制約解集合プログラミングシステムの設計方式に関する考察

宋 剛秀　則武 治樹　番原 睦則　田村 直之　井上 克巳

解集合プログラミング (Answer Set Programming; ASP) は，一階述語論理に基づいた論理型プログラミング言語の一種である．記号上の探索問題を簡潔に記述できることが特長の一つであるが，算術的な制約を直接表現することはできない．一方，**制約プログラミング** (Constraint Programming; CP) は算術的な制約を対象としている．このような背景から ASP と CP の統合が注目を集めており，両者の機能を併せ持つ**制約解集合プログラミング** (Constraint Answer Set Programming; CASP) が研究されるようになった．本論文では CASP システムの一構成方式について考察する．ここでは，CASP システム muffin を ASP システム上に実装する．muffin の CASP 言語のベースは ASP であり，ASP の記号的な制約の記述に加えて算術的な制約の記述が可能である．また ASP の記号や項を変数名やコレクション名として用いることができるため，簡潔な問題記述が可能である．これまで提案されてきた CASP システムと異なり，muffin は前処理方式とライブラリ方式を融合したシステムであることにも特長がある．論文では実装した muffin を用いた基礎実験の結果についても報告を行う．

Answer Set Programming (ASP) is a logic programming based on the first order logic, which has a rich yet simple modeling language for symbolic constraints. However, it is not so well to represent arithmetic and global constraints for which Constraint Programming (CP) has a suited language. To integrate advanteges of both programming paradigms, Constraint Answer Set Programming (CASP) has been studied. In this paper, we consider a design of CASP systems and its implementation called muffin. The CASP language of muffin is based on ASP and can represent arithmetic and global constraints in addition to symbolic constraints. Symbols and terms in ASP can be represented by names of variables and collections in muffin. Another feature is the translation between CASP and ASP: it is processed by using pre-processing and ASP library. We also report preliminaly experiments on muffin.

1　はじめに

解集合プログラミング (Answer Set Programming; ASP) [21] は，一階述語論理に基づいた論理型プログラミング言語の一種である．ASP は Prolog 等の従来の論理型プログラミング言語の拡張として，2000 年頃から活発に研究されてきた [12] [29]．研究内容は基礎から応用にまで及び，近年の論理プログラミングに関する国際会議 (ICLP) においても，ASP に関する発表は非常に多く，この分野における主要なトピックの 1 つとなっている．

ASP の研究内容は基礎的な研究だけにとどまらず，実際的な応用まで様々である，近年の成功例としては，その記述力と ASP ソルバーの性能を活かして，ASP を時間割問題に適用した研究がある [5]．この研究では時間割問題が解集合プログラムによって簡潔に記述でき，その解が ASP ソルバーによってシミュレーティド・アニーリング法などの，従来研究と比較しても高速に計算されることが報告されている．

その他の ASP の特徴を以下に述べる．

- 知識表現言語として優れ，扱える問題は標準論理プログラムを用いる場合はクラス NP，選言論

Studies on a Design of Constraint Answer Set Programming Systems

Takehide Soh, Mutsunori Banbara, Naoyuki Tamura, 神戸大学情報基盤センター, Information Science and Technology Center, Kobe University.

Haruki Noritake, 神戸大学大学院システム情報学研究科, Graduate School of System Informatics, Kobe University.

Katsumi Inoue, 国立情報学研究所, National Institute of Informatics.

理プログラムの場合はクラス $\sum_2^P$ に属する [13].

- 安定モデル意味論に基づいたモデルの探索・列挙が行える.
- 記述言語は非 Horn 節である.
- 一階述語形式で問題を記述可能で，与えられた問題を自然・簡潔に記述できる.
- 演繹データベース，論理プログラミング，非単調推論を起源としている [21].
- ASP ソルバーの開発が活発に行われている．特に与えられた ASP を基礎化するソルバー gringo [11][14] と基礎化された ASP から解集合を求める clasp [16] が事実上の標準となっている．この他に Cmodels[27] は ASP を充足可能性判定 (SAT) 問題に変換して求解を行う SAT 型システムである.

制約充足問題 (Constraint Satisfaction Problem; CSP) は，各変数に与えられたドメインから値を割り当てることで，与えられた制約のすべてを満たすことができるかどうかを判定する問題である [7][9]．すべての制約を満たす値割り当てが存在する場合，元の CSP は充足可能 (SAT) であるといい，その値割り当てが解となる．値割り当てが存在しない場合，元の CSP は充足不能 (UNSAT) であるという．本論文では CSP を整数の有限領域上の問題として扱う．制約には算術論理演算等で記述されるものの他に，alldifferent 等に代表されるいわゆるグローバル制約がある.

グローバル制約 (global constraint) は，複数の変数に対する複雑な (しかし意味のある) 条件を簡潔に表すために導入された．例えば，alldifferent$(x_1, x_2, \ldots, x_n)$ は，x_i が互いに異なることを表す．$x_i \neq x_j$ を個別に記述するよりも簡潔になり，また効率の良い解法アルゴリズムの存在が知られている [20]．このようなグローバル制約は，記述性および効率性の向上を目的として制約ソルバーや制約プログラミングシステムに数多く取り入れられている．これらのグローバル制約を集約した **Global Constraint Catalog** [†1] には，300 以上のグローバル制約が紹介されている.

その他の CSP の特徴を以下に述べる.

- NP 完全問題に属する.
- 研究・応用が活発に行われている [8][10][18][19][22][23][31][34][35][37][38].
- 様々なソルバーが開発されており，中でも SAT 型システムが成功を収めている．特に，SAT 型制約ソルバーの Sugar[33] [†2] が 2008–2009 年国際 CSP ソルバー競技会において 2 年連続優勝 (GLOBAL 部門) する等の実績を残している.
- 共通の制約モデリング言語がなく，ソルバーにより異なる．なお，上記の国際 CSP ソルバー競技会においては，XML 形式で記述された CSP(XCSP) が用いられている.
- どの言語も算術制約やグローバル制約の記述性は保証されているが，配列を扱う構文が存在しない．また，項を変数として扱うことができない.

ここまで述べたように，ASP と CSP はそれぞれ異なる特長を持っている．ASP は記述言語として優れており，複雑な問題を簡潔に記述することができるが，算術的な記述やグローバル制約に関しては CSP の方が優れている．しかし，それぞれの言語はソルバーの系統が全く異なり，互いの問題を扱うことができない．そのため，各々の良い点を統合した言語およびソルバーの開発が注目を集めている [26].

制約解集合プログラミング (Constraint Answer Set Programming; CASP) は，解集合プログラミングをベースとし，さらに制約プログラミング分野で活発に研究されている算術制約やグローバル制約の処理を兼ね備えたプログラミング言語である．両方の利点を兼ね備えることにより，グローバル制約を含んだ規模の大きな問題を，ASP や CSP と比較して簡単な構文で扱うことができる．これまでいくつか ASP と CSP の両方に関わる研究がされている [30][12][3][1][2][15][28]．これらの包括的なサーベイとしては論文 [26] が詳しい．上記に挙げた中でも特に CASP システム実現している clingcon [30] は ASP 言語を拡張して算術制約を取り扱えるようにしてお

†1 http://www.emn.fr/z-info/sdemasse/gccat/

†2 http://bach.istc.kobe-u.ac.jp/sugar/

り，制約ソルバーと ASP ソルバーをハイブリッドに実行することで解を求める．CASP システム inca [12] も ASP 言語を算術制約を扱えるようにした拡張言語を入力とするところは clingcon と同じであるが，初期の inca については入力を全て ASP の記号制約に符号化してから ASP ソルバーによって求解する前処理方式を取るという特徴がある．

本論文では CASP システムの一構成方式として従来の ASP 言語の述語を用いて算術制約を記述し，前処理方式によって求解する方法を提案する．特に符号化アプローチの部分については算術制約の処理に ASP で記述されたライブラリを用いており，今後の開発で別の符号化方法を少ないコストで導入可能であることが特長である．提案する CASP システムの実装として muffin を開発した．muffin は順序符号化 [34] [35] を用いて算術制約およびグローバル制約を ASP の記号制約へと符号化する．順序符号化は国際 CSP ソルバー競技会の優勝ソルバーである Sugar [33] にも採用されており，前処理方式に組込むことで muffin の高速な求解を期待出来る．

muffin の性能を評価するための部分実験を行った．muffin は CASP システムであるので，ASP, CSP, CASP の問題を入力とすることができるが，今回は CSP の問題をベンチマークとして用いて，CSP に対する muffin の性能を調査する．この理由は ASP において muffin で採用しているバックエンドのソルバー gringo と clasp の性能は既に過去の ASP ソルバー競技会 [17] の結果より十分に証明されていること，CASP についてはまだ研究途上であり十分な数のベンチマークがないことである．よって今回は部分実験として国際 CSP ソルバー競技会で用いられた 1797 問を用いて行った結果について報告を行う．

以下では，第 2 節 CASP システムの設計方式について述べ，第 3 節でその実装である muffin の説明を行う．第 4 節では muffin の基本性能の評価実験の結果について説明する．

2 CASP システムの設計

本研究では, CASP システムの実現方法として符号化アプローチをとる．入力言語は ASP をベースにするが，整数変数，算術制約，グローバル制約を記述できるようにする．

本論文では CASP 言語のベースとして ASP の記述言語である標準論理プログラム (Normal Logic Program; NLP) を用いる．詳細については文献 [13] [21] に詳しいがここでは簡単に説明を行う．NLP は以下の形式のルールの集合として構成される．

$$L_1 \leftarrow L_2, \ldots, L_m, not\ L_{m+1}, \ldots, not\ L_n$$

ここで $m \geq n \geq 0$ であり, L_i は正または負のリテラルで, not は**デフォルトの否定**, “,” は連言を表す．このルールにおいて $\leftarrow$ の左部を**ルールのヘッド**，右側を**ルールのボディ**と呼ぶ．ボディが空のルールを**ファクト**と呼び，ヘッドが空のルールを**制約**と呼ぶ．

以上が NLP におけるルールの簡単な説明である．上記に加えていくつかの ASP ソルバーでは組合せ問題を ASP で解くために便利なアグリゲート (aggregate) と呼ばれる表記法がいくつか用意されている．これらの詳細についても文献 [13] [21] に詳しい．

2.1 CASP 言語の設計

本研究で提案する CASP 言語の設計方針は上述の ASP をベースにするということ，整数変数，算術制約，グローバル制約を記述できるようにするということの 2 点である．本研究では ASP 上の述語を用いてこれらの方針に基づいた言語を実現する．提案する CASP の基礎となる述語と意味は以下になる．

- `variable(V, int(LB, UB))`
  ドメインの上限下限が `UB` と `LB` となる整数変数 V の定義.
- `collection(C, X)`
  変数または制約 X がコレクション C に属することの定義.
- `constraint(A)`
  制約 A の定義.

特に `collection` の導入によりグローバル制約を特定のコレクション変数に対して適用だけで，そのコレクション変数に属する全ての変数と制約に適用出来るようになるという利点がある．`constraint` で定義される制約 A は，表 1, 表 2, 表 3, 表 4 の述語を組み合わせたものである．なおここでは算術制約，比較

表 1　算術制約

述語	意味
neg(X)	$-X$
add(X1, X2, ..)	$X_1 + X_2 + \ldots$
sub(X1, X2, ..)	$X_1 - X_2 - \ldots$
mul(X1, X2)	$X_1 \times X_2$
div(X1, X2)	X_1/X_2
mod(X1, X2)	X_1 を X_2 で割った余り
min(X1, X2)	X_1 と X_2 の小さい値
max(X1, X2)	X_1 と X_2 の大きい値

表 2　比較制約

述語	意味
eq(X1, X2)	$X_1 = X_2$
ne(X1, X2)	$X_1 \neq X_2$
le(X1, X2)	$X_1 \leq X_2$
lt(X1, X2)	$X_1 < X_2$
ge(X1, X2)	$X_1 \geq X_2$
gt(X1, X2)	$X_1 > X_2$

表 3　論理制約

述語	意味
not(L)	$\neg L$
and(L1, L2, ..)	$L_1 \wedge L_2 \wedge \ldots$
or(L1, L2, ..)	$L_1 \vee L_2 \vee \ldots$
imp(L1, L2)	$L_1 \rightarrow L_2$
xor(L1, L2)	L_1, L_2 の排他的論理和
iff(L1, L2)	L_1, L_2 の同値関係
if(L, T, F)	論理式 L が成り立つなら T, 異なるなら F

制約，論理制約，グローバル制約を区別して記述するが，以下では特に問題がない場合には記号制約との対比としてこれらの制約を総称して算術制約と呼ぶ．述語の中で可変長引数 $X_1, X_2, \ldots$ を引数に取るものはコレクション変数を引数にして記述する事ができる．また，表 4 のグローバル制約については一例であり，後述の Sugar(3.1 節) で読み込めるグローバル制約を記述することが可能である．

表 4　グローバル制約

述語	意味
alldifferent(X1, X2, ...)	$X_1, X_2, \ldots$ は全て異なる値である
predicate((P, Xs), (P(Xs)))	P(Xs) を定義 ($Xs \Leftrightarrow X_1, X_2 \ldots$)

```
 1: (int x11 1 9)
 2: (int x12 1 9)
    ...
 9: (int x33 1 9)
10: (= (+ x11 x12 x13) 15)
11: (= (+ x21 x22 x23) 15)
12: (= (+ x31 x32 x33) 15)
13: (= (+ x11 x21 x31) 15)
14: (= (+ x12 x22 x32) 15)
15: (= (+ x13 x23 x33) 15)
16: (= (+ x11 x22 x33) 15)
17: (= (+ x13 x22 x31) 15)
18: (alldifferent x11 x12 x13 x21
              x22 x23 x31 x32 x33)
```

図 1　Sugar CSP による 3×3 の魔方陣の記述

```
 1: #const n = 3.
 2: #const sum = n*(n*n+1)/2.
 3: num(1..n).
 4: variable(x(I,J), int(1,n*n)) :- num(I), num(J).
 5: collection(xs, x(I,J)) :- num(I), num(J).
 6: collection(row(I), x(I,J)) :- num(I), num(J).
 7: collection(col(J), x(I,J)) :- num(I), num(J).
 8: collection(d1, x(I,I)) :- num(I).
10: collection(d2, x(I,n-I+1)) :- num(I).
11: constraint(alldifferent(collection(xs))).
12: line(row(I)) :- num(I).
13: line(col(J)) :- num(J).
14: line(d1;d2).
15: constraint(eq(add(collection(L)), sum)) :-
                                        line(L).
```

図 2　CASP を用いた 3×3 の魔方陣の記述

2.2　CASP を用いた魔方陣の記述

魔方陣は $n \times n$ の行列について，縦・横・斜めのいずれについても，数字の合計が同じになるように $1 \sim n^2$ の異なる数字を配置する問題である．例えば $x(1,1), x(1,2) \ldots x(3,3)$ の 3×3 行列について考える場合には縦・横・斜めのいずれについても，総和が

15 になるように各 x に 1 ～ 9 のそれぞれ異なる値を割当てる．

図 1 は Sugar CSP の，図 2 は CASP の 3×3 の魔方陣に対する記述である．CSP の 18 行に対し，CASP は 14 行で記述することができる．

また $n \times n$ の魔方陣について考える場合，CSP で記述すると $n^2 + 2n + 3$ の行数が必要になるが，CASP ならば 3×3 の場合と同様に 14 行だけで記述する事ができる．さらに，魔方陣のサイズを変える場合，CSP は全体的に書きなおす必要があるが，CASP は図 2 の 1 行目の変数を変えるだけで対応できる．また，CSP と比べて CASP は項をそのまま変数として記述している．例えば $(2,1)$ のマスの変数を記述するとき，CSP では $x21$ となるのに対し，CASP では $x(2,1)$ と分かりやすく記述することができる．

2.3 CASP を用いたナップサック問題の記述

ナップサック問題とはある容量 C を持つナップサックと値段と価値の対によって表される n 種類の品物が与えられた時に，容量 C を超えない範囲で価値の和が最大になるように品物を選ぶ問題である．ここでは例として以下のような品物 a, b, c, d, e と容量 $C = 15$ が与えられた時に，価値が 19 以上になるような組合せを求める問題を考える．

名前	a	b	c	d	e
容量	3	4	5	7	9
価値	4	5	6	8	10

この問題は Sugar CSP では図 3，CASP では図 4 のように記述できる．CSP は算術制約が得意であり，ナップサック問題については簡潔に書けるが，変数が増えたり条件が変わると全体的に書きなおす必要がある点は魔方陣と同様である．一方，CASP は `item(e, 9, 10)`. までの行がインスタンスであり，それ以降がモデリングである．値段や価値の上限や下限といった条件を変数，(名前，値段，価値) の組の要素を `item` として記述しており，モデリングが書けていれば条件や要素が変わっても分かりやすく書き換えることができる．

```
1: (int a 0 1)
2: (int b 0 1)
3: (int c 0 1)
4: (int d 0 1)
5: (int e 0 1)
6: (<= (+ (* 3 a) (* 4 b) (* 5 c)
          (* 7 d) (* 9 e)) 15)
7: (>= (+ (* 4 a) (* 5 b) (* 6 c)
          (* 8 d) (* 10 e)) 19)
```

図 3　Sugar CSP によるナップサック問題の記述

```
 1: #const ub_price = 15.
 2: #const lb_value = 19.
 3: item(a, 3, 4).
 4: item(b, 4, 5).
 5: item(c, 5, 6).
 6: item(d, 7, 8).
 7: item(e, 9, 10).
 8: variable(x(Item), int(0,1)) :-
             item(Item, Price, Value).
 9: collection(prices, mul(Price,x(Item))) :-
             item(Item, Price, Value).
10: collection(values, mul(Value,x(Item))) :-
             item(Item, Price, Value).
11: constraint(
       le(add(collection(prices)), ub_price)).
12: constraint(
       ge(add(collection(values)), lb_value)).
```

図 4　CASP を用いたナップサック問題の記述

2.4 CASP 処理系の設計

設計の指針を以下に示す．

- ASP では CSP の記述は扱えないため，算術制約やグローバル制約といった CSP の記述部分を分割し，処理を行う．
- グローバル制約には可変長引数を扱う制約も多いが，ASP 上では可変長引数を扱う制約を記述することは難しい上に動作が遅いため，ASP 外で変換を行う．
- ソルバーや符号化法を交換可能なシステムにする．

このシステムの利点は，従来のソルバーや正規化手法の良いところを少ない開発コストで利用できることである．また，算術制約を処理するライブラリを書き換えることで他の符号化法に変更したり，ASP ソル

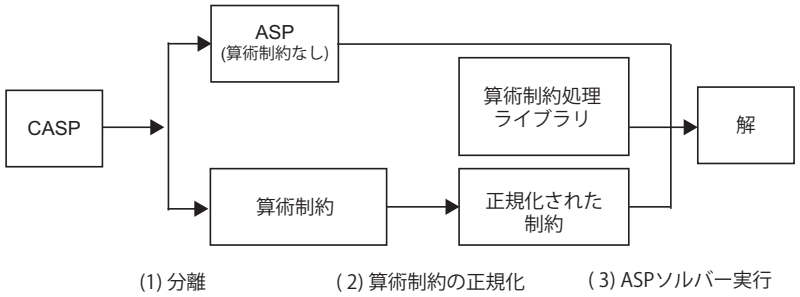

図 5　CASP システムの設計方式

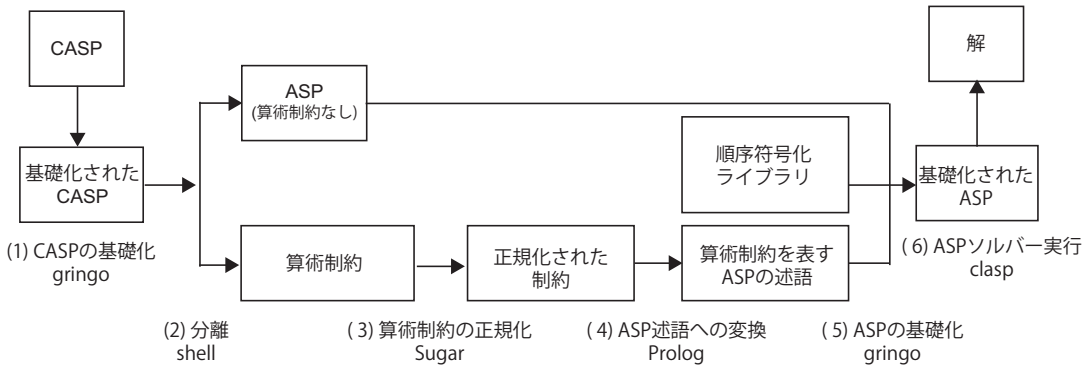

図 6　CASP システム muffin の構成

バーを交換することで様々な ASP ソルバーをそのまま利用することが可能である．

図 5 に提案する CASP システムの設計方式を示す．

1. CASP を ASP の部分と CSP の部分に分割する．
2. CSP を，算術制約を処理するライブラリと ASP ソルバーの組み合わせで読み込めるような ASP に変換する．
3. 2 つの ASP とライブラリと組み合わせ，ASP ソルバーで求解する．

上記の手順により CASP の解を ASP ソルバーによって求める．

3　CASP システム muffin の実装

3.1　全体の処理

図 6 に実際に実装したシステムを示す．なお CSP と ASP 間の変換は今回は Prolog を用いて行った．

- CASP の正規化
  CASP は CSP と ASP が混在しており，CSP の構文に対して ASP の述語が書いてあることも多い．設計した CASP の構文は gringo で読み込めるようになっているため，まずは gringo で述語の解釈を行う．
- ASP と CSP の分離
  CSP として処理する部分が `variable`, `collection`, `constraint` の 3 つで記述がまとめてあるため，シェルの `grep` を用いることにより簡単に分離できる．
- 分離した CSP 部分の正規化
  Sugar を用いる．Sugar は SAT 型の CSP ソルバーであるが，CSP を正規化して出力する機能もある．Sugar を用いるメリットとして以下の要素が挙げられる．
  – 明らかに不要な探索範囲の削減や，可変長引数の制約とグローバル制約を全て 3 項以下の制約に正規化する機能がある．この機能により，ASP で可変長引数を扱う点につい

て考慮する必要がなくなる．また，ASP のみで処理するよりも高速である．
  - CASP だけでなく，Sugar CSP や XCSP も読み込める．
  - Sugar が改良された際にはそのまま流用できる．
- 算術制約処理ライブラリの実装
  算術制約を処理する方法として，以下のような符号化法がこれまでに提案されている．
  - 直接符号化法 [10] [31] [38]
  - 順序符号化法 [34] [35]
  - 支持符号化法 [19] [23]
  - 対数符号化法 [18] [22]

  今回のシステムでは，順序符号化法 (Order Encoding; OE) を用いて算術制約処理ライブラリを実装する (以下 oe.lp と参照)．順序符号化法の内容については後述する．順序符号化法は，求解困難な問題として知られるショップ・スケジューリング問題，二次元矩形パッキング問題，組合せテストのテストケース生成問題に対して，未知の最適値を決定するなど優れた性能を示している [4] [6] [24] [25] [32] [34]．また，単位伝播に優れており，記号上の制約として扱う上で高い効果が得られる．
- ASP ソルバー
  ASP を正規化する gringo と，SAT ソルバーとしても動作する clasp の組み合わせを用いる．

3.2 順序符号化法

順序符号化法 [34] [35] では，各整数変数 z について，そのドメインが $\{a_1, a_2, \ldots, a_n\}$ の時 (ただし $a_1 < a_2 < \ldots < a_n$)，$z \leq a_i$ を表す $n-1$ 個の命題変数 $p(z \leq a_1)$, $p(z \leq a_2)$, $\ldots$, $p(z \leq a_{n-1})$ を用いる．なお，$z \leq a_n$ は常に真であるため，命題変数 $p(z \leq a_n)$ は不要である．また，これらの命題変数間の関係を表す以下の節を用いる．

$$\neg p(z \leq a_i) \vee p(z \leq a_{i+1}) \quad (1 \leq i \leq n-2)$$

例えば，変数 z のドメインが $\{0, 1, 2, 3\}$ の場合，3 つの命題変数 $p(z \leq 0)$, $p(z \leq 1)$, $p(z \leq 2)$ を用い，以下の節を追加する．

$$\neg p(z \leq 0) \vee p(z \leq 1) \quad \neg p(z \leq 1) \vee p(z \leq 2)$$

この時，上記の節を充足可能にする真理値割り当ては 4 通りあり，それぞれ $z = 0$, $z = 1$, $z = 2$, $z = 3$ に対応する．

$p(z \leq 0)$	$p(z \leq 1)$	$p(z \leq 2)$	解釈
1	1	1	$z = 0$
0	1	1	$z = 1$
0	0	1	$z = 2$
0	0	0	$z = 3$

制約については，制約に違反する範囲を符号化すればよい．一般の線形制約については，以下のようになる．今，a_i を非零の整数定数，c を整数定数，z_i を互いに異なる整数変数とする．この時，制約 $\sum_{i=1}^{n} a_i\, z_i \leq c$ は以下のように符号化できる．

$$\bigwedge_{\Sigma_{i=1}^{n} b_i = c-n+1} \bigvee_{i} (a_i\, z_i \leq b_i)^{\#}$$

ここで b_i は，$\sum_{i=1}^{n} b_i = c - n + 1$ を満たすように動くとし，変換 $()^{\#}$ は以下のように定義する．

$$(a\, z \leq b)^{\#} \quad \equiv \quad \begin{cases} p(z \leq \lfloor b/a \rfloor) & (a > 0) \\ \neg p(z \leq \lceil b/a \rceil - 1) & (a < 0) \end{cases}$$

ただし，z の取り得る最小値未満の a については $p(z \leq a)$ を偽に変換し，最大値以上については真に変換する．例えば，整数変数 w, z のドメインが $\{0, 1, 2, 3\}$ の時，制約 $w - z \leq -1$ は以下の 4 つの節に符号化される．

$$\neg p(z \leq 0) \qquad p(w \leq 0) \vee \neg p(z \leq 1)$$
$$p(w \leq 1) \vee \neg p(z \leq 2) \qquad p(w \leq 2)$$

ここで，$p(w \leq 0) \vee \neg p(z \leq 1)$ は「$w \leq 0$ または $z > 1$」であること，すなわち「$w \geq 1$ かつ $z \leq 1$」が制約に違反する領域であることを表している．

図 7 に $w - z \leq -1$ の充足領域を表す．制約を満たす点は $\bullet$，違反点は $\times$ でプロットされている．

$w \neq z$ の場合，一旦 $w - z \leq -1 \vee z - w \leq -1$ のように線形制約に置き換え，さらに Tseitin 変換を用

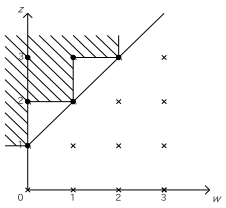

図 7　$w - z \leq -1$ の解領域を表したグラフ

```
dom(V,LB..UB) :- var(V), lb(V,LB), ub(V,UB).
{ p(V,A) : dom(V,A) } :- var(V).
:- p(V,A), not p(V,A+1), dom(V,A+1), var(V).
:- not p(V,UB), var(V), ub(V,UB).
```

図 8　整数変数とドメインの定義

いて $(p \vee q) \wedge (\neg p \vee w - z \leq -1) \wedge (\neg q \vee z - w \leq -1)$ と変換してから符号化する (p, q は新しい命題変数). したがって $w \neq z$ は以下の 9 個の節に符号化される.

$$p \vee q$$

$\neg p \vee \neg p(z \leq 0)$	$\neg q \vee \neg p(w \leq 0)$
$\neg p \vee p(w \leq 0) \vee \neg p(z \leq 1)$	$\neg q \vee p(z \leq 0) \vee \neg p(w \leq 1)$
$\neg p \vee p(w \leq 1) \vee \neg p(z \leq 2)$	$\neg q \vee p(z \leq 1) \vee \neg p(w \leq 2)$
$\neg p \vee p(w \leq 2)$	$\neg q \vee p(z \leq 2)$

順序符号化法では，直接符号化，支持符号化，対数符号化等と比較し，より短い節が生成される．また，順序符号化法で生成した節に対しては，単位伝播が整数制約上での範囲伝播に対応することが知られている [36].

3.3　算術制約処理ライブラリの実装

3.2 節で述べたような順序符号化を実装するために ASP 上で次のように定義を行う.

変数とドメインの定義 (図 8).　1 行目は変数 V，上限 UB，下限 LB が事実として記述されると，対応するドメイン `dom(V,A)` ($LB \leq A \leq UB$) が定義されることを示している．2 行目でそれぞれの変数 V に対して，`dom(V,A)` を満たすような V, A の組全てに対し，命題変数 `p(V,A)` が定義される．ASP では `:-` より左側を空節にすると右側の否定条件になるため，3 行目は $\neg p(V, A) \vee p(V, A+1)$ を表している．また，4 行目は上限のドメイン $p(V, UB)$ が常に真であることを示している.

```
% le0/2: X <= c
:- le0(X,C), not p(X,C), dom(X,C).
:- le0(X,C), lb(X,LB), C < LB.

% lt0/2: X < c
le0(X,C-1) :- lt0(X,C).

% gt0/2: X > c
:- gt0(X,C), p(X,C), dom(X,C).
:- gt0(X,C), ub(X,UB), C >= UB.

% ge0/2: X >= c
gt0(X,C-1) :- ge0(X,C).

% eq0/2: X == c
le0(X,C) :- eq0(X,C).
ge0(X,C) :- eq0(X,C).

% ne0/2: X != c
1 { gt0(X,C), lt0(X,C) } :- ne0(X,C).
```

図 9　比較制約の定義

```
% le0/3: aX <= c
:- le0(A,X,C), A>0, C>0, not p(X,D), dom(X,D),
                                      D=C/A.
:- le0(A,X,C), A>0, C<0, not p(X,D), dom(X,D),
                                   D=(C-A+1)/A.
:- le0(A,X,C), A<0, C>0, p(X,D-1), dom(X,D),
                                     D=C/A.
:- le0(A,X,C), A<0, C<0, p(X,D-1), dom(X,D),
                                  D=(C+A+1)/A.
:- le0(A,X,C), A==0, C<0.
:- le0(A,X,C), A>0, C==0, not p(X,0), dom(X,0).
:- le0(A,X,C), A<0, C==0, p(X,-1), dom(X,-1).
:- le0(A,X,C), A>0, lb(X,LB), A*LB>C.
:- le0(A,X,C), A<0, ub(X,UB), A*UB>C.
```

図 10　制約 $AX \leq C$ の定義

等号，不等号の定義 (図 9).　X を変数，C を定数とする.

- $X \leq C$ (`le0/2`)：1 行目は $X \leq C$ ならば $p(X, C)$ が真であること，2 行目は C が下限未満の値であれば定義が成り立たないことを表している.
- $X < C$ (`lt0/2`)：$X \leq C$ を用いて定義される.
- $X > C$ (`gt0/2`)：1 行目は $X > C$ ならば $p(X, C)$ が偽であること，2 行目は C が上限以上の値であれば定義が成り立たないことを表して

```
% ge0/3: aX >= c
le0(MA,X,MC) :- ge0(A,X,C), MA=(-1)*A, MC=(-1)*C.

% eq0/3: aX == c
le0(A,X,C) :- eq0(A,X,C).
ge0(A,X,C) :- eq0(A,X,C).

% le/5: aX+bY <= c  (aX <= c - bY)
le0(A,X,C) :- le(A,X,0,Y,C).
le0(A,X,T) :- le(A,X,B,Y,C), B<0, p(Y,DY), dom(Y,DY), T=(C-B*DY).
le0(A,X,T) :- le(A,X,B,Y,C), B>0, not p(Y,DY-1), dom(Y,DY), T=(C-B*DY).

% ge/5: aX+bY >= c
le(MA,X,MB,Y,MC) :- ge(A,X,B,Y,C), MA=(-1)*A, MB=(-1)*B, MC=(-1)*C.

% eq/5: aX+bY == c
le(A,X,B,Y,C) :- eq(A,X,B,Y,C).
ge(A,X,B,Y,C) :- eq(A,X,B,Y,C).

% le/7: aX+bY+cZ <= d  (aX + bY <= d - cZ)
le(A,X,B,Y,D) :- le(A,X,B,Y,0,Z,D).
le(A,X,B,Y,T) :- le(A,X,B,Y,C,Z,D), C<0, p(Z,DZ), dom(Z,DZ), T=(D-C*DZ).
le(A,X,B,Y,T) :- le(A,X,B,Y,C,Z,D), C>0, not p(Z,DZ-1), dom(Z,DZ), T=(D-C*DZ).

% ge/7: aX+bY+cZ >= d
le(MA,X,MB,Y,MC,Z,MD) :- ge(A,X,B,Y,C,Z,D), MA=(-1)*A, MB=(-1)*B, MC=(-1)*C, MD=(-1)*D.

% eq/7: aX+bY+cZ == d
le(A,X,B,Y,C,Z,D) :- eq(A,X,B,Y,C,Z,D).
ge(A,X,B,Y,C,Z,D) :- eq(A,X,B,Y,C,Z,D).
```

図 11　3 項制約の定義

```
% false causes unsat.
:- false.

% remove domain
:- p(V,A-1), not p(V,A), remdom(V,A).
:- not p(V,A-1), p(V,A), remdom(V,A).
```

図 12　追加で必要な定義

いる．

- $X \geq C$ (ge0/2)：$X > C$ を用いて定義される．
- $X = C$ (eq0/2)：$X \leq C \wedge X \geq C$ で定義．
- $X \neq C$ (ne0/2)：$X < C \vee X > C$ で定義．

1 項の線形制約 ($AX \leq C$) の定義 (図 10).　X を変数，A と C を定数とする．定数同士の四則演算は直接記述しても問題はない．$AX \leq C$ を単純に変形すると $X \leq C/A$ であるが，ドメインが正だけではなく負の可能性もあり，小数点以下を切り捨てる際に不都合が生じる．故に A と C が正か負か，あるいは 0 かで細かく場合分けして記述する必要がある．

3 項以下の線形制約の定義 (図 11).　$AX \geq C$, $AX \neq C$ については $AX \leq C$ を用いて定義する．また，2 項と 3 項の線形制約についても，それぞれ 1 項と 2 項の線形制約を使うことにより簡潔に定義できる．ただし，係数の正負により場合分けをする必要がある．

その他の留意すべき点 (図 12).　Sugar による処理で明らかに制約を満たせないことが分かった場合，Sugar の出力は `false.` のみとなるため，`false.` が偽であるという制約を書いておく必要がある．

また，変数が離散的に定義される事も考慮する必要がある．例えば $V \in \{1 \ldots 4,\ 7 \ldots 10\}$ の場合である．変数 V の欠けているドメイン I を定義から取り除くと

```
:-p(x(1,1),1),not p(x(1,1),2).
:-p(x(1,1),2),not p(x(1,1),3).
...
:-p(x(1,1),8),not p(x(1,1),9).
:-not p(x(1,1),9).
```

図 13 $x(1,1)$ **の基礎化例**

```
le(1,x(1,1),1,x(2,1),6):-not p(x(3,1),8).
le(1,x(1,1),1,x(2,1),7):-not p(x(3,1),7).
le(1,x(1,1),1,x(2,1),8):-not p(x(3,1),6).
le(1,x(1,1),1,x(2,1),9):-not p(x(3,1),5).
le(1,x(1,1),1,x(2,1),10):-not p(x(3,1),4).
le(1,x(1,1),1,x(2,1),11):-not p(x(3,1),3).
le(1,x(1,1),1,x(2,1),12):-not p(x(3,1),2).
le(1,x(1,1),1,x(2,1),13):-not p(x(3,1),1).
le(1,x(1,1),1,x(2,1),14).
```

図 14 $x(1,1)+x(1,2)+x(1,3) \leq 15$ **の基礎化例**

```
le0(1,x(1,1),-1) :-
   le(1,x(1,1),1,x(2,1),8),not p(x(2,1),8).
le0(1,x(1,1),0) :-
   le(1,x(1,1),1,x(2,1),8),not p(x(2,1),7).
le0(1,x(1,1),1) :-
   le(1,x(1,1),1,x(2,1),8),not p(x(2,1),6).
le0(1,x(1,1),2) :-
   le(1,x(1,1),1,x(2,1),8),not p(x(2,1),5).
le0(1,x(1,1),3) :-
   le(1,x(1,1),1,x(2,1),8),not p(x(2,1),4).
le0(1,x(1,1),4) :-
   le(1,x(1,1),1,x(2,1),8),not p(x(2,1),3).
le0(1,x(1,1),5) :-
   le(1,x(1,1),1,x(2,1),8),not p(x(2,1),2).
le0(1,x(1,1),6) :-
   le(1,x(1,1),1,x(2,1),8),not p(x(2,1),1).
le0(1,x(1,1),7) :-
   le(1,x(1,1),1,x(2,1),8).
```

図 15 $x(1,1)+x(2,1)+x(3,1) \leq 15$ **の基礎化例**

きは `remdom(V,I)` と定義し，順序符号化法においては $p(V,I)$ と $p(V,I-1)$ が同値関係になるように処理する．例の場合だと，`var(V). lb(V,1). ub(V,10).` の後に，`remdom(V,5). remdom(V,6).` を定義する．

```
:-le0(1,x(1,1),9),not p(x(1,1),9).
:-le0(1,x(1,1),1),not p(x(1,1),1).
:-le0(1,x(1,1),2),not p(x(1,1),2).
:-le0(1,x(1,1),3),not p(x(1,1),3).
:-le0(1,x(1,1),4),not p(x(1,1),4).
:-le0(1,x(1,1),5),not p(x(1,1),5).
:-le0(1,x(1,1),6),not p(x(1,1),6).
:-le0(1,x(1,1),7),not p(x(1,1),7).
:-le0(1,x(1,1),8),not p(x(1,1),8).
```

図 16 $x(1,1) \leq C$ **の基礎化例**

3.4 gringo による算術制約の基礎化

gringo で，3.3 節の oe.lp と，CASP から生成された 2 つの ASP コードを読み込んで正規化 ASP を生成する．

整数変数の宣言は，oe.lp により対応するドメインと命題変数が生成される．例えば，
`var(x(1,1)). lb(x(1,1), 1). ub(x(1,1), 9).`
について生成される ASP 式は，図 13 のようになる．

線形制約については，それ自身は一時変数と同様に変数記号として扱われる．制約内に線形制約が存在する場合，gringo により変数割り当てが総当りで行われ，命題変数に関わる制約式が自動生成される．例えば，

$$x(1,1)+x(2,1)+x(3,1) \leq 15$$

について，まず 3 項制約の定義により $x(3,1)$ のそれぞれの場合について 2 項制約を生成する (図 14)．次に，一例として $x(3,1) > 6$ の場合について考える．すなわち $x(1,1)+x(2,1) \leq 8$ が成り立つ場合，さらに $x(2,1)$ のそれぞれの場合について 1 項制約を生成する (図 15)．1 項制約については安易に命題変数の制約に置換ができる (図 16)．

このように算術制約を記号上の制約に落としこむことで，CSP の要素を ASP として扱い，求解することが可能となる．生成される制約が膨大になるが，順序符号化法は制約伝播に優れているため，ほとんどの制約は高速で処理することができる．

4 CSP を用いた部分実験

実装した muffin の基本性能を評価するための CSP を用いた部分実験を行った．

muffin は CASP システムであるので，ASP, CSP, CASP の問題を入力とすることができるが，今回は CSP の問題をベンチマークとして用いた．この理由は ASP において muffin で採用しているバックエンドのソルバー gringo と clasp の性能は既に過去の ASP ソルバー競技会 [17] の結果より十分に証明されていること，CASP についてはまだ研究途上であり十分な数のベンチマークがないことである．よって今回は国際 CSP ソルバー競技会で用いられた問題によって部分実験を行う．

以下の実験において実験環境は Linux マシン (CPU 3.0GHz, メモリ 6GB)，問題の変換からソルバーによる求解までの全体のタイムアウトは 600 秒とした．

4.1 clasp の動作モードの選択

今回設計した muffin で用いる clasp には数種類の動作モードがあり，muffin で運用する上でどの動作モードが最も適しているのかを調査するための実験を行った．ベンチマークとして 2009 年の国際 CSP ソルバー競技会のベンチマーク問題から外延的制約を除いた問題 (1797 問) から，各シリーズごとに 3 問ずつ選択した 174 問を用いた．

これらの問題に対し，muffin を用いて，clasp2.1.3[†3]において標準で提供されている 3 つの動作モードに対し実験を行った: frumpy は clasp のデフォルトの動作モードで，汎用的な用途に用いられる．jumpy は探索のリスタートを頻繁に行い，充足可能な問題に強い．trendy は構造的な問題に対して効果的に働く．

ベンチマーク問題は XCSP で書かれており，Sugar で読み込めるため，muffin でも求解が可能である．

図 17 に実験結果のカクタスプロットを示す．カクタスプロットの横軸は解けた問題数を，縦軸は昇順に並び替えたソルバーの CPU 時間を示している．すなわち，この表はグラフが右にあるほど多くの問題を解き，下にあるほど高速に求解していることを示す．結果として，frumpy が 88 問，jumpy が 94 問，trendy が 93 問求解しており，また jumpy は他の動作モードと比較しても常にほぼ右下にある，また frumpy と trendy が解いた問題は全て jumpy でも求解ができたため，muffin の評価実験を jumpy で行うことにする．

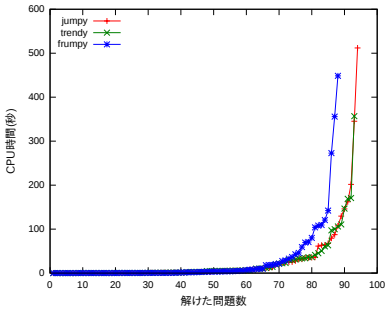

図 17　clasp の動作モードの比較

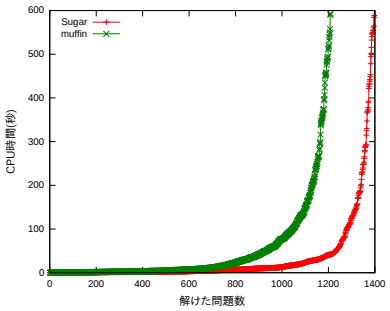

図 18　CSP 競技会ベンチマーク 1707 問の実験結果

4.2 部分実験の結果

CSP Solver Competition(2009) のベンチマーク問題から外延的制約を除いた全ての問題 1797 問を用いて評価実験を行った．これらの問題に対し，muffin と Sugar の 2 通りで実験を行った．Sugar は SAT 型 CSP ソルバーであり，SAT ソルバーを用いて求解を行う．今回は muffin との比較のために clasp を用いる．また，clasp の動作モードは前節の結果より，muffin, Sugar ともに jumpy に設定する．

実験結果の解けた問題数と CPU 時間を比較するために，図 18 にカクタスプロットを示す．解けた問題数については，muffin が 1209 問，Sugar が 1397 問であった．XCSP 問題は CSP ソルバーに向けて作られたベンチマークのため直接の比較は困難であるが，求解数について，muffin は Sugar の 87%程度の性能

†3　http://www.cs.uni-potsdam.de/clasp/

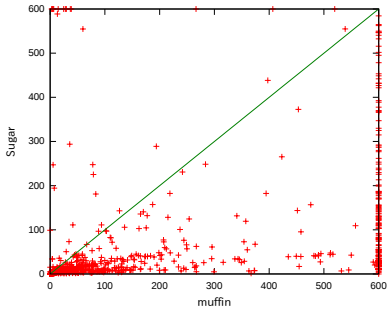

図 19　ソルバー求解時間の比較

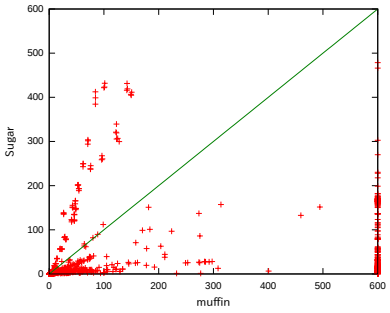

図 20　符号化時間の比較

となっている．

また，実験結果について詳しく Sugar との比較を行うために，図 19 にスキャッタープロットを示す．スキャッタープロットの横軸は muffin の CPU 時間を，縦軸は Sugar の CPU 時間を示している．スキャッタープロットの中心の直線より右下に点が集中しているため，Sugar の方が全体的に動作が速いが，左上の方にもいくつかまとまってプロットが存在しており，muffin の方が Sugar よりも動作が速い問題がいくつか存在する．

ASP が CSP より部分的に優れている点についてより詳しく考察する．問題をシリーズごとに分けた求解数と動作時間の平均を表 5 と表 5 に示す．

動作時間の平均については，解けなかった問題をタイムアウトの 600 秒に換算して算出している．これらの表を見ると，All Interval Series, Pseudo-Boolean (fpga のみ), Queen Attacking, Social Golfers に限っては muffin の方が優れていることが分かった．これらは変数 1 つあたりのドメインの範囲が小さな問題が多い．ドメインの範囲が小さな問題が得意な理由としては，ドメインが小さいと整数を扱う制約と記号的な制約のギャップが小さくなり，gringo の動作が高速になる点が考えられる．

一方，Cabinet, FAPP については 1 問も解けなかった．これらは変数のドメインがかなり欠けている問題であり，`remdom` の処理にかなりの時間がかかってしまったためだと考えられる．

次に，遅くなる原因についても考察する．図 20 に CSP から clasp で処理する直前の基礎化 ASP への変換にかかった時間のスキャッタープロットを示す．変

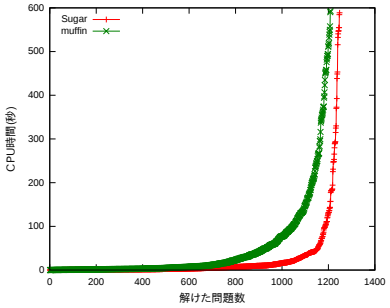

図 21　符号化でタイムアウトした問題を除く CSP 競技会ベンチマーク 1322 問の実験結果

換中にタイムアウト，もしくはメモリ不足になった問題数について調べると，Sugar は 10 問しかなかったのに対し，muffin は 475 問もあった．さらに，変換が正常に終了している問題に限定したカクタスプロットを図 21 に示す．図 21 は図 18 と比べると，大幅に差が縮まっていることが分かる．また，gringo の前までの変換は全てタイムアウトまでに終わっており，muffin の変換過程で最も動作時間がかかるのは gringo であることが確認されている．gringo が最も遅い原因は，CSP をより規模の大きな問題である ASP に変換しているためだと考えられる．gringo に処理させる ASP の記述を改善することで，大幅な性能上昇が見込める．

今後の課題としては，算術制約処理ライブラリの記述の改善が挙げられる．例えば 2 項以上の制約を 1 項少ない制約に変換するとき，現状では決められた順序で展開しているが，この時の順序を探索範囲が少なくなるように条件分岐で変更すると gringo で生成される制約の数が軽減される．

表 5　各シリーズ毎の平均 CPU 時間

Series	muffin	Sugar
2D Strip Packing (20)	457.45	**340.2**
All Interval Series (15)	**219.94**	253.68
BIBD (83)	106.89	**39.6**
BMC (15)	54.31	**15.78**
BQWH (20)	2.01	**1.53**
Cabinet (40)	600	**18.53**
Chessboard Coloration (15)	142.57	**130.2**
Costas Array (11)	171.46	**162.76**
Cumulative Job-Shop (10)	368.9	**366**
Domino (10)	107.58	**68.38**
FAPP (120)	600	**369.05**
Fischer (25)	454.68	**313.63**
Golomb Ruler (28)	290.6	**266.37**
Graph Coloring (141)	204.02	**197.18**
Haystacks (15)	493.9	**490.4**
Job-Shop (76)	166.02	**106.64**
Knights (10)	346.58	**260.4**
Langford (43)	255.89	**252.76**
Latin Square (10)	61.52	**61.13**
Magic Square (18)	410.71	**300.49**
Multi Knapsack (6)	269.49	**166.53**
NengFa (10)	47.44	**14.84**
Open-Shop (75)	124.17	**46.9**
Perfect Square Packing (74)	310.17	**183.83**
Pigeons (29)	124.44	**124.43**
Primes (76)	319.26	**236.87**
Pseudo-Boolean-GLB (100)	208.79	**180.15**
Pseudo-Boolean-fpga (15)	**25.84**	299.76
Pseudo-Boolean-others (248)	187.71	**140.66**
Quasigroup Existence (35)	104.27	**97.21**
Queen Attacking (10)	**368.44**	383.1
Queens (25)	270.32	**149.91**
RCPSP (78)	3.69	**1.7**
RLFAP (59)	133.29	**6.14**
Rader Surveillance (65)	4.7	**1.7**
Ramsey (16)	285.54	**270.6**
Schurr's Lemma (10)	122.33	**104.29**
Social Golfers (10)	**250.68**	251.06
Super-solutions (85)	254.14	**193.86**
Timetabling (46)	206.11	**111.94**

Series	muffin	Sugar
2D Strip Packing (20)	5	**10**
All Interval Series (15)	**10**	9
BIBD (83)	74	**80**
BMC (15)	15	15
BQWH (20)	20	20
Cabinet (40)	0	**40**
Chessboard Coloration (15)	12	12
Costas Array (11)	8	**9**
Cumulative Job-Shop (10)	4	4
Domino (10)	9	**10**
FAPP (120)	0	**64**
Fischer (25)	11	**16**
Golomb Ruler (28)	16	16
Graph Coloring (141)	98	**99**
Haystacks (15)	3	3
Job-Shop (76)	59	**66**
Knights (10)	5	**6**
Langford (43)	26	26
Latin Square (10)	9	9
Magic Square (18)	6	**10**
Multi Knapsack (6)	4	**5**
NengFa (10)	10	10
Open-Shop (75)	70	**71**
Perfect Square Packing (74)	39	**56**
Pigeons (29)	23	23
Primes (76)	41	**47**
Pseudo-Boolean-GLB (100)	**75**	73
Pseudo-Boolean-fpga (15)	**15**	8
Pseudo-Boolean-others (248)	178	**196**
Quasigroup Existence (35)	30	30
Queen Attacking (10)	4	4
Queens (25)	17	**20**
RCPSP (78)	78	78
RLFAP (59)	55	**59**
Rader Surveillance (65)	65	65
Ramsey (16)	9	**10**
Schurr's Lemma (10)	8	**9**
Social Golfers (10)	6	6
Super-solutions (85)	53	**62**
Timetabling (46)	39	**41**
Solved (1797)	1209	**1397**

5　まとめ

本論文では，ASP と CSP の利点を統合した CASP システムの設計方式について提案を行った．提案する CASP システムにおける言語の設計方針は次の 2 つである．(a) まず ASP の記述言語が拡張され，算術的な制約やグローバル制約を含む規模が大きな問題についても簡潔に記述可能にする．(b) いくつかの変数をまとめて 1 つのコレクション変数として扱えるようにする．また CASP システムの言語処理系の設計方針は次の 2 つである．(a) 算術的な制約やグローバル制約といった CSP の記述部分を分割し，ASP 外で正規化する．(b) 各機能をモジュール化し，符号化ライブラリやソルバーを容易に変更可能にする．

また提案した CASP システム設計方式の実装としてを muffin を開発した．muffin は近年成功を収めて

いる順序符号化を用いて開発した符号化ライブラリと，最新の ASP ソルバーを用いて実装した．

実装した CASP システム muffin については CSP 競技会で使われた 1797 問の CSP を使って部分実験を行い CSP 問題に対する基本性能についても評価を行った．muffin は ASP, CSP, CASP の問題を入力とすることができるが，今回 CSP をベンチマークとして用いた理由は CASP がまだ研究途上であり広範なベンチマークの整備がされていないこと，バックエンドの ASP ソルバー gringo と clasp は ASP において既に高い性能が示されていることである．muffin の CSP に対する今後の課題としては，作成した符号化ライブラリの記述の改良を行うことによる符号化にかかる時間の短縮を行うことが考えられる．またベンチマーク整備も CASP 研究分野の課題として挙げられる．

参 考 文 献

[1] Balduccini, M. and Lierler, Y.: ASP-Based Problem Solving with Cutting-Edge Tools, *ICLP 2011 Workshop on Answer Set Programming and Other Computing Paradigms (ASPOCP 2011)*, 2011, pp. 14–28.

[2] Balduccini, M. and Lierler, Y.: Practical and methodological aspects of the use of cutting-edge ASP tools, *Practical Aspects of Declarative Languages*, Springer, 2012, pp. 78–92.

[3] Banbara, M., Gebser, M., Inoue, K., Schaub, T., Soh, T., Tamura, N., and Weise, M.: Aspartame: Solving Constraint Satisfaction Problems with Answer Set Programming, *CoRR*, Vol. abs/1312.6113(2013).

[4] Banbara, M., Matsunaka, H., Tamura, N., and Inoue, K.: Generating Combinatorial Test Cases by Efficient SAT Encodings Suitable for CDCL SAT Solvers, *Proceedings of the 17th International Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR-17)*, 2010, pp. 112–126.

[5] Banbara, M., Soh, T., Tamura, N., Inoue, K., and Schaub, T.: Answer set programming as a modeling language for course timetabling, *TPLP*, Vol. 13, No. 4-5(2013), pp. 783–798.

[6] Banbara, M., Tamura, N., and Inoue, K.: Generating Event-Sequence Test Cases by Answer Set Programming with the Incidence Matrix, *Technical Communications of the 28th International Conference on Logic Programming (ICLP 2012)*, Dovier, A. and Costa, V. S.(eds.), LIPIcs, Vol. 17, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012, pp. 86–97.

[7] Barták, R.: On-line Guide to Constraint Programming, `http://kti.ms.mff.cuni.cz/~bartak/constraints/`, 1998.

[8] Béjar, R., Hähnle, R., and Manyà, F.: A Modular Reduction of Regular Logic to Classical Logic, *Proceedings of the 31st IEEE International Symposium on Multiple-Valued Logic (ISMVL 2001)*, 2001, pp. 221–226.

[9] Bordeaux, L., Hamadi, Y., and Zhang, L.: Propositional Satisfiability and Constraint Programming: A Comparative Survey, *ACM Computing Surveys*, Vol. 38, No. 4(2006).

[10] de Kleer, J.: A Comparison of ATMS and CSP Techniques, *Proceedings of the 11th International Joint Conference on Artificial Intelligence (IJCAI 1989)*, 1989, pp. 290–296.

[11] Delgrande, J. and Faber, W.(eds.): *Proceedings of the Eleventh International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'11)*, Lecture Notes in Artificial Intelligence, Vol. 6645, Springer-Verlag, 2011.

[12] Drescher, C. and Walsh, T.: A translational approach to constraint answer set solving, *TPLP*, Vol. 10, No. 4-6(2010), pp. 465–480.

[13] Gebser, M., Kaminski, R., Kaufmann, B., and Schaub, T.: *Answer Set Solving in Practice*, Synthesis Lectures on Artificial Intelligence and Machine Learning, Morgan and Claypool Publishers, 2012.

[14] Gebser, M., Kaminski, R., König, A., and Schaub, T.: Advances in *gringo* Series 3, In Delgrande and Faber [11], pp. 345–351.

[15] Gebser, M., Hinrichs, H., Schaub, T., and Thiele, S.: xpanda: A (simple) preprocessor for adding multi-valued propositions to ASP, *Proceedings of the 23rd Workshop on (Constraint) Logic Programming 2009*, Universitätsverlag Potsdam, 2010, pp. 51.

[16] Gebser, M., Kaufmann, B., Neumann, A., and Schaub, T.: Conflict-Driven Answer Set Solving, *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI 2007)*, 2007, pp. 386–.

[17] Gebser, M., Liu, L., Namasivayam, G., Neumann, A., Schaub, T., and Truszczynski, M.: The First Answer Set Programming System Competition, *LPNMR*, 2007, pp. 3–17.

[18] Gelder, A. V.: Another Look at Graph Coloring via Propositional Satisfiability, *Discrete Applied Mathematics*, Vol. 156, No. 2(2008), pp. 230–243.

[19] Gent, I. P.: Arc Consistency in SAT, *Proceedings of the 15th European Conference on Artificial Intelligence (ECAI 2002)*, 2002, pp. 121–125.

[20] Gent, I. P., Miguel, I., and Nightingale, P.: Generalised Arc Consistency for the AllDifferent Constraint: An Empirical Survey, *Artificial Intelligence*, Vol. 172, No. 18(2008), pp. 1973–2000.

[21] 井上克巳, 坂間千秋: 論理プログラミングから解集合プログラミングへ, コンピュータソフトウェア, Vol. 25, No. 3(2008), pp. 20–32.

[22] Iwama, K. and Miyazaki, S.: SAT-Variable Complexity of Hard Combinatorial Problems, *Proceedings of the IFIP 13th World Computer Congress*, 1994, pp. 253–258.

[23] Kasif, S.: On the Parallel Complexity of Discrete Relaxation in Constraint Satisfaction Networks, *Artificial Intelligence*, Vol. 45, No. 3(1990), pp. 275–286.

[24] 越村三幸, 鍋島英知, 藤田博, 長谷川隆三: SAT 変換による未解決ジョブショップスケジューリング問題への挑戦, スケジューリング・シンポジウム *2009* 講演論文集, 2009, pp. 209–213.

[25] Koshimura, M., Nabeshima, H., Fujita, H., and Hasegawa, R.: Solving Open Job-Shop Scheduling Problems by SAT Encoding, *IEICE TRANSACTIONS on Information and Systems*, Vol. E93-D, No. 8(2010), pp. 2316–2318.

[26] Lierler, Y.: Relating constraint answer set programming languages and algorithms, *Artif. Intell.*, Vol. 207(2014), pp. 1–22.

[27] Lierler, Y. and Maratea, M.: Cmodels-2: SAT-based answer set solver enhanced to non-tight programs, *Logic Programming and Nonmonotonic Reasoning*, Springer, 2004, pp. 346–350.

[28] Liu, G., Janhunen, T., and Niemelä, I.: Answer Set Programming via Mixed Integer Programming, *KR*, 2012.

[29] Niemelä, I.: Answer Set Programming: A Declarative Approach to Solving Search Problems, *JELIA*, Fisher, M., van der Hoek, W., Konev, B., and Lisitsa, A.(eds.), Lecture Notes in Computer Science, Vol. 4160, Springer, 2006, pp. 15–18.

[30] Ostrowski, M. and Schaub, T.: ASP modulo CSP: The clingcon system, *TPLP*, Vol. 12, No. 4-5(2012), pp. 485–503.

[31] Selman, B., Levesque, H. J., and Mitchell, D. G.: A New Method for Solving Hard Satisfiability Problems, *Proceedings of the 10th National Conference on Artificial Intelligence (AAAI 1992)*, 1992, pp. 440–446.

[32] Soh, T., Inoue, K., Tamura, N., Banbara, M., and Nabeshima, H.: A SAT-based Method for Solving the Two-dimensional Strip Packing Problem, *Fundamenta Informaticae*, Vol. 102, No. 3–4(2010), pp. 467–487.

[33] Tamura, N. and Banbara, M.: Sugar: a CSP to SAT Translator Based on Order Encoding, *Proceedings of the 2nd International CSP Solver Competition*, 2008, pp. 65–69.

[34] Tamura, N., Taga, A., Kitagawa, S., and Banbara, M.: Compiling Finite Linear CSP into SAT, *Proceedings of the 12th International Joint Conference on Principles and Practice of Constraint Programming (CP 2006), LNCS 4204*, 2006, pp. 590–603.

[35] Tamura, N., Taga, A., Kitagawa, S., and Banbara, M.: Compiling Finite Linear CSP into SAT, *Constraints*, Vol. 14, No. 2(2009), pp. 254–272.

[36] 田村直之, 丹生智也, 番原睦則: 制約最適化問題と SAT 符号化, 人工知能学会誌, Vol. 25, No. 1(2010), pp. 77–85.

[37] Tanjo, T., Tamura, N., and Banbara, M.: A Compact and Efficient SAT-Encoding of Finite Domain CSP, Springer, 2011, pp. 375–376.

[38] Walsh, T.: SAT v CSP, *Proceedings of the 6th International Conference on Principles and Practice of Constraint Programming (CP 2000)*, 2000, pp. 441–456.