情報基礎特論: 制約プログラミング入門

田村直之

神戸大学

2011 年 6 月 6 日
2011 年 8 月 22 日 修正
2012 年 6 月 24 日 修正
2014 年 5 月 19 日 修正

内容

1. **制約充足問題**
2. **制約充足アルゴリズム**
3. Cream **の紹介**
   - Java **上の制約プログラミング用クラス・ライブラリ**
   - OpenOffice.org Calc **インターフェイスのデモ**
4. Sugar **の紹介**
   - SAT **型制約ソルバー**
   - Scala **インターフェイスのデモ**

制約充足問題

制約充足問題 (CSP)

制約充足問題 (CSP; Constraint Satisfaction Problem) **とは，与えられた制約を満たす解が存在するかを判定する問題である．**

制約充足問題

制約充足問題は，以下を満たす組 (X, D, C) **として与えられる．**

- X: **変数** (variable) **の有限集合**
- D: **各変数** $x \in X$ **の取り得る値の集合である**ドメイン (domain) **を定義する関数 (例:** $D(x) = \{1, 2, 3\}$**)．通常，** $D(x)$ **は整数の有限集合とすることが多い．**
- C: X **上の制約** (constraint) **の有限集合**

- X **上の関数** v **が，各変数** x **について** $v(x) \in D(x)$ **を満たす時，** v **を値割当て** (assignment) **という．**
- **ある値割当て** v **が，** C **中の全制約を充足** (satisfy) **する時，その** CSP **は充足可能** (satisfiable) **と呼ばれ，** v **を解という．充足可能でない場合，充足不能** (unsatisfiable) **と呼ばれる．**

制約充足問題の例

4 クイーン配置問題

$$
\begin{aligned}
X &= \{q_0, q_1, q_2, q_3\} \\
D(q_i) &= \{1, 2, 3, 4\} \quad (i = 0, 1, 2, 3) \\
C &= \{q_i \neq q_j \mid i < j\} \\
&\cup \{q_i + i \neq q_j + j \mid i < j\} \\
&\cup \{q_i - i \neq q_j - j \mid i < j\}
\end{aligned}
$$

	1	2	3	4
q_0		Q		
q_1				Q
q_2	Q			
q_3			Q	

- $v = \{q_0 = 2,\ q_1 = 4,\ q_2 = 1,\ q_3 = 3\}$ **は上の** CSP **の解．**
  - q_i **は，すべて互いに異なっている．**
  - $q_i + i$ **は，すべて互いに異なっている．**
  - $q_i - i$ **は，すべて互いに異なっている．**
- **一般には** n **クイーン配置問題と呼ばれる．**

制約ソルバー

制約充足問題の解を探索するプログラムを制約ソルバー (Constraint solver) **あるいは** CSP **ソルバー** (CSP solver) **という．制約ソルバーにおける制約充足問題の記述方法には以下の** 2 **種類がある．**

- **既存のプログラミング言語の枠内で制約充足問題を記述するもの (制約プログラミング**, constraint programming)
  - **論理プログラミング**: SICStus Prolog, B-Prolog
  - C++: IBM ILOG CP (**商用**)
  - Java: Choco, Cream
- **なんらかの制約モデリング言語を用いるもの**
  - Gecode, Sugar

- **ほとんどの制約ソルバーは，制約充足問題だけでなく制約最適化問題** (COP; Constaint Optimization Problem) **なども解くことができる．**

CSP および COP の応用

- **スケジューリング問題**
  - **J リーグの日程調整** (IBM ILOG CP)
- **人員配置計画**
- **輸送計画**
- **パッキング問題**
- **時間割作成**
- **パズル・ソルバー**

問題集

- CSP Lib ▸ Web
- OR Library ▸ Web
- XCSP benchmarks ▸ Web

制約充足アルゴリズム

生成検査法 (Generate and Test)

for ($q_0 \in D(q_0)$)
  for ($q_1 \in D(q_1)$)
    for ($q_2 \in D(q_2)$)
      for ($q_3 \in D(q_3)$)
        if (q_0, q_1, q_2, q_3 **が全制約を充足する**)
          q_0, q_1, q_2, q_3 **は解**

- **アルゴリズムは単純だが，非常に遅い．**

- $q_0 = 1,\ q_1 = 1,\ q_2 = 1,\ q_3 = 1$ **は制約を充足しない．**

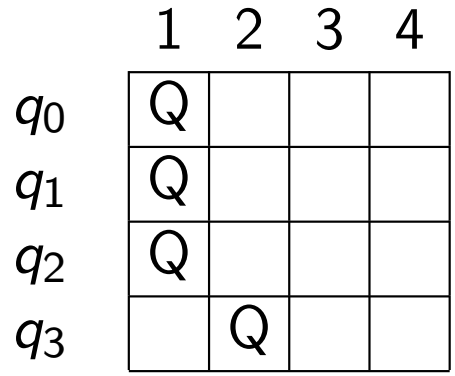
	1	2	3	4
q_0	Q			
q_1	Q			
q_2	Q			
q_3	Q			

生成検査法 (Generate and Test)

for ($q_0 \in D(q_0)$)
  for ($q_1 \in D(q_1)$)
    for ($q_2 \in D(q_2)$)
      for ($q_3 \in D(q_3)$)
        if (q_0, q_1, q_2, q_3 **が全制約を充足する**)
          q_0, q_1, q_2, q_3 **は解**

- **アルゴリズムは単純だが，非常に遅い．**

- $q_0 = 1,\ q_1 = 1,\ q_2 = 1,\ q_3 = 1$ **は制約を充足しない．**
- $q_0 = 1,\ q_1 = 1,\ q_2 = 1,\ q_3 = 2$ **も制約を充足しない．**



バックトラック法 (Backtracking)

```
for (q0 ∈ D(q0))
  for (q1 ∈ D(q1))
    if (q1 が q0 との制約を充足しない) continue
    for (q2 ∈ D(q2))
      if (q2 が q0, q1 との制約を充足しない) continue
      for (q3 ∈ D(q3))
        if (q3 が q0, q1, q2 との制約を充足しない) continue
        q0, q1, q2, q3 は解
```

- $q_0 = 1$ の時，$q_1 = 1$ および $q_1 = 2$ は制約を充足しない．

	1	2	3	4
q_0	Q			
q_1	×	×		
q_2				
q_3				

バックトラック法 (Backtracking)

```
for (q0 ∈ D(q0))
  for (q1 ∈ D(q1))
    if (q1 が q0 との制約を充足しない) continue
    for (q2 ∈ D(q2))
      if (q2 が q0, q1 との制約を充足しない) continue
      for (q3 ∈ D(q3))
        if (q3 が q0, q1, q2 との制約を充足しない) continue
        q0, q1, q2, q3 は解
```

- $q_0 = 1$ の時，$q_1 = 1$ および $q_1 = 2$ は制約を充足しない．
- $q_0 = 1$, $q_1 = 3$ の時，q_2 のどの値も制約を充足しない．

	1	2	3	4
q_0	Q			
q_1	x	x	Q	
q_2	x	x	x	x
q_3				

制約伝播 (Constraint Propagation)

- **制約伝播を用いると多くの無駄な探索を省くことができる.**
- **制約伝播には以下のような方法がある.**
  - **フォワード・チェック法** (FC; Forward Checking)
  - **アーク整合性維持法** (MAC; Maintaining Arc Consistency)
- **いずれも，各制約に対して**アーク整合性アルゴリズム (Arc Consistency Algorithm) **を適用し，制約に違反** (conflict) **する値を削除する方法を用いる.**

アーク整合性 (Arc Consistency)

アーク整合性

変数 x, y 間の制約 $C(x, y)$ が以下を満たす時，アーク整合 (arc consistent) **であるという．**

- **x のどの値についても制約を満たす y の値が存在する．**

- **$D(x) = D(y) = \{1, 2, 3\}$ の時，制約 $x < y$ はアーク整合でない．**
  - **$x = 1$ の場合，$y = 2, 3$ が $x < y$ を満たし，**
  - **$x = 2$ の場合，$y = 3$ が $x < y$ を満たすが，**
  - **$x = 3$ の場合，$x < y$ を満たす y が存在しない．**

アーク整合性 (Arc Consistency)

アーク整合性

変数 x, y 間の制約 $C(x, y)$ が以下を満たす時，アーク整合 (arc consistent) **であるという．**

- **x のどの値についても制約を満たす y の値が存在する．**

- **$D(x) = D(y) = \{1, 2, 3\}$ の時，制約 $x < y$ はアーク整合でない．**
  - **$x = 1$ の場合，$y = 2, 3$ が $x < y$ を満たし，**
  - **$x = 2$ の場合，$y = 3$ が $x < y$ を満たすが，**
  - **$x = 3$ の場合，$x < y$ を満たす y が存在しない．**
- **$D(x)$ から 3 を削除すれば，アーク整合になる．**

アーク整合性 (Arc Consistency)

アーク整合性

変数 x, y 間の制約 $C(x, y)$ が以下を満たす時，アーク整合 (arc consistent) **であるという．**

- **x のどの値についても制約を満たす y の値が存在する．**

- **$D(x) = D(y) = \{1, 2, 3\}$ の時，制約 $x < y$ はアーク整合でない．**
  - **$x = 1$ の場合，$y = 2, 3$ が $x < y$ を満たし，**
  - **$x = 2$ の場合，$y = 3$ が $x < y$ を満たすが，**
  - **$x = 3$ の場合，$x < y$ を満たす y が存在しない．**
- **$D(x)$ から 3 を削除すれば，アーク整合になる．**
- **アーク整合性には向きがある点に注意する．この場合，y からも調べる必要がある．**
  - **$y = 1$ の場合，$x < y$ を満たす x は存在しない．**
- **$D(y)$ から 1 を削除すれば，逆向きもアーク整合になる．**

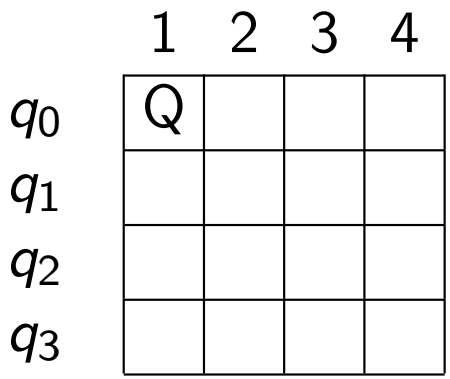
フォワード・チェック法 (Forward Checking)

for ($q_0 \in D(q_0)$)
  q_0 と q_1, q_2, q_3 をアーク整合にする
  if (いずれかの変数のドメインが空) backtrack; continue
  for ($q_1 \in D(q_1)$)
    q_1 と q_2, q_3 をアーク整合にする
    if (いずれかの変数のドメインが空) backtrack; continue
    for ($q_2 \in D(q_2)$)
      q_2 と q_3 をアーク整合にする
      if (いずれかの変数のドメインが空) backtrack; continue
      for ($q_3 \in D(q_3)$)
        q_0, q_1, q_2, q_3 は解

- backtrack ではバックトラック処理を行う．すなわちアーク整合処理で削除した値を元に戻す．

フォワード・チェック法 (Forward Checking)

- $q_0 = 1$ とする.



フォワード・チェック法 (Forward Checking)

- $q_0 = 1$ とする.
- q_0 と q_1, q_2, q_3 をアーク整合にする.

	1	2	3	4
q_0	Q			
q_1	x	x		
q_2	x		x	
q_3	x			x

フォワード・チェック法 (Forward Checking)

- $q_0 = 1$ とする.
- q_0 と q_1, q_2, q_3 をアーク整合にする.
- $q_1 = 3$ とする.

	1	2	3	4
q_0	Q			
q_1	x	x	Q	
q_2	x		x	
q_3	x			x

フォワード・チェック法 (Forward Checking)

- $q_0 = 1$ とする.
- q_0 と q_1, q_2, q_3 をアーク整合にする.
- $q_1 = 3$ とする.
- q_1 と q_2, q_3 をアーク整合にする.

	1	2	3	4
q_0	Q			
q_1	x	x	Q	
q_2	x	x	x	x
q_3	x		x	x

フォワード・チェック法 (Forward Checking)

- $q_0 = 1$ とする．
- q_0 と q_1, q_2, q_3 をアーク整合にする．
- $q_1 = 3$ とする．
- q_1 と q_2, q_3 をアーク整合にする．
- バックトラックし，$q_1 = 4$ とする．

	1	2	3	4
q_0	Q			
q_1	x	x		Q
q_2	x		x	
q_3	x			x

フォワード・チェック法 (Forward Checking)

- $q_0 = 1$ とする．
- q_0 と q_1, q_2, q_3 をアーク整合にする．
- $q_1 = 3$ とする．
- q_1 と q_2, q_3 をアーク整合にする．
- バックトラックし，$q_1 = 4$ とする．
- q_1 と q_2, q_3 をアーク整合にする．

	1	2	3	4
q_0	Q			
q_1	x	x		Q
q_2	x		x	x
q_3	x	x		x

フォワード・チェック法 (Forward Checking)

- $q_0 = 1$ とする．
- q_0 と q_1, q_2, q_3 をアーク整合にする．
- $q_1 = 3$ とする．
- q_1 と q_2, q_3 をアーク整合にする．
- バックトラックし，$q_1 = 4$ とする．
- q_1 と q_2, q_3 をアーク整合にする．
- $q_2 = 2$ とし，q_2 と q_3 をアーク整合にする．

	1	2	3	4
q_0	Q			
q_1	x	x		Q
q_2	x	Q	x	x
q_3	x	x	x	x

フォワード・チェック法 (Forward Checking)

- $q_0 = 1$ とする．
- q_0 と q_1, q_2, q_3 をアーク整合にする．
- $q_1 = 3$ とする．
- q_1 と q_2, q_3 をアーク整合にする．
- バックトラックし，$q_1 = 4$ とする．
- q_1 と q_2, q_3 をアーク整合にする．
- $q_2 = 2$ とし，q_2 と q_3 をアーク整合にする．
- バックトラックし，$q_0 = 2$ とする．

	1	2	3	4
q_0		Q		
q_1	x	x	x	
q_2		x		x
q_3		x		

フォワード・チェック法 (Forward Checking)

- $q_0 = 1$ とする．
- q_0 と q_1, q_2, q_3 をアーク整合にする．
- $q_1 = 3$ とする．
- q_1 と q_2, q_3 をアーク整合にする．
- バックトラックし，$q_1 = 4$ とする．
- q_1 と q_2, q_3 をアーク整合にする．
- $q_2 = 2$ とし，q_2 と q_3 をアーク整合にする．
- バックトラックし，$q_0 = 2$ とする．
- $q_1 = 4$ とし，q_2, q_3 とアーク整合にする．

	1	2	3	4
q_0		Q		
q_1	x	x	x	Q
q_2		x	x	x
q_3		x		x

フォワード・チェック法 (Forward Checking)

- $q_0 = 1$ とする．
- q_0 と q_1, q_2, q_3 をアーク整合にする．
- $q_1 = 3$ とする．
- q_1 と q_2, q_3 をアーク整合にする．
- バックトラックし，$q_1 = 4$ とする．
- q_1 と q_2, q_3 をアーク整合にする．
- $q_2 = 2$ とし，q_2 と q_3 をアーク整合にする．
- バックトラックし，$q_0 = 2$ とする．
- $q_1 = 4$ とし，q_2, q_3 とアーク整合にする．
- $q_2 = 1$ とし，q_3 とアーク整合にする．

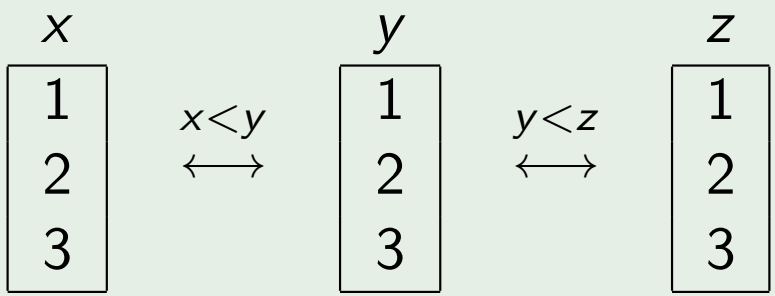
	1	2	3	4
q_0		Q		
q_1	x	x	x	Q
q_2	Q	x	x	x
q_3	x	x		x

フォワード・チェック法 (Forward Checking)

- $q_0 = 1$ とする．
- q_0 と q_1, q_2, q_3 をアーク整合にする．
- $q_1 = 3$ とする．
- q_1 と q_2, q_3 をアーク整合にする．
- バックトラックし，$q_1 = 4$ とする．
- q_1 と q_2, q_3 をアーク整合にする．
- $q_2 = 2$ とし，q_2 と q_3 をアーク整合にする．
- バックトラックし，$q_0 = 2$ とする．
- $q_1 = 4$ とし，q_2, q_3 とアーク整合にする．
- $q_2 = 1$ とし，q_3 とアーク整合にする．
- $q_3 = 3$ とする．

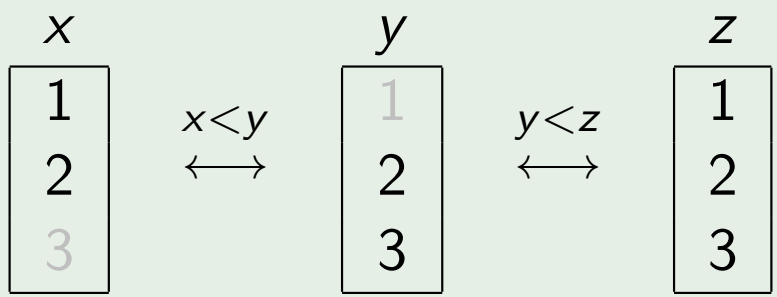
	1	2	3	4
q_0		Q		
q_1	x	x	x	Q
q_2	Q	x	x	x
q_3	x	x	Q	x

アーク整合アルゴリズム



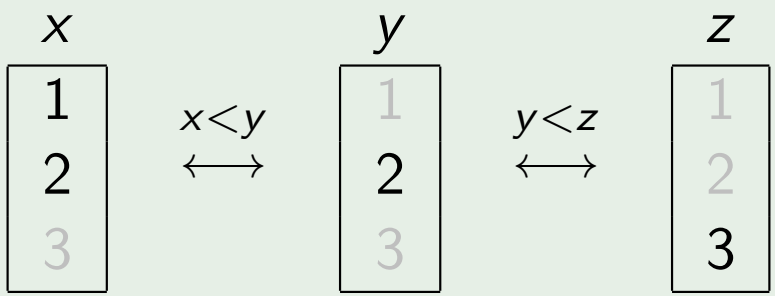
- アーク整合で，得られた情報を伝播する．
- ある制約がアーク整合だったとしても，他の制約をアーク整合にすることにより，アーク整合でなくなることがある．

アーク整合アルゴリズム



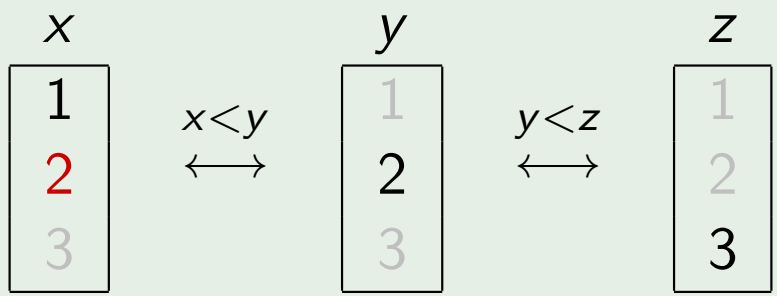
- アーク整合で，得られた情報を伝播する．
  - $x < y$ をアーク整合にする．
- ある制約がアーク整合だったとしても，他の制約をアーク整合にすることにより，アーク整合でなくなることがある．

アーク整合アルゴリズム



- アーク整合で，得られた情報を伝播する．
  - $x < y$ をアーク整合にする．
  - $y < z$ をアーク整合にする．
- ある制約がアーク整合だったとしても，他の制約をアーク整合にすることにより，アーク整合でなくなることがある．

アーク整合アルゴリズム



- アーク整合で，得られた情報を伝播する．
  - $x < y$ をアーク整合にする．
  - $y < z$ をアーク整合にする．
- ある制約がアーク整合だったとしても，他の制約をアーク整合にすることにより，アーク整合でなくなることがある．
  - $x < y$ について再びアーク整合処理が必要．

AC3 アルゴリズム

AC3 アルゴリズム

```
function AC3(x) {
  Q = x を含む制約の集合
  while (Q が空でない) {
    Q から一つ制約 c を取り除く
    c に対してアーク整合処理を行う
    if (アーク整合処理で変数 y から値が削除された) {
      if (y のドメインが空)
        return false
      Q に y を含む制約を追加する
    }
  }
  return true
}
```

- 他に AC4, AC7 などのアルゴリズムが考案されている.

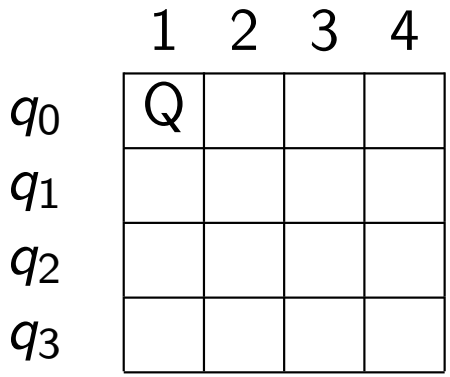
アーク整合性維持法 (MAC; Maintaining Arc Consistency)

```
for (q0 ∈ D(q0))
  if (not AC3(q0)) backtrack; continue
  for (q1 ∈ D(q1))
    if (not AC3(q1)) backtrack; continue
    for (q2 ∈ D(q2))
      if (not AC3(q2)) backtrack; continue
      for (q3 ∈ D(q3))
        q0, q1, q2, q3 は解
```

- backtrack **ではバックトラック処理を行う．すなわちアーク整合処理で削除した値を元に戻す．**

アーク整合性維持法 (MAC; Maintaining Arc Consistency)

- $q_0 = 1$ とする.



アーク整合性維持法 (MAC; Maintaining Arc Consistency)

- $q_0 = 1$ とする.
- q_0 と q_1, q_2, q_3 をアーク整合にする.

	1	2	3	4
q_0	Q			
q_1	x	x		
q_2	x		x	
q_3	x			x

アーク整合性維持法 (MAC; Maintaining Arc Consistency)

- $q_0 = 1$ とする.
- q_0 と q_1, q_2, q_3 をアーク整合にする.
- q_1 と q_2 をアーク整合にする.

	1	2	3	4
q_0	Q			
q_1	x	x	x	
q_2	x		x	x
q_3	x			x

アーク整合性維持法 (MAC; Maintaining Arc Consistency)

- $q_0 = 1$ とする.
- q_0 と q_1, q_2, q_3 をアーク整合にする.
- q_1 と q_2 をアーク整合にする.
- q_1 と q_3 をアーク整合にする.

	1	2	3	4
q_0	Q			
q_1	x	x	x	
q_2	x		x	x
q_3	x	x		x

アーク整合性維持法 (MAC; Maintaining Arc Consistency)

- $q_0 = 1$ とする.
- q_0 と q_1, q_2, q_3 をアーク整合にする.
- q_1 と q_2 をアーク整合にする.
- q_1 と q_3 をアーク整合にする.
- q_2 と q_3 をアーク整合にする.

	1	2	3	4
q_0	Q			
q_1	×	×	×	
q_2	×	×	×	×
q_3	×	×	×	×

アーク整合性維持法 (MAC; Maintaining Arc Consistency)

- $q_0 = 1$ とする．
- q_0 と q_1，q_2，q_3 をアーク整合にする．
- q_1 と q_2 をアーク整合にする．
- q_1 と q_3 をアーク整合にする．
- q_2 と q_3 をアーク整合にする．
- バックトラックし，$q_0 = 2$ とする．

	1	2	3	4
q_0		Q		
q_1				
q_2				
q_3				

アーク整合性維持法 (MAC; Maintaining Arc Consistency)

- $q_0 = 1$ とする.
- q_0 と q_1, q_2, q_3 をアーク整合にする.
- q_1 と q_2 をアーク整合にする.
- q_1 と q_3 をアーク整合にする.
- q_2 と q_3 をアーク整合にする.
- バックトラックし，$q_0 = 2$ とする.
- q_0 と q_1, q_2, q_3 をアーク整合にする.

	1	2	3	4
q_0		Q		
q_1	×	×	×	
q_2		×		×
q_3		×		

アーク整合性維持法 (MAC; Maintaining Arc Consistency)

- $q_0 = 1$ とする．
- q_0 と q_1，q_2，q_3 をアーク整合にする．
- q_1 と q_2 をアーク整合にする．
- q_1 と q_3 をアーク整合にする．
- q_2 と q_3 をアーク整合にする．
- バックトラックし，$q_0 = 2$ とする．
- q_0 と q_1，q_2，q_3 をアーク整合にする．
- q_1 と q_2 をアーク整合にする．

	1	2	3	4
q_0		Q		
q_1	×	×	×	
q_2		×	×	×
q_3		×		

アーク整合性維持法 (MAC; Maintaining Arc Consistency)

- $q_0 = 1$ とする．
- q_0 と q_1，q_2，q_3 をアーク整合にする．
- q_1 と q_2 をアーク整合にする．
- q_1 と q_3 をアーク整合にする．
- q_2 と q_3 をアーク整合にする．
- バックトラックし，$q_0 = 2$ とする．
- q_0 と q_1，q_2，q_3 をアーク整合にする．
- q_1 と q_2 をアーク整合にする．
- q_1 と q_3 をアーク整合にする．

	1	2	3	4
q_0		Q		
q_1	x	x	x	
q_2		x	x	x
q_3		x		x

アーク整合性維持法 (MAC; Maintaining Arc Consistency)

- $q_0 = 1$ とする.
- q_0 と q_1, q_2, q_3 をアーク整合にする.
- q_1 と q_2 をアーク整合にする.
- q_1 と q_3 をアーク整合にする.
- q_2 と q_3 をアーク整合にする.
- バックトラックし，$q_0 = 2$ とする.
- q_0 と q_1, q_2, q_3 をアーク整合にする.
- q_1 と q_2 をアーク整合にする.
- q_1 と q_3 をアーク整合にする.
- q_2 と q_3 をアーク整合にする.

	1	2	3	4
q_0		Q		
q_1	×	×	×	
q_2		×	×	×
q_3	×	×		×

アーク整合性維持法 (MAC; Maintaining Arc Consistency)

- $q_0 = 1$ とする.
- q_0 と q_1, q_2, q_3 をアーク整合にする.
- q_1 と q_2 をアーク整合にする.
- q_1 と q_3 をアーク整合にする.
- q_2 と q_3 をアーク整合にする.
- バックトラックし，$q_0 = 2$ とする.
- q_0 と q_1, q_2, q_3 をアーク整合にする.
- q_1 と q_2 をアーク整合にする.
- q_1 と q_3 をアーク整合にする.
- q_2 と q_3 をアーク整合にする.
- $q_1 = 4, q_2 = 1, q_3 = 3$ とする.

	1	2	3	4
q_0		Q		
q_1	x	x	x	Q
q_2	Q	x	x	x
q_3	x	x	Q	x

その他

- **変数選択順序も性能に大きく影響する．**
  - **たとえば，ドメインサイズの小さいものから選ぶ．**
  - **早目に失敗するほうが良い** (first-fail principle)**．**
- **アーク整合性は二変数制約に関するものだが，** n **変数制約にも一般化できる** (General Arc Consistency)**．**
- **厳密なアーク整合性処理のためには，各変数のドメインを厳密に表現する必要があるが，それにはコストがかかる．そのため，ドメインをその上下限のみで表すことも多い．**
  - **上下限のみの伝播処理は，範囲伝播** (Bounds Propagation) **と呼ばれる．**
  - **ドメインサイズが小さくなった時点で，ビット集合等による厳密な表現に切替える．**
- alldifferent **等，グローバル制約** (global constraints) **と呼ばれるものについては，優れた整合性アルゴリズムが考案されている．**
  - alldifferent **については，**2 **部グラフのマッチングアルゴリズムを用いる．**

CSP のオンライン資料

- Roman Barták: On-line Guide to Constraint Programming, http://kti.ms.mff.cuni.cz/~bartak/constraints/ ▸ Web
- Norvig and Ruseel: Constraint Satisfaction Problems, http://aima.cs.berkeley.edu/newchap05.pdf ▸ Web
- Kim Marriott and Peter J. Stuckey: Programming with Constraints: an Introduction, http://www.cs.mu.oz.au/~pjs/book/course.html ▸ Web
- Gert Smolka: Constraint Programming, http://www.ps.uni-sb.de/courses/cp-ss07/ ▸ Web
- Christian Schulte: Constraint Programming, http://www.ict.kth.se/courses/ID2204/ ▸ Web

Java 上の
制約プログラミング用クラス・ライブラリ
Cream

Cream の概要

Cream の概要

- **整数ドメインに対する制約プログラミング用クラス・ライブラリ**
- **すべて** Java
- LGPL **で公開**
  - http://bach.istc.kobe-u.ac.jp/cream/ ▸ Web
- AC3 **による制約伝播**
- Simulated Annealing, Taboo Search **等のローカルサーチ**
- **マルチスレッドでの並行探索**
- OpenOffice.org Calc **インターフェイス**

鶴亀算 (1)

鶴亀算の問題

鶴と亀が合計で 7 匹，足の本数の合計は 20 本です．鶴と亀はそれぞれ何匹でしょうか．

$$x \in \mathbf{Z} \qquad y \in \mathbf{Z}$$
$$x \geq 0 \qquad y \geq 0$$
$$x + y = 7 \qquad 2x + 4y = 20$$

鶴亀算 (2)

制約ネットワークを生成

```
Network net = new Network();
```

変数の生成

```
IntVariable x = new IntVariable(net);
IntVariable y = new IntVariable(net);
```

制約の記述

```
x.ge(0);
y.ge(0);
x.add(y).equals(7);
x.multiply(2).add(y.multiply(4)).equals(20);
```

鶴亀算 (3)

制約ソルバーを生成

```
Solver solver = new DefaultSolver(net);
```

解の探索

```
Solution solution = solver.findFirst();
```

値の取り出し

```
int xv = solution.getIntValue(x);
int yv = solution.getIntValue(y);
```

鶴亀算 (4)

```
import jp.ac.kobe_u.cs.cream.*;

public class FirstStep {
  public static void main(String args[]) {
    Network net = new Network();
    IntVariable x = new IntVariable(net);
    IntVariable y = new IntVariable(net);
    x.ge(0);
    y.ge(0);
    x.add(y).equals(7);
    x.multiply(2).add(y.multiply(4)).equals(20);
    Solver solver = new DefaultSolver(net);
    Solution solution = solver.findFirst();
    int xv = solution.getIntValue(x);
    int yv = solution.getIntValue(y);
    System.out.println("x = " + xv + ", y = " + yv);
  }
}
```

n クイーン配置問題 (1)

n クイーン配置問題

$n \times n$ のチェス盤に，n 個のクイーンを互いに取られないように配置しなさい．

$$
\begin{aligned}
X &= \{q_0, q_1, \ldots, q_{n-1}\} \\
D(q_i) &= \{1, 2, \ldots, n\} \quad (i = 0, 1, \ldots, n-1) \\
C &= \{q_i \neq q_j \mid i < j\} \\
&\cup \{q_i + i \neq q_j + j \mid i < j\} \\
&\cup \{q_i - i \neq q_j - j \mid i < j\}
\end{aligned}
$$

n クイーン配置問題 (2)

制約ネットワークを生成

```
Network net = new Network();
```

変数を生成

```
int n = 100;
IntVariable[] q = new IntVariable[n];
for (int i = 0; i < n; i++)
  q[i] = new IntVariable(net, 1, n);
```

制約の記述 1

```
new NotEquals(net, q);
```

n クイーン配置問題 (3)

制約の記述 2

```
IntVariable[] u = new IntVariable[n];
for (int i = 0; i < n; i++)
  u[i] = q[i].add(i);
new NotEquals(net, u);
```

制約の記述 3

```
IntVariable[] d = new IntVariable[n];
for (int i = 0; i < n; i++)
  d[i] = q[i].subtract(i);
new NotEquals(net, d);
```

n クイーン配置問題 (4)

制約ソルバーを生成

```
Solver solver = new DefaultSolver(net);
```

解の探索

```
for (solver.start(); solver.waitNext(); solver.resume()) {
  Solution solution = solver.getSolution();
  for (int i = 0; i < n; i++)
    System.out.print(solution.getIntValue(q[i]) + " ");
  System.out.println();
}
solver.stop();
```

n クイーン配置問題 (5)

```
Network net = new Network();
int n = 100;
IntVariable[] q = new IntVariable[n];
for (int i = 0; i < n; i++)
  q[i] = new IntVariable(net, 1, n);
new NotEquals(net, q);
IntVariable[] u = new IntVariable[n];
for (int i = 0; i < n; i++)
  u[i] = q[i].add(i);
new NotEquals(net, u);
IntVariable[] d = new IntVariable[n];
for (int i = 0; i < n; i++)
  d[i] = q[i].subtract(i);
new NotEquals(net, d);
Solver solver = new DefaultSolver(net);
for (solver.start(); solver.waitNext(); solver.resume()) {
  Solution solution = solver.getSolution();
  for (int i = 0; i < n; i++)
    System.out.print(solution.getIntValue(q[i]) + " ");
  System.out.println();
}
solver.stop();
```

Calc/Cream

Calc/Cream

Calc/Cream は，制約プログラミングのための OpenOffice.org の Calc スプレッドシート・フロントエンドである．

- ユーザは，スプレッドシート中に変数や制約を記述することができる．
- Cream をバックエンドの制約ソルバーとして使用している．
- OpenOffice.org の Java プラグインとして実現されている．

- http://bach.istc.kobe-u.ac.jp/cream/calc.html ▸ Web

Calc/Cream の使い方 (1)

変数の宣言

- CVARIABLES(Inf; Sup; $Range_1$; ...; $Range_n$)
  - Inf: **変数の下限値**
  - Sup: **変数の上限値**
  - $Range_i$: **変数の位置**

制約の宣言

- CONSTRAINTS($Range_1$; ...; $Range_n$)
  - $Range_i$: **制約の位置**

Calc/Cream の使い方 (2)

制約に利用できる式

- +, −, *, +, ABS(*Cell*)
- =, <>, <, <=, >, >=
- SUM($Range_1$;...;$Range_n$)
- CEQUALS($Range_1$;...;$Range_n$): **すべて等しい**
- CNOTEQUALS($Range_1$;...;$Range_n$): **互いに異なる**

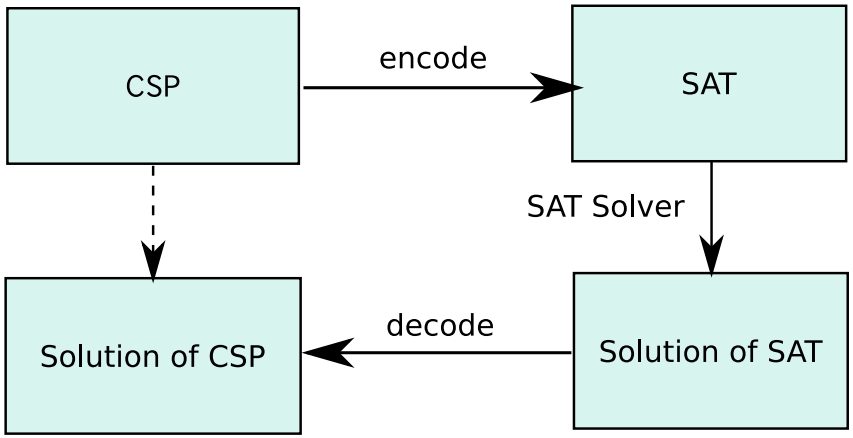
- **最適化のためのマクロ式も用意されているが，今回は省略する．**

SAT 型制約ソルバー Sugar

Sugar の概要

SAT 型制約ソルバー Sugar

- Sugar は，制約充足問題を命題論理式に変換し，高速な SAT ソルバーを用いて求解を行う SAT 型の制約ソルバーである．
- BSD ライセンスで公開
  - http://bach.istc.kobe-u.ac.jp/sugar/ ▸ Web



- 2008 年および 2009 年の国際 CSP ソルバー競技会の複数部門で優勝．

SAT

SAT (Boolean Satisfiability Testing)

与えられた命題論理式について，充足する値割当てが存在するか否かを判定する問題

- SAT は，理論上も実際上も計算機科学の中心的課題である．
- SAT は，NP 完全であることが初めて示された問題でもある [Cook 1971]．
- 近年になって，非常に高速な SAT ソルバーが開発された (MiniSat 等)．
- それを背景として，多くの分野で SAT ソルバーの利用が活発になっている．

CNF

- **通常，命題論理式は連言標準形** (CNF; Conjunctive Normal Form) **で与えられる．**
  - **CNF 式は，複数の節の連言** (AND) **である．**
  - **節** (clause) **は，複数のリテラルの選言** (OR) **である．**
  - **リテラル** (literal) **は，命題変数またはその否定である．**
- **標準的フォーマットとしては，**DIMACS CNF **が用いられる．**

```
p cnf 9 7       ; Number of variables and clauses
1 2 0           ; a ∨ b
9 3 0           ; c ∨ d
1 8 4 0         ; a ∨ e ∨ f
-2 -4 5 0       ; ¬b ∨ ¬f ∨ g
-4 6 0          ; ¬f ∨ h
-2 -6 7 0       ; ¬b ∨ ¬h ∨ i
-5 -7 0         ; ¬g ∨ ¬i
```

SAT ソルバー

- **SAT ソルバー** (SAT solver) **は，与えられた** CNF **の命題論理式が，充足可能** (SAT) **か充足不能** (UNSAT) **かを判定するプログラムである．**
- **答えが** SAT **の場合，**SAT **ソルバーは値割当ても解として出力する．**
- **系統的** (systematic)SAT **ソルバーは，** SAT **か** UNSAT **かを答える．**
  - **多くは，DPLL アルゴリズムを用いている．**
- **確率的** (stochastic)SAT **ソルバーは，** SAT **のみを答える** (UNSAT **は答えない**)**．**
  - **局所探索アルゴリズムが用いられる．**

DPLL アルゴリズム

[Davis & Putnam 1960], [Davis, Logemann & Loveland 1962]

```
(1)  function DPLL(S: a CNF formula, σ: a variable assignment)
(2)  begin
(3)    while ∅ ∉ Sσ and ∃{l} ∈ Sσ do /* unit propagation */
(4)      if l is positive then σ := σ ∪ {l ↦ 1};
(5)                       else  σ := σ ∪ {l̄ ↦ 0};
(6)    if S is satisfied by σ then return true;
(7)    if ∅ ∈ Sσ then return false;
(8)    choose an unassigned variable x from Sσ;
(9)    return DPLL(S, σ ∪ {x ↦ 0}) or DPLL(S, σ ∪ {x ↦ 1});
(10) end
```

- **$S\sigma$ は，S に σ を適用した結果を表す．**
- **$\emptyset$ は，空節 (すなわち矛盾) を表す．**

DPLL

1. a を選択し，$a \mapsto 0$ とする

$$C_1 : a \vee b$$
$$C_2 : c \vee d$$
$$C_3 : a \vee e \vee f$$
$$C_4 : \neg b \vee \neg f \vee g$$
$$C_5 : \neg f \vee h$$
$$C_6 : \neg b \vee \neg h \vee i$$
$$C_7 : \neg g \vee \neg i$$

DPLL

$$C_1 : a \vee b$$
$$C_2 : c \vee d$$
$$C_3 : a \vee e \vee f$$
$$C_4 : \neg b \vee \neg f \vee g$$
$$C_5 : \neg f \vee h$$
$$C_6 : \neg b \vee \neg h \vee i$$
$$C_7 : \neg g \vee \neg i$$

1. **a を選択し，$a \mapsto 0$ とする**
   - **C_1 から，$b \mapsto 1$ を伝播**

DPLL

$$C_1 : a \vee b$$
$$C_2 : c \vee d$$
$$C_3 : a \vee e \vee f$$
$$C_4 : \neg b \vee \neg f \vee g$$
$$C_5 : \neg f \vee h$$
$$C_6 : \neg b \vee \neg h \vee i$$
$$C_7 : \neg g \vee \neg i$$

1. a を選択し，$a \mapsto 0$ とする
   - C_1 から，$b \mapsto 1$ を伝播
2. c を選択し，$c \mapsto 0$ とする

DPLL

$$C_1 : a \vee b$$
$$C_2 : c \vee d$$
$$C_3 : a \vee e \vee f$$
$$C_4 : \neg b \vee \neg f \vee g$$
$$C_5 : \neg f \vee h$$
$$C_6 : \neg b \vee \neg h \vee i$$
$$C_7 : \neg g \vee \neg i$$

1. a を選択し，$a \mapsto 0$ とする
   - C_1 から，$b \mapsto 1$ を伝播
2. c を選択し，$c \mapsto 0$ とする
   - C_2 から $d \mapsto 1$ を伝播

DPLL

$$C_1 : a \vee b$$
$$C_2 : c \vee d$$
$$C_3 : a \vee e \vee f$$
$$C_4 : \neg b \vee \neg f \vee g$$
$$C_5 : \neg f \vee h$$
$$C_6 : \neg b \vee \neg h \vee i$$
$$C_7 : \neg g \vee \neg i$$

1. a を選択し，$a \mapsto 0$ とする
   - C_1 から，$b \mapsto 1$ を伝播
2. c を選択し，$c \mapsto 0$ とする
   - C_2 から $d \mapsto 1$ を伝播
3. e を選択し，$e \mapsto 0$ とする

DPLL

$$C_1 : a \vee b$$
$$C_2 : c \vee d$$
$$C_3 : a \vee e \vee f$$
$$C_4 : \neg b \vee \neg f \vee g$$
$$C_5 : \neg f \vee h$$
$$C_6 : \neg b \vee \neg h \vee i$$
$$C_7 : \neg g \vee \neg i$$

1. a を選択し，$a \mapsto 0$ とする
   - C_1 から，$b \mapsto 1$ を伝播
2. c を選択し，$c \mapsto 0$ とする
   - C_2 から $d \mapsto 1$ を伝播
3. e を選択し，$e \mapsto 0$ とする
   - C_3 から $f \mapsto 1$ を伝播

DPLL

$$C_1 : a \vee b$$
$$C_2 : c \vee d$$
$$C_3 : a \vee e \vee f$$
$$C_4 : \neg b \vee \neg f \vee g$$
$$C_5 : \neg f \vee h$$
$$C_6 : \neg b \vee \neg h \vee i$$
$$C_7 : \neg g \vee \neg i$$

1. a を選択し，$a \mapsto 0$ とする
   - C_1 から，$b \mapsto 1$ を伝播
2. c を選択し，$c \mapsto 0$ とする
   - C_2 から $d \mapsto 1$ を伝播
3. e を選択し，$e \mapsto 0$ とする
   - C_3 から $f \mapsto 1$ を伝播
   - C_4 から $g \mapsto 1$ を伝播

DPLL

$$C_1 : a \vee b$$
$$C_2 : c \vee d$$
$$C_3 : a \vee e \vee f$$
$$C_4 : \neg b \vee \neg f \vee g$$
$$C_5 : \neg f \vee h$$
$$C_6 : \neg b \vee \neg h \vee i$$
$$C_7 : \neg g \vee \neg i$$

1. a を選択し，$a \mapsto 0$ とする
   - C_1 から，$b \mapsto 1$ を伝播
2. c を選択し，$c \mapsto 0$ とする
   - C_2 から $d \mapsto 1$ を伝播
3. e を選択し，$e \mapsto 0$ とする
   - C_3 から $f \mapsto 1$ を伝播
   - C_4 から $g \mapsto 1$ を伝播
   - C_7 から $i \mapsto 0$ を伝播

DPLL

$$C_1 : a \vee b$$
$$C_2 : c \vee d$$
$$C_3 : a \vee e \vee f$$
$$C_4 : \neg b \vee \neg f \vee g$$
$$C_5 : \neg f \vee h$$
$$C_6 : \neg b \vee \neg h \vee i$$
$$C_7 : \neg g \vee \neg i$$

1. a を選択し，$a \mapsto 0$ とする
   - C_1 から，$b \mapsto 1$ を伝播
2. c を選択し，$c \mapsto 0$ とする
   - C_2 から $d \mapsto 1$ を伝播
3. e を選択し，$e \mapsto 0$ とする
   - C_3 から $f \mapsto 1$ を伝播
   - C_4 から $g \mapsto 1$ を伝播
   - C_7 から $i \mapsto 0$ を伝播
   - C_5 から $h \mapsto 1$ を伝播

DPLL

$$C_1 : a \vee b$$
$$C_2 : c \vee d$$
$$C_3 : a \vee e \vee f$$
$$C_4 : \neg b \vee \neg f \vee g$$
$$C_5 : \neg f \vee h$$
$$C_6 : \neg b \vee \neg h \vee i$$
$$C_7 : \neg g \vee \neg i$$

1. a を選択し，$a \mapsto 0$ とする
   - C_1 から，$b \mapsto 1$ を伝播
2. c を選択し，$c \mapsto 0$ とする
   - C_2 から $d \mapsto 1$ を伝播
3. e を選択し，$e \mapsto 0$ とする
   - C_3 から $f \mapsto 1$ を伝播
   - C_4 から $g \mapsto 1$ を伝播
   - C_7 から $i \mapsto 0$ を伝播
   - C_5 から $h \mapsto 1$ を伝播
   - C_6 で矛盾が発生

DPLL

$$C_1 : a \vee b$$
$$C_2 : c \vee d$$
$$C_3 : a \vee e \vee f$$
$$C_4 : \neg b \vee \neg f \vee g$$
$$C_5 : \neg f \vee h$$
$$C_6 : \neg b \vee \neg h \vee i$$
$$C_7 : \neg g \vee \neg i$$

1. a を選択し，$a \mapsto 0$ とする
   - C_1 から，$b \mapsto 1$ を伝播
2. c を選択し，$c \mapsto 0$ とする
   - C_2 から $d \mapsto 1$ を伝播
3. e を選択し，$e \mapsto 0$ とする
   - C_3 から $f \mapsto 1$ を伝播
   - C_4 から $g \mapsto 1$ を伝播
   - C_7 から $i \mapsto 0$ を伝播
   - C_5 から $h \mapsto 1$ を伝播
   - C_6 で矛盾が発生
4. バックトラックし，$e \mapsto 1$ とする

最新の SAT ソルバー

- **最新の SAT ソルバーには，以下の技術が** DPLL **に導入され，大幅な性能向上が実現されている．**
  - **矛盾からの節学習** (CDCL; Conflict Driven Clause Learning) [Silva 1996]
  - **非時間順バックトラック法** [Silva 1996]
  - **ランダム・リスタート** [Gomes 1998]
  - **監視リテラル** [Moskewicz & Zhang 2001]
  - **変数選択ヒューリスティック** [Moskewicz & Zhang 2001]
- **Chaff と zChaff は，**1 **桁から** 2 **桁の速度改善を実現した** [2001]**．**
- SAT **競技会および** SAT **レースが** 2002 **年から開催され，**SAT **ソルバーの実現技術の進歩に貢献している．**
- **MiniSat ソルバーは，**C++**で** 1000 **行以内のコードで，**2005 **年の** SAT **競技会で素晴らしい成績を収めた．**
- **最新の** SAT **ソルバーは，**10^6 **以上の変数，**10^7 **以上の節を取り扱うことができる．**

矛盾からの節学習 (CDCL)

- **矛盾が生じたら，矛盾の原因が節として抽出され，学習節として記憶される．**
- **学習節は，その後の探索で探索空間を大幅に削減する．**
- **学習節は，後ろ向きに導出 (resolution) を行うことで生成できる．**
- **後ろ向きの導出は，First UIP (Unique Implication Point) で停止するのが最も効果的である [Moskewicz & Zhang 2001]．**

前の例では，$\neg b \vee \neg f$ が学習節として生成される．

SAT ベース・アプローチ

SAT ソルバーをバックエンドとして利用する SAT ベースな手法が，困難な組合せ問題を解くための方法として広く用いられるようになっている．

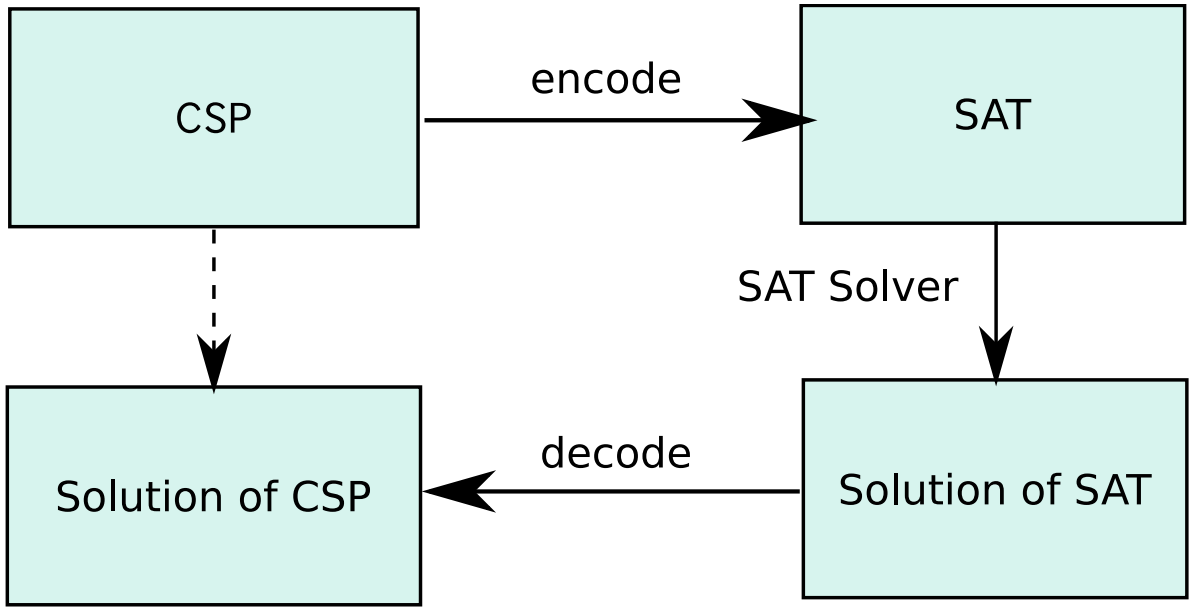
- **プランニング** (SATPLAN, Blackbox) [Kautz & Selman 1992]
- **自動テストパターン生成** [Larrabee 1992]
- **ジョブ・ショップ・スケジューリング** [Crawford & Baker 1994]
- **ソフトウェア仕様** (Alloy) [1998]
- **有界モデル検査** [Biere 1999]
- **ソフトウェアパッケージの依存性解析** (SATURN)
  - SAT4J は，Eclipse 3.4 に用いられている．
- **書換え系** (Aprove, Jambox)
- **Answer Set Programming** (clasp, Cmodels-2)
- **一階論理の定理証明系** (iProver, Darwin)
- **一階論理のモデル発見機** (Paradox)
- **制約充足問題** (Sugar) [Tamura et al. 2006]

Sugar の制約モデリング言語

```
(int q0 1 4)
(int q1 1 4)
(int q2 1 4)
(int q3 1 4)
(alldifferent q0 q1 q2 q3)
(alldifferent (+ q0 1) (+ q1 2) (+ q2 3) (+ q3 4))
(alldifferent (- q0 1) (- q1 2) (- q2 3) (- q3 4))
```

- Sugar **では，**Lisp **風表記による制約モデリング言語を用いている．**

Sugar の動作



- Sugar は，制約モデリング言語による CSP の記述を入力として，以下のように動作する．
  1. CSP を SAT に符号化 (encode) する．
  2. SAT ソルバーを起動し，SAT の解を求める．
  3. 得られた SAT の解を CSP の解に復号化 (decode) する．

なぜ SAT ベースなのか? (個人的見解)

SAT ソルバーは，非常に高速である．

- 2 リテラル監視などの賢いインプリメンテーション
  - バックトラックのために保存する情報が最小限
- キャッシュを考慮したインプリメンテーション [Zhang & Malik 2003]
  - たとえば，オープン・ショップ・スケジューリング問題の gp10-10 は MiniSat で約 4 秒で解けるが，99%以上のキャッシュ・ヒット率である．

```
$ valgrind --tool=cachegrind minisat gp10-10-1091.cnf
L2 refs:       42,842,531  ( 31,633,380 rd +11,209,151 wr)
L2 misses:     25,674,308  ( 19,729,255 rd + 5,945,053 wr)
L2 miss rate:         0.4% (        0.4%   +       1.0%  )
```

なぜ SAT ベースなのか? (個人的見解)

SAT ベース・アプローチは，80 年代にパターソンらが提唱した RISC アプローチに似ている．

- **RISC**: Reduced Instruction Set Computer
- パターソンは，複雑な命令セットのコンピュータ (CISC) よりも，単純で高速な命令セットのコンピュータ (RISC) で最適化コンパイラを利用するほうが高速だと主張した．

SAT ソルバー	$\Longleftrightarrow$	RISC
SAT 符号化	$\Longleftrightarrow$	最適化コンパイラ

- SAT ソルバーと SAT 符号化の双方の研究が，重要だと考える．

SAT の参考資料

- **私のブックマーク：SAT ソルバー人工知能学会誌，第 28 巻 第 2 号** (2013 **年** 3 **月**)
  http://www.ai-gakkai.or.jp/my-bookmark_vol28-no2/ ▸ Web
- **特集「最近の SAT 技術の発展」，人工知能学会誌，第 25 巻 第 1 号** (2010 **年** 1 **月**)
- Handbook of Satisfiability, IOS Press, 2009.

Scala 上の制約プログラミングシステム Copris

Scala

Martin Odersky による関数+オブジェクト指向言語

言語の特徴

- 関数型言語とオブジェクト指向言語の融合
- 強力な型推論，高階関数
- Immutable Collections
- 埋込み DSL の実装に適している

処理系の特徴

- JVM へのコンパイラ処理系
- Java のクラス・ライブラリをそのまま利用できる
- インタラクティブな実行環境も用意されている

- DSL: Domain-Specific Language
- JVM: JVM: Java Virtual Machine

Copris

Copris

- Scala に埋め込まれた制約プログラミング用 DSL
- 自然な制約記述が可能
- 様々な制約ソルバーをバックエンドとして利用可能
  - Sugar
  - SMT ソルバー (Z3, MathSAT など)
  - JSR331 ソルバー (Choco など)
- SAT ソルバーとして Sat4j を用いれば，すべてが JVM 上で動作
- BSD ライセンスで公開
  - http://bach.istc.kobe-u.ac.jp/copris/ ▸ Web
- Copris 上の多数のパズルソルバーも公開している

記述例 (1)

CSP の生成

```
val csp = CSP()
```

変数の生成

```
val x = Var("x")
val y = Var("y")
csp.int(x, 0, 7)
csp.int(y, 0, 7)
```

制約の追加

```
csp.add(x + y === 7)
csp.add(x * 2 + y * 4 === 20)
```

- 演算子のオーバロードで，自然な記述が可能になっている．

記述例 (2)

ソルバーの生成

```
val solver = DefaultSolver(csp)
```

解の表示

```
if (solver.find)
  println(solver.solution)
```

- Scala で記述した変数や制約は，Sugar の形式に変換され，さらに SAT に符号化され，外部の SAT ソルバーを用いて求解される．

記述例 (3)

```
import jp.kobe_u.copris._

val csp = CSP()
val x = Var("x")
val y = Var("y")
csp.int(x, 0, 7)
csp.int(y, 0, 7)
csp.add(x + y === 7)
csp.add(x * 2 + y * 4 === 20)
val solver = DefaultSolver(csp)
if (solver.find)
  println(solver.solution)
```

記述例 (3)

```
import jp.kobe_u.copris._
import jp.kobe_u.copris.dsl._

int('x, 0, 7)
int('y, 0, 7)
add('x + 'y === 7)
add('x * 2 + 'y * 4 === 20)
if (find)
  println(solution)
```

- DSL メソッドを import すると，簡潔な記述が可能になる．
- 既定の CSP オブジェクトや Solver オブジェクトの利用
- シンボル ('x 等) から変数オブジェクトへの暗黙変換

n クイーン配置問題

n クイーン配置問題

```
def queens(n: Int) = {
  for (i <- 1 to n) int('q(i), 1, n)
  add(Alldifferent((1 to n).map(i => 'q(i))))
  add(Alldifferent((1 to n).map(i => 'q(i) + i)))
  add(Alldifferent((1 to n).map(i => 'q(i) - i)))
  if (find)
    do {
      println(solution)
    } while (findNext)
}
```

実行例

```
Map(q(2)->7,q(3)->5,q(7)->6,q(4)->0,q(5)->2,q(0)->3,q(6)->4,q(1)->1)
Map(q(2)->1,q(3)->5,q(7)->7,q(4)->2,q(5)->0,q(0)->4,q(6)->3,q(1)->6)
.....
```

完全正方形分割

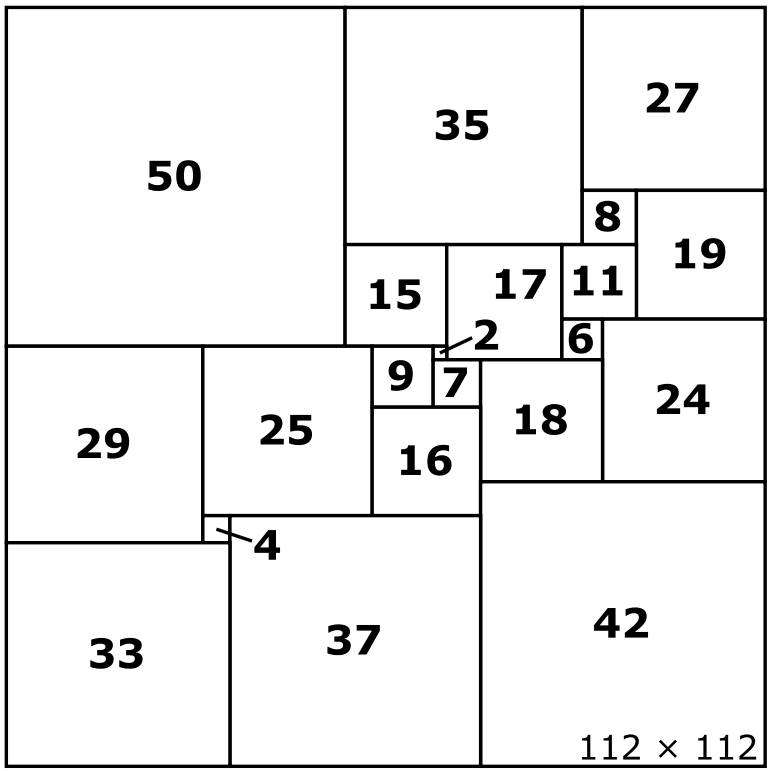
完全正方形分割 (ルジンの問題)

辺長が 112 **の正方形を，辺長が** 2, 4, 6, 7, 8, 9, 11, 15, 16, 17, 18, 19, 24, 25, 27, 29, 33, 35, 37, 42, 50 **の** 21 **個に分割する．**

完全正方形分割

完全正方形分割 (ルジンの問題)

辺長が 112 **の正方形を，辺長が** 2, 4, 6, 7, 8, 9, 11, 15, 16, 17, 18, 19, 24, 25, 27, 29, 33, 35, 37, 42, 50 **の** 21 **個に分割する．**



(Wikipedia **より**)

完全正方形分割

```
val size = 112
val s = List(2,4,6,7,8,9,11,15,16,17,18,
             19,24,25,27,29,33,35,37,42,50)
val n = s.size
for (i <- 0 to n-1) {
  int('x(i), 0, size - s(i))
  int('y(i), 0, size - s(i))
}
for (i <- 0 to n-1; j <- i+1 to n-1) {
  add('x(i) + s(i) <= 'x(j) || 'x(j) + s(j) <= 'x(i) ||
      'y(i) + s(i) <= 'y(j) || 'y(j) + s(j) <= 'y(i))
}
if (find) {
  解の表示
}
```

- 90 秒程度で求解できる.

まとめ

以下の紹介を行った.

1. 制約充足問題
2. 制約充足アルゴリズム
3. Cream の紹介
   - Java 上の制約プログラミング用クラス・ライブラリ
   - OpenOffice.org Calc インターフェイスのデモ
4. Sugar の紹介
   - SAT 型制約ソルバー
5. Copris の紹介