821 – P. 520–534.

[5] Estevez-Martin S., Fernandez A. J., Hortala-Gonzalez T., Saenz-Perez F. A Proposal for the Cooperation of Solvers in Constraint Functional Logic Programming // Electronic Notes in Theor. Comput. Sci. (ENTCS) archive. – Elsevier Science Publishers, 2007. – Vol. 188. – P. 37–51.

[6] Apt K. R., Wallace M. Constraint Logic Programming using Eclipse. – Cambridge University Press, 2007. – Open source project Eclipse Web site http://87.230.22.228.

[7] Colmerauer A. Le Manuel de Prolog IV. – PrologIA: 1996.

[8] Brooke A., Kendrick D., Murows A. Gams Release 2.25: A User's Guide/Book. – Course Technology, 1992. – The GAMS project Web site http://www.gams.com.

[9] Fourer R., Gay D. M., Kernighan B. W. AMPL: A Modeling Language for Mathematical Programming. – Duxbury Press, 2003.

[10] The COCONUT Project. –
Web site http://www.mat.univie.ac.at/users/coconut.

[11] The Sage Open Source Project. –
Web site http://www.sagemath.org/doc/reference.

[12] Wolfram S. The Mathematica Book. – Cambridge University Press, 1999. – Online manual http://reference.wolfram.com

[13] Hanselman D., Littlefield B. R. Mastering MATLAB 6. – Prentice Hall, 2000. – Online manual http://www.mathworks.com/access/helpdesk/help/techdoc

[14] Narin'yani A. S. Sub-Definiteness, Over-Definiteness, and Absurdity in Knowledge Representation (Some Algebraic Aspects) // Proc. Conf. Artificial Intelligence Applications 1985. – IEEE Computer Society/North-Holland, 1985. – P. 142–143

[15] Neumaier A., Shcherbina O. Safe bounds in linear and mixed-integer programming // Math. Programming. – 2004. – Vol. 99 – P. 283–296.

[16] Kearfott B. Preconditioners for the interval Gauss-Seidel method // SIAM J. on Numerical Analysis. – 1990. – Vol. 27, Iss. 3. – P. 804–822.

[17] Collavizza M., Deloble F., Rueher M. Comparing partial consistencies // J. Reliable Computing. – 1999. – Vol. 5. – P. 1–16.

[18] Botoeva E., Kostov Yu., Petrov E. A reliable linear constraint solver for the UniCalc system // Joint Bull. of NCC&IIS. Ser.: Comput. Sci. – 2006. – Iss. 24. – P. 101–111.