敵対者に対応する協調問題解決：限量記号付き分散制約充足問題

Cooperative Problem Solving against Adversary: Quantified Distributed Constraint Satisfaction Problem

馬場 里美[*1] Satomi Baba　西村 直史[*1] Naofumi Nishimura　岩崎 敦[*1] Atsushi iwasaki　櫻井 祐子[*1*2] Yuko Sakurai　横尾 真[*1] Makoto yokoo

[*1]九州大学大学院システム情報科学府
Graduate School of ISEE, Kyushu University

In this paper, we extend the traditional formalization of a Distributed Constraint Satisfaction Problems (Dis-CSP) to a Quantified Dis-CSP, which includes several universally quantified variables, while all variables in a traditional dis-CSP are existentially quantified. A universally quantified variable represents a choice of the nature or an adversary. A Quantified Dis-CSP formalizes a situation where a team of agents is trying to make a robust plan against the nature or an adversary. In this paper, we present the formalization of Quantified Dis-CSP. Furthermore, we develop an algorithm for solving a quantified dis-CSP. This algorithm generalizes the asynchronous backtracking algorithm for solving a dis-CSP. In this algorithm, agents communicate a value assignment called *good* in addition to a *nogood* used in the asynchronous backtracking. Interestingly, the procedures executed by an adversarial/cooperative agent for *good*/*nogood* are totally symmetrical.

1. はじめに

制約充足問題 [Mackworth 92] は全ての制約を満足する，変数への値の割当を求める問題である．各変数は有限で離散的な領域から値を取る．制約充足問題は人工知能の様々な問題を定式化できる．そのため，制約充足問題の研究は非常に重要で，これまでに多くの研究がなされている．

分散制約充足問題 [Yokoo 01] は制約充足問題の変数と制約を複数エージェントへ分散させた問題である．様々なマルチエージェントシステムの問題 [Yokoo 01] (分散資源割当問題, 分散スケジューリング問題, マルチエージェントによるデータの真偽値管理等) が分散制約充足問題で定式化できる．

一方，限量記号付き制約充足問題 [Chen 04] は一部の変数を全称限量記号で限量化することにより制約充足問題を拡張した問題である．限量記号付き制約充足問題では，全称限量化された変数の値に関わらず，存在限量化された変数へ全ての制約を満足する値を割り当てることを目的とする．制約充足問題が NP 完全問題であるのに対し，限量記号付き制約充足問題は PSPACE 完全問題である．限量記号付き制約充足問題では，全称限量化された変数を不確定要素や敵などと考えることができるため，不確定要素を含むプランニングや敵が存在するゲームなどの問題を定式化できる．また，充足可能性問題を一般化した限量記号付きブール式では，様々なアルゴリズム (e.g.[Zhang 02]) が提案されている．限量記号付き制約充足問題を解くアルゴリズムの多くはそれらをベースとしている．

本論文では分散制約充足問題と限量記号付き制約充足問題を組み合わせた限量記号付き分散制約充足問題を提案する．通常の分散制約充足問題では，全ての変数が存在限量化されていると考えられ，協調的なエージェントがその変数の値を決定する．一方，本論文で提案する限量記号付き分散制約充足問題には，全称限量化されている変数が含まれており，敵対的なエージェントがその変数の値を決定する．したがって，限量記号付き分散制約充足問題は不確定要素や敵を含む問題を定式化できる．本問題では，協調的なエージェントのチームが，全ての制約を満足するように敵に対抗するプランを構築することを目指す．さらに，本論文では限量記号付き分散制約充足問題を解くアルゴリズムを提案する．本アルゴリズムは分散制約充足問題を解く非同期バックトラッキングアルゴリズム [Yokoo 92] を一般化したものである．本アルゴリズムでは，制約を違反する変数への値の割当である *nogood* と，いくつかの制約を満足する変数への値の割当である *good* をエージェントが通信する．また，本アルゴリズムでは敵対的なエージェントが *good*/*nogood* に関して協調的なエージェントと対称的な処理を行う．

連絡先: 馬場 里美，九州大学大学院システム情報科学府 819-0395 福岡県福岡市西区元岡 744 番地，(092)802-3576, s-baba@agent.is.kyushu-u.ac.jp

*2 現在はヤフー株式会社所属．

2. 関連研究

2.1 限量記号付き制約充足問題

制約充足問題は，n 個の変数 $x_1, x_2, ..., x_n$ と，変数のそれぞれが値をとる有限で離散的な領域 $D_1, D_2, ..., D_n$，および制約の集合 $C_1, C_2, ..., C_m$ で定義され，制約を満足する値の割当を発見することを目的とした問題である．

一部の変数を全称限量化することで，制約充足問題を一般化した問題が限量記号付き制約充足問題である．制約充足問題が NP 完全問題であるのに対し，限量記号付き制約充足問題は PSPACE 完全問題である．限量記号付き制約充足問題は，変数の集合，領域の集合，制約の集合と限量記号のシーケンスで構成され，シーケンス Q と制約 C で QC と定義する．

限量記号のシーケンスは限量記号 $Q_i(\exists, \forall)$ と変数 x_i で構成される．限量記号のシーケンスは，各変数を限量化する限量記号を定義するとともに，変数の順序も定義している．例えば，$\exists x_1 \forall x_2$ と $\forall x_2 \exists x_1$ では，それぞれのもつ意味が異なるため，順序の定義にも注意する必要がある．$\exists x_1 \forall x_2 C$ と $\forall x_2 \exists x_1 C$ が解決可能であるとき，それぞれの意味は次の通りである．

- $\exists x_1 \forall x_2 C$ が解決可能：ある x_1 の値が存在し，その値は任意の x_2 の値に対して制約 C を満足する．
- $\forall x_2 \exists x_1 C$ が解決可能：x_2 がとり得る値それぞれに対して，制約 C を満足する x_1 の値が存在する．

限量記号付き制約充足問題の意味論は，再帰的に次に示すように定義される．

- C が空ならその問題の制約は満足される．Q が $\exists x_1 Q_2 x_2 \cdots Q_n x_n$ の形のとき，$Q_2 x_2 \cdots Q_n x_n C[(x_1, a)]$ を満足する値 $a \in D(x_1)$ が存在するならその問題の制約は満足される．Q が $\forall x_1 Q_2 x_2 \cdots Q_n x_n$ の形のとき，任意の値 $a \in D(x_1)$ に対して $Q_2 x_2 \cdots Q_n x_n C[(x_1, a)]$ を満足する割り当てが存在するならその問題の制約は満足される．(x_1, a) は x_1 に値 a を割り当てることを意味する．

2.2 分散制約充足問題

分散制約充足問題とは，制約充足問題の変数が複数のエージェントに分散された問題である．各エージェントは，エージェントが管理する変数に関する知識 (例えば，変数の領域や制約) のみを持っている状態で，他のエージェントと通信を行う．本論文では，エージェント間通信はメッセージ通信によってなされると仮定する．また，アルゴリズムの説明の簡単化のため，各エージェントの持つ変数は唯一であり，エージェント間の制約は全て二項関係 (binary) であることを仮定する．

非同期バックトラッキングアルゴリズム

非同期バックトラッキングアルゴリズムでは，各エージェントは非同期，並行的に自分の変数の値を決定し，その値を他エージェントに送信した後，メッセージ待ちの状態となり，以降，到着したメッセージに関する処理を行う．エージェントには識別子により優先順位が与えられるとする．非同期バックトラッキングアルゴリズムでは次の二種類のメッセージを用いる．

- (**ok?**, $(x_j, value)$) : 変数 x_j の値の割当が $value$ であることを伝えるメッセージ．
- (**nogood**, $x_j, nogood$) : 新たに発生した $nogood$ を伝えるバックトラックメッセージ．$nogood$ は制約違反を起こす値の組合せである．例えば，$nogood\{(x_i, d_i), (x_j, d_j)\}$ は，(x_i, d_i) と (x_j, d_j) が制約違反を起こすことを表す．

エージェントは **ok?**メッセージ，**nogood** メッセージを受信すると，現在の値が制約違反を起こしていないかチェックする．現在の値が違反を起こしている場合は，制約を満足する値を探すが，見つからなければ，自分より優先順位の高いエージェントのうち最も優先順位の低いエージェントへ **nogood** メッセージを送信する．

3. 限量記号付き分散制約充足問題

3.1 問題の定義

限量記号付き分散制約充足問題は，変数と制約がエージェントへ分散された限量記号付き制約充足問題である．存在限量化された変数は協調的なエージェントに分散され，全称限量化された変数は敵対的なエージェントが持つとする．また，限量記号のシーケンスは値の決定順序を定義する．すなわち，シーケンス中で x_i が x_j より前に現れるなら，x_j の値を決定するときには，x_i の値は観測可能である．限量記号付き分散制約充足問題において，協調的なエージェントのチームは，敵対的なエージェントがどのような値を選択しても，全ての制約を満足する値の組合せを求めることを目的とする．

3.2 限量記号付き分散制約充足問題の適用例

限量記号付き分散制約充足問題の適用例を，三目並べを例に紹介する．三目並べは，二人のプレーヤーが 3×3 マスのボードに交互に駒を置いていき，縦，横，斜めのいずれかに相手より先に駒を三つ並べ，ラインを作ることを目的とするゲームである．ここでは，後手 (敵) がどのような手を打っても，先手が負けない打ち方を求めることを目的とする．

三目並べを限量記号付き分散制約充足問題として定式化すると，変数は $x_1, \cdots, x_9$ で，各変数の持つ領域はボードの各マス $\{1, 2, \cdots, 9\}$ である．また，敵が先にラインを作らないという制約があり，限量記号のシーケンスは $\exists x^1 \forall x^2 \cdots \exists x^9$ (i が奇数のとき $\exists x_i$，i が偶数のとき $\forall x_i$) と定義できる．

4. 限量記号付き分散制約充足問題を解くアルゴリズム

本章では，限量記号付き分散制約充足問題を解くアルゴリズムを提案する．提案アルゴリズムは，非同期バックトラッキングアルゴリズムをベースとしたアルゴリズムである．

4.1 基本的なアイデア

以下に，非同期バックトラッキングアルゴリズムからの拡張アイデアを示す．

- 敵対的なエージェントを模倣して，探索に協力する仮想的なエージェントを配置する．仮想的なエージェントは敵の行動を模倣するが，協調的なエージェントのチームが求めるプランの探索に協力する．ある意味では，協調的なエージェントのチームはオフラインでプランを構築する．チームが実際に敵と対戦するとき，オフライン探索で得られたプランを実行する．実際には，チーム中のあるエージェントが仮想的なエージェントとして行動する．
- エージェント間の優先順位は限量記号のシーケンスに基づいて決定する．x_j が x_i より前に現れるなら，x_j は x_i より優先順位が高い．ただし，シーケンス中で隣り合っている存在限量化された変数の順位は任意に決定される．
- 簡単化のため，エージェントは優先順位によって決定される DFS 木を知っているものとする．
- エージェントは，**ok?**メッセージ，**nogood** メッセージに加え，**good** メッセージを用いて通信を行う．$good$ は **good** メッセージの送信者とその子孫が持つ制約を満足する値の組合せである．敵対的なエージェントは $good/nogood$ に関して，協調的なエージェントと対称的な処理を行う．

4.2 $nogood$ と $good$

本節では，$good/nogood$ に対する存在限量記号/全称限量記号で限量化された変数における処理を比較する．$good/nogood$ の論理的な意味と生成について述べた上で比較を行う．

***nogood*:** $nogood$ は，矛盾を導く変数の値の組合せである．例えば，$nogood\{(x_1, 1), (x_2, 2)\}$ は，$x_1 = 1 \wedge x_2 = 2 \rightarrow \bot$ であることを表している．すなわち，$x_1 = 1 \wedge x_2 = 2$ という値の組合せのとき，矛盾が生じることを示す．

***nogood* の生成:** 新たな $nogood$ は，以下のように導かれる．

- x_2 が存在限量化されている変数のとき，$x_2 = 1 \vee x_2 = 2$，$nogood\{(x_1, 1), (x_2, 1)\}$，$nogood\{(x_1, 1), (x_2, 2)\}$ から新たな $nogood\{(x_1, 1)\}$ を導くことができる．
- x_2 が全称限量化された変数のとき，$nogood\{(x_1, 1), (x_2, 1)\}$ より $nogood\{(x_1, 1)\}$ を導くことができる．

存在限量化された変数を持つ協調的なエージェントは，自分の持つ変数がとり得る全ての値が (自分の持つ制約と受信した

nogood によって) 矛盾を引き起こす場合に限って，**nogood** メッセージを送信する．

一方，全称限量化された変数を持つ仮想的/敵対的なエージェントは，矛盾を引き起こす値を少なくとも一つ発見したなら，すぐに **nogood** メッセージを送信する．

***good*:** *good* はいくつかの制約を満足する値の組合せである．例えば，エージェント x_3 が送信した $good\{(x_1,1),(x_2,2)\}$ は $x_1 = 1 \wedge x_2 = 2$ が x_3 とその子孫の制約を満足することを表す．

***good* の生成:** *good* は以下のようにして導かれる．

- x_2 が存在限量化されているとき，$x_1 = 1 \wedge x_2 = 1$ が x_1 と x_2 間の制約を満足するなら，唯一の子 x_3 から送信された $good\{(x_1,1),(x_2,1)\}$ より $good\{(x_1,1)\}$ を導ける．
- x_2 が全称限量化されているとき，$x_1 = 1 \wedge x_2 = 1$ と $x_1 = 1 \wedge x_2 = 2$ が x_1 と x_2 間の制約を満足するなら，$x_2 = 1 \vee x_2 = 2$ と唯一の子 x_3 から送信された $good\{(x_1,1),(x_2,1)\}$，$good\{(x_1,1),(x_2,2)\}$ より $good\{(x_1,1)\}$ を導くことができる．

存在限量化された変数を持つ協調的なエージェントは，少なくとも一つの値に対して *good* を受信したなら (かつその値が制約を満足するなら)，すぐに **good** メッセージを送信する．一方，全称限量化された変数を持つ仮想的/敵対的なエージェントは，自分の持つ変数がとり得る全ての値に対して，*good* を受信した (かつ各値が制約を満足する) ときに限り，**good** メッセージを送信する．

4.3 アルゴリズムの詳細

本アルゴリズムでは，非同期バックトラッキングアルゴリズムと同様に，各エージェントは非同期，並行的に値を決定する．簡単化のため，エージェントは **ok?**メッセージを用いて，値を全ての子孫エージェントへ送信するとする．エージェントは値を送信した後，メッセージ待ちの状態になり，以降，到着したメッセージに関する処理を行う．

本アルゴリズムで通信されるメッセージは，**ok?**，**nogood**，**good** の三種類である．図 1 に協調的なエージェントのメッセージ処理，図 2 に敵対的/仮想的なエージェントのメッセージ処理を示す．また，図 1，2 中の **backtrack** は *nogood* を親エージェントへ送信する処理，**send_good** は *good* を親エージェントへ送信する処理を表している．

協調的なエージェントのメッセージ処理は，非同期バックトラッキングアルゴリズムのメッセージ処理とほぼ同じであり，非同期バックトラッキングアルゴリズムとの違いは，**good** メッセージに対する処理である．ただし，葉エージェントが **ok?** メッセージを受信し，制約を満足する値が存在する場合は，親エージェントへ **good** メッセージを送信する．一方，敵対的なエージェントが **ok?**メッセージを受信したとき，エージェントは，制約違反を起こす値を一つでも発見できたなら，**nogood** メッセージを送信する．

協調的なエージェントが全ての制約を満足する値を求めるのに対し，敵対的/仮想的なエージェントは制約を違反する値を求めようとする．したがって，敵対的なエージェントは，*good*, *nogood* に関して協調的なエージェントと対称な処理を行う．

協調的なエージェントが **nogood** メッセージを受信したとき，エージェントは制約を満足する他の値を探す．制約を満足する値がなければ，親へ **nogood** メッセージを送信する．一方，敵対的なエージェントが **nogood** メッセージを受信したとき，エージェントは制約を満足する値を探さず，**nogood** メッセージを親へ送信する．なぜなら，エージェントは受信した *nogood* 中の値を選択し，制約を違反できるからである．

```
when received (ok?, (x_j, value)) do
  add (x_j, value) to agent_view;
  check_agent_view;
  when agent is a leaf and agent_view contains all ancestors do
    send_good; end do; end do;

when received (nogood, x_j, nogood) do
  add nogood to nogood_list
  check_agent_view; end do;

when received (good, x_j, good) do
  add good to good_list
  when received consistent good from all children
    and agent_view contains all ancestors do
    send_good; end do; end do;

procedure check_agent_view;
  when current_value and agent_view are inconsistent do
    change current_value to a new consistent value;
    when cannot find such a value do backtrack; end do;
  send (ok?, (x_i, current_value)) to descendants;
```

図 1: 協調的なエージェントのメッセージに対する処理

```
when received (ok?, (x_j, value)) do
  add (x_j, value) to agent_view;
  check_agent_view_adversary;
  end do;

when received (nogood, x_j, nogood) do
  add nogood to nogood_list;
  when nogood is consistent with agent_view and current_value,
    and agent_view contains all ancestors do
    backtrack; end do; end do;

when received (good, x_j, good) do
  add good to good_list;
  check_agent_view_adversary; end do;

procedure check_agent_view_adversary;
  when for some v ∈ D_i, v and agent_view are inconsistent do
    backtrack; end do;
  when for all v ∈ D_i, received good messages from all children
    (or a leaf agent), and agent_view and v are consistent do
    send_good; end do;
  otherwise do
    choose v ∈ D_i so that some children have not send good message yet;
    change current_value to v;
    send (ok?, (x_i, current_value)) to descendants; end do;
```

図 2: 敵対的なエージェントのメッセージに対する処理

協調的なエージェントが同じ値の組合せに対して全ての子から **good** メッセージを受信したとき，エージェントは受信した *good* 中の値を選択し，制約を満足できるため，すぐに親へ **good** メッセージを送信する．一方，敵対的なエージェントが **good** メッセージを受信したとき，エージェントは制約を違反する値を探す．そのような値がなければ，すなわち，そのエージェントの持つ変数がとり得る全ての値に対して **good** メッセージを受信したなら，親へ **good** メッセージを送信する．

簡単化のため，提案アルゴリズムは問題を解決可能か否かのみを求めるものとする．敵に対する行動プランを構築するためには，存在限量化された変数を持つエージェントは子から受信した *good* を記録しなければならない．

4.4 アルゴリズムの実行例

アルゴリズムの実行例を図 3(a) に示す．図 3(a) は，$Q = \exists x_1 \forall x_2 \exists x_3 \exists x_4$，$C = \{nogood\{(x_1,1),(x_3,1)\}, nogood\{(x_2,1),(x_3,2)\}, nogood\{(x_2,2),(x_4,1)\}, D_1 = D_2 = D_3 = D_4 = \{1,2\}$ で構成される問題である．

まず，x_1，x_2 から **ok?**メッセージを受信した後，x_3，x_4 の *agent_view* は $\{(x_1,1),(x_2,1)\}$ となる (図 3(b))．x_3 の値はどちらも *agent_view* と制約を満足しないため，x_3 は親である x_2 へ **nogood** メッセージを送信する．一方，x_4 は制約を満足できる値を選択できるので，親である x_2 へ **good** メッセージを送信する (図 3(c))．x_2 はこの **nogood** メッセージと **good**

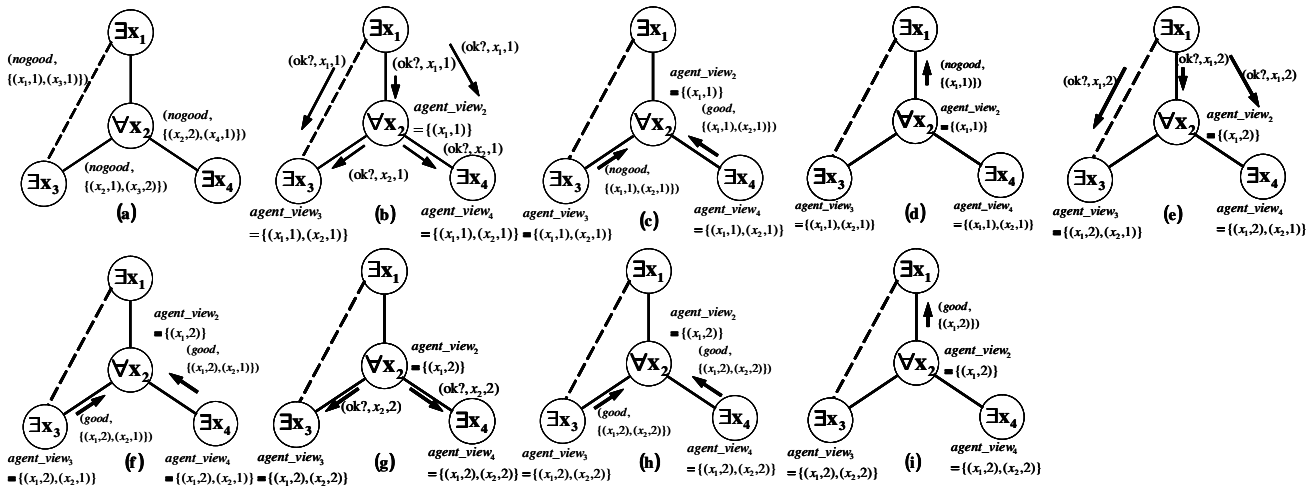

図 3: アルゴリズムの実行例

メッセージを受信する．x_2 は，敵対的なエージェントであるため，x_1 へ **nogood** メッセージを送信する (図 3(d))．

x_1 は，x_2 から送信された **nogood** メッセージより，$x_1 = 1$ が制約違反を引き起こすことを知る．そこで，x_1 は，値を 2 に変更し，子孫へ **ok?**メッセージを送信する (図 3(e))．

この **ok?**メッセージを受信した x_2,x_3,x_4 は，その値を *agent_view* に記録する．値 1 が制約を満足するため，x_3 は x_2 へ **good** メッセージを送信する．x_4 にも制約を満足する値があるため，x_2 へ **good** メッセージを送信する (図 3(f))．

そうすると，x_2 は全ての子から値 1 に対して，*good* を受信したことになる．そこで x_2 は，(制約違反を起こす可能性が残っている) 値 2 をとり，**ok?**メッセージを送信する (図 3(g))．

x_3，x_4 は，この **ok?**メッセージを受信し，*agent_view* に値を記録する．x_3，x_4 ともに制約を満足する値をとることができるため，x_2 へ **good** メッセージを送信する (図 3(h))．この **good** メッセージを x_2 が受信すると，x_2 がとり得る全ての値が制約を満足できるため，x_2 は x_1 へ **good** メッセージを送信する (図 3(i))．x_1 は，協調的なエージェントであるため，現在の値を選び，空集合の *good* を生成する．空集合の *good* が導かれたため，全ての制約を満足する解の存在を示した．

4.5 アルゴリズムの健全/完全性

本アルゴリズムは，根エージェントで空集合の *nogood* が得られれば，解が存在しないことが示され，停止する．根エージェントで空集合の *good* が得られれば，解が存在することが示され，アルゴリズムは停止する．したがって，本アルゴリズムの健全性と完全性を示すには以下の 2 点を証明すればよい．

1. 新たな *nogood*/*good* を生成するための処理は論理的に正しい．すなわち，根エージェントで空集合の *good* が得られれば，解が存在し，空集合の *nogood* が得られれば，解は存在しない．

2. 根エージェントで *nogood* あるいは *good* の空集合が導かれる前に停止したり，処理の無限ループに陥ることは無い．

ここで，この 2 点を定理 1，定理 2 とする．

定理 1 根エージェントで空集合の *good* が生成されるなら解が存在し，根エージェントで空集合の *nogood* が生成されるなら解は存在しない．

定理 2 本アルゴリズムを実行した場合，根エージェントで空集合の *good* あるいは *nogood* が導かれる前に停止したり，処理の無限ループに陥ることはない．

定理 1，定理 2 は数学的帰納法を用いて証明できる．[*1]

5. おわりに

本論文では，分散制約充足問題を一般化した限量記号付き分散制約充足問題を提案した．本問題はエージェントのチームが敵に対抗するプランを構築する状況を定式化した問題である．また，本論文では，非同期バックトラッキングアルゴリズムを拡張し，限量記号付き分散制約充足問題を解くアルゴリズムを提案した．

今後の課題としては，分散制約最適化問題への拡張，限量記号付き分散制約充足問題を解くより効率的なアルゴリズムの開発が挙げられる．

参考文献

[Chen 04] Chen, H. M.: *The computational complexity of quantified constraint satisfaction*, PhD thesis, Ithaca, NY, USA (2004), Adviser-Kozen,, Dexter

[Mackworth 92] Mackworth, A. K.: Constraint Satisfaction, in Shapiro, S. C. ed., *Encyclopedia of Artificial Intelligence*, pp. 285–293, John Wiley & Sons, New York (1992)

[Yokoo 92] Yokoo, M., Durfee, E. H., Ishida, T., and Kuwabara, K.: Distributed Constraint Satisfaction for Formalizing Distributed Problem Solving, in *Proceedings of the Twelfth IEEE International Conference on Distributed Computing Systems*, pp. 614–621 (1992)

[Yokoo 01] Yokoo, M.: *Distributed Constraint Satisfaction: Foundation of Cooperation in Multi-agent Systems*, Springer (2001)

[Zhang 02] Zhang, L. and Malik, S.: Towards a Symmetric Treatment of Satisfaction and Conflicts in Quantified Boolean Formula Evaluation, in *CP*, pp. 200–215 (2002)

∗1 紙幅の都合上，定理の証明は省略する．詳細は著者らまで連絡されたい．