Vol.2010-SLDM-144 No.63
Vol.2010-EMB-16 No.63
Vol.2010-MBL-53 No.63
Vol.2010-UBI-25 No.63
2010/3/28

メッセージ・シークエンス・チャートに対するSAT ソルバーを用いたディスコード計算手法

中川智文[†1]　浜口清治[†1]　垣内洋介[†1]
中西正樹[†3]　谷本匡亮[†2]

メッセージ・シークエンス・チャートは並行プロセス間の動作を図的に表現する手段として提案され，UML などにも取り込まれる形で利用が進んでいる．一方，イベント間の発生順序を必要最低限しか指定しないという特性をもつため，ユーザがかならずしも意図しないイベント発生系列を含む場合がある．これを解析するための手段として，ディスコードの概念が Peled らによって提案され，計算アルゴリズムが示されているが，与えられたメッセージ・シークエンス・チャートのサイズに対して，指数的な計算時間を要する．本報告では，SAT を使ったディスコード計算のアルゴリズムを示す．

SAT-based Discord Computation for Message Sequence Charts

Tomofumi Nakagawa,[†1] Kiyoharu Hamaguchi,[†1]
Yosuke Kakiuchi,[†1] Masaki Nakanishi[†3] and Tadaaki Tanimoto[†2]

Message Sequence Chart is a method for representing interactions between concurrent processes visually, which has been incorporated to UML. On the other hand, only minimum orders among events are specified in the charts, which is one of the main features, may lead to unintentional descriptions where event orders are not what the users wish to describe. In order to analyse such situations, Peled et.al proposed a concept of “discord” and an algorithm for computing discords for a message sequence chart. Their algorithm, however, requires exponential computational time in terms of the size of the given message sequence chart. In this report, we show a new SAT-base algorithm for discord computation.

1. はじめに

近年の半導体回路の大規模化，複雑化に伴って回路設計が複雑になり，検証の必要性が増大している．特に組込みシステムでは，ハードウェア・ドライバ・OS 間のインターフェース仕様が煩雑になってきている．また，プログラム設計に関してもマルチスレッドで高速に処理することが可能になった一方で，シングルスレッドでは起こらないような, 処理の実行順に起因する問題を検証する必要がある．検証が十分に行われなかった場合，意図しない動作で故障を引き起こす可能性がある．しかし，並行動作する複雑な通信システムの場合，仕様を正確かつ分かりやすく記述することは困難になる．そのため,「検証すべき動作を見落とす可能性が高まる」といった問題があり，動作や構造が理解しやすいよう，工夫された仕様記述法が求められる．そこで，通信システムを直感的かつ視覚的に記述可能であるメッセージ・シーケンス・チャート (Message Sequence Chart: MSC)[8] やハイレベル・メッセージ・シーケンス・チャート (High-level Message Sequence Chart: HMSC)[2] が利用される．

MSC とは，プロセス間におけるメッセージのやり取りの流れを表した図である．イベントの発生順序によって，メッセージの処理順序を時系列的に表現することが可能である．MSC はインタフェースや通信の仕様記述に適しており，類似の記述法が UML[9] でも採用されている．しかし，単一の MSC ではある特定の動作に対する処理の流れしか表現することしかできないため，複数の動作に対して処理の流れを記述したい場合には多くの MSC を作成する必要がある．そこで HMSC が利用される．HMSC は各頂点がそれぞれ 1 つの MSC であり，頂点間が有向辺で繋がれている有向グラフである．HMSC の 1 つのパスはパス上の MSC を連結した MSC と考える事で，1 つのシナリオに対する処理の流れを表現できる．そのため，異なるパスをたどることによって，複数の動作に対する処理の流れを表現することが可能である．HMSC は複雑な仕様を 1 つの図で視覚的に表現できるので，通信システムを記述するために利用できる．

しかし，HMSC には「HMSC の表すメッセージの順序と実際に想定していたメッセージの順序が異なる」という問題が起こる可能性がある．そのため，HMSC で書かれた仕様そ

†1 大阪大学 大学院情報科学研究科
Osaka University, Graduate School of Information Science and Technology
†2 (株) ルネサス テクノロジ
Renesas Technology Corp.
†3 山形大学 地域教育文化学部
Yamagata University, Faculty of Education, Art and Science

Vol.2010-SLDM-144 No.63
Vol.2010-EMB-16 No.63
Vol.2010-MBL-53 No.63
Vol.2010-UBI-25 No.63
2010/3/28

のものを検証する必要がある．HMSC を検証する方法としてディスコードを利用する方法がある[1)]．2 つのメッセージに対する可能な処理順序は 6 種類あるが，その 6 種類の順序に対して良し悪しをつけ，2 つのメッセージで起こりうる処理順序のうち最も悪い順序がディスコードである．2 つのメッセージに対するディスコードを計算することで，メッセージの実行順が想定したものと異なるか効率よく調べることができる．しかし，1 つの HMSC に対して多数のパスが存在するので，全てのパスについてメッセージの実行順序を解析すると膨大な時間がかかってしまう．Peled らの方法[1)] では計算量が $O(n|E|2^{4|P|}|P|^2)$ (n は頂点数, E は辺の数，P はプロセス数) にもなることが示されている．そのため,「プロセス数が大きい HMSC でディスコードを計算するのに膨大な時間がかかってしまう」という問題がある．

そこで，頂点数やプロセス数が大きい HMSC を対象とし，HMSC を論理式にエンコードして，ディスコード計算における主要な手続きを充足可能性判定問題に帰着することにより充足可能性判定ツール (SAT ソルバ) を利用して解き，その結果からディスコードを計算するという手法を提案する．近年 SAT ソルバは急速に性能向上が進んでおり，これを利用して，Peled らの手法よりも高速にディスコードを計算することを目指す．

実験では提案した HMSC を論理式にエンコードし，充足可能性判定を利用して解く手法と既存手法の両方を実装して, ディスコードの計算にかかる時間を比べることで，提案手法の有効性を評価する．

2. 準　　備

2.1 MSC

メッセージ・シーケンス・チャート (Message Sequence Chart: MSC) とはプロセス間のメッセージのやり取りを表した図表現である．図 1 が MSC の一例である．図 1 を用いて MSC について説明する． 図 1 の P_1，P_2，P_3 を**プロセス**と呼び，そこから伸びている縦の線をプロセスラインと呼ぶ．図 1 の e_1，e_2，...，e_6 を**イベント**と呼ぶ．イベントは各メッセージの矢印の始点もしくは終点に存在する．矢印の始点のイベントはメッセージの送信を, 矢印の終点のイベントは受信を表す．また，イベント同士は以下のような順序関係を持つ．

[イベントの順序規則]

(1)　あるメッセージの送信イベント e, と受信イベント e' に対して $e < e'$

(2)　同プロセスライン上にある 2 つのイベントのうち，上方にあるイベントを e，下方にあるイベントを e' とすると $e < e'$

例えば, メッセージ M_1 において，規則 1 より $e_1 < e_2$ が成立する．また，プロセス P_2 において，規則 2 より $e_1 < e_6$ が成立する．

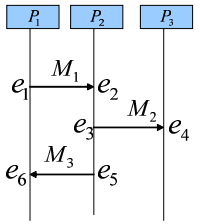

図 1　MSC の例
Fig. 1

図 1 の M_1，M_2，M_3 の矢印を**メッセージ**と呼ぶ．メッセージは送信イベントと受信イベントで構成される．矢印の始点で送信イベントが生起し，矢印の終点で受信イベントが生起することを意味する．例えば，M_1 はプロセス P_1 の送信イベント e_1 で送信され，プロセス P_2 の受信イベント e_2 で受信される．メッセージ m の送信イベントを s, 受信イベントを r として，$m = (s, r)$ のようにイベント対でメッセージを表記する．また MSC C に含まれるメッセージ $m = (s, r)$ について，$m \in C$ や, $m = (s, r) \in C$ と表記することもある. MSC のイベントの順序関係は推移律を持っている．例えば，図 1 の MSC のイベント e_1，e_2，e_3，e_4 の順序関係を推移律を用いて考えると，$e_1 < e_2 < e_3 < e_4$ となる．

問題となるのは例えば図 1 のイベント e_3，e_4，e_5，e_6 の順序関係である．$e_3 < e_4$ や $e_3 < e_5 < e_6$ は成立する．$e_3 < e_4 < e_5 < e_6$ は e_4 と e_5，e_4 と e_6 には順序関係がないため，成立しない．このように MSC のイベント同士の順序関係は半順序関係であり，全順序関係とは限らない．

2 つのイベント e，e' で $e < e'$ が成立するとき，e は e' と連鎖があるといい，そうではないとき，e は e' と連鎖がないという．例えば図 1 では，e_3 は e_4，e_5，e_6 のそれぞれと連鎖があるが，e_4 は e_5，e_6 のそれぞれと連鎖がない．

2 つの MSC C_1，C_2 について，プロセスが同一で, かつ共通のイベントを持たないとき C_1 と C_2 を連結してできる MSC を $(C_1; C_2)$ と書く. このとき, $(C_1; C_2)$ は次のようになる.

- $(C_1; C_2)$ のプロセスは C_1，C_2 と同一である.
- $(C_1; C_2)$ は，C_1 と C_2 に含まれる全てのメッセージを含む.
- C_1 の任意のイベント e と C_2 の任意のイベント e' が同じのプロセスにあるとき，$(C_1; C_2)$ では $e < e'$ が成立する.

Vol.2010-SLDM-144 No.63
Vol.2010-EMB-16 No.63
Vol.2010-MBL-53 No.63
Vol.2010-UBI-25 No.63
2010/3/28

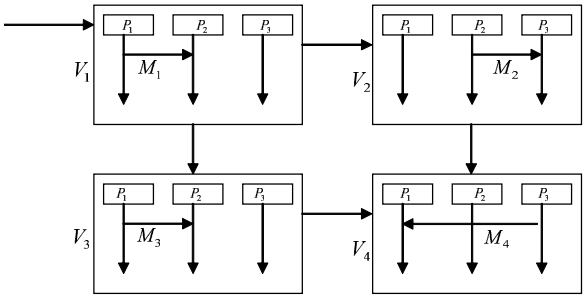

図 2 HMSC の例

2.2 HMSC

ハイレベル・メッセージ・シーケンス・チャート (High-level Message Seaquence Chart: HMSC) とは 1 つの頂点に 1 つの MSC を対応させた有向グラフである．図 2 は HMSC の一例である．これを用いて HMSC について説明する．

図 2 の V_1，V_2，V_3，V_4 がグラフの頂点にあたる．各頂点はそれぞれ 1 つの MSC に対応しており，HMSC 内のすべての MSC は共通のプロセスを持つ．始点が頂点にない辺の終点にある頂点が初期頂点であり，その HMSC で最初に処理が行われる頂点を示している．図 2 では頂点 V_1 が初期頂点である．また，図 2 の辺 (V_2，V_4) は頂点 V_2 の MSC の後に頂点 V_4 の MSC が続くことを示す．図 2 の頂点 V_1 のように 1 つの頂点から辺が複数出ている場合，その複数の辺のうちから 1 つが選択されることになる．例えば, 図 2 の V_1 の MSC には頂点 V_2 か V_3 のどちらかの MSC が続くことになる．HMSC の**パス**はあるシナリオに対する処理の流れを示す．例えば，図 2 においてパス (V_1，V_3，V_4) は頂点 V_1，V_3，V_4 の順序，つまり $M_1 \rightarrow M_3 \rightarrow M_4$ と処理するように思われる．しかし，パスの実行は，そのパス上の全ての MSC をパスの順序で接続された MSC と同じになる．例えばパス (V_1，V_3，V_4) に対応する MSC は図 3 の MSC と同じである．ここで問題となるのは，パスが表す順序と起こりうるメッセージの順序が異なる場合がある点である．例えばパス (V_1，V_3，V_4) は頂点 $V_1 \rightarrow V_3 \rightarrow V_4$ の順でメッセージの処理を行うことを示すが，実行順序は図 3 の MSC より，メッセージ M_1 の送信イベントはメッセージ M_4 の送信イベントと連鎖がないため，メッセージ M_4 はメッセージ M_1 よりも先に処理が始まる，つまりはパスの順序と異なる順序でメッセージが生起する場合がある．

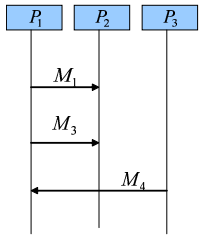

図 3 パス (V_1，V_3，V_4) に対応する MSC

2.3 ディスコード

2.2 節で述べたように，ある HMSC の頂点 V_1 から頂点 V_2 へのパスを考えるとき，V_1 に対する MSC C_1 の全てのメッセージが V_2 に対する MSC C_2 の全てのメッセージに必ずしも先行するとは限らない．そこでアレンロジックを基礎とした，連結した MSC で発生する順序の不一致の解析を可能とするディスコードを利用した方法が文献[1] で提案されている．

アレンロジック (Allen's logic) とは 2 つのメッセージの実行順序を述語で示すものである[4]．2 つのメッセージ $m_1 = (s_1, r_1)$ と $m_2 = (s_2, r_2)$ がとる可能性がある実行順序は最大で 6 種類であり，6 種類の詳細な順序とそれに対応する述語は以下の通りである．

- $\mathbf{p}$: m_1 が m_2 に先行する場合．($s_1 < r_1 < s_2 < r_2$)
- $\mathbf{p^{-1}}$: m_2 が m_1 に先行する場合．($s_2 < r_2 < s_1 < r_1$)
- $\mathbf{o}$: m_1 が m_2 に重なる場合．($s_1 < s_2 < r_1 < r_2$)
- $\mathbf{o^{-1}}$: m_2 が m_1 に重なる場合．($s_2 < s_1 < r_1 < r_2$)
- $\mathbf{d}$: m_1 が m_2 の間にある場合．($s_2 < s_1 < r_1 < r_2$)
- $\mathbf{d^{-1}}$: m_2 が m_1 の間にある場合．($s_1 < s_2 < r_2 < r_1$)

次に，ある連結された MSC $(C_1; C_2)$ について考える. 2 つのメッセージ m_1=(s_1，r_1) $\in C_1$，m_2=(s_2，r_2) $\in C_2$ の実行順序で $(C_1; C_2)$ に対する最も良いシナリオは C_1 内のメッセージが C_2 内のメッセージに先行する $m_1\mathbf{p}m_2$ であり，最も悪いシナリオは C_2 内のメッセージが C_1 内のメッセージに先行する $m_1\mathbf{p^{-1}}m_2$ である．このように $m_1\mathbf{p}m_2$，つまり $s1 < r1 < s2 < r2$ を最良とし，この順序と異なる点が多いほど悪いとしてアレンジロジックの述語に次の全順序関係 $\prec$ を与える．

- 半順序関係 $\prec_0$= { ($\mathbf{p}$，$\mathbf{o}$),($\mathbf{o}$，$\mathbf{d^{-1}}$)，($\mathbf{d^{-1}}$，$\mathbf{d}$)，($\mathbf{d}$，$\mathbf{o^{-1}}$)，($\mathbf{o^{-1}}$，$\mathbf{p^{-1}}$) } とし，$\prec_0$ の推移閉包をとったものを全順序関係 $\prec$ とする．

$\mathbf{o}$ は $s1 < s2 < r1 < r2$ なので異なる点は 1 つ，同様に$\mathbf{o^{-1}}$は 3 つ，$\mathbf{p^{-1}}$は 4 つだが，$\mathbf{d^{-1}}$と $\mathbf{d}$

Vol.2010-SLDM-144 No.63
Vol.2010-EMB-16 No.63
Vol.2010-MBL-53 No.63
Vol.2010-UBI-25 No.63
2010/3/28

は両方ともに 2 つである．ここで，$\mathbf{d^{-1}} \prec \mathbf{d}$ となっているのは, メッセージの送信の順序の方が受信の順序よりも重要であるため, $s_2 < s_1 < r_1 < r_2$ である **d** をより悪いとしている．

これを用いて，ディスコードは次のように定義されている．

- MSC $(C_1;C_2;...;C_k)$ を考える．$m_1 \in C_1$，$m_2 \in C_k$ が $m_1 R m_2$(R はアレンロジックの述語) であるとき，m_1 と m_2 の**ディスコード**とは R の中で $\prec$ による最大の述語である．例えば $m_1 \in C_1$ と $m_2 \in C_2$ が m_1**pod**m_2 であるとき，m_1 と m_2 のディスコードは **d** である．

2.4 Path with No Chain

文献[1] では 2 つのメッセージのディスコードを計算するために，2 つのイベントの連鎖がないパス (Path with No Chain: PNC) を調べ，その結果でディスコードを求めている．その手法について説明する．

PNC とは，HMSC と 2 つのイベント $e \in C$，$e' \in C'$ が与えられたとき，e から e' へ連鎖がない C から C' へのパスのことを指す．2 つのイベント e，e' についての PNC が存在するかを調べる関数を $PNC(e, e')$ と記す．e，e' が PNC を持つとき $PNC(e, e')$=1, そうでないときは $PNC(e, e')$=0 を返す.

$PNC(e, e')$ を用いて m_1=(s_1，r_1) $\in C$ と m_2=(s_2，r_2) $\in C'$ に対するディスコードを計算するには，文献[1] の命題 2 より以下のように行えば良い.

- ディスコードが **p** となるのは $PNC(r_1, s_2)$=0 の時, かつその時に限る．
- ディスコードが **o** となるのは $PNC(r_1, s_2)$=1 かつ $PNC(s_1, s_2)$=0 かつ $PNC(r_1, r_2)$=0 の時, かつその時に限る．
- ディスコードが$\mathbf{d^{-1}}$となるのは $PNC(s_1, s_2)$=0 かつ $PNC(r_1, r_2)$=1 の時, かつその時に限る．
- ディスコードが **d** となるのは $PNC(s_1, s_2)$=1 かつ C から C' への全てのパスについて, s_1 から s_2 への連鎖がある，または r_1 から r_2 へ連鎖がある時, かつその時に限る．
- ディスコードが$\mathbf{o^{-1}}$となるのは $PNC(s_1, r_2)$=1 かつ，s_1 から s_2 へ連鎖がなく，かつ r_1 から r_2 への連鎖がない C から C' へのパスが存在する時，かつその時に限る．
- ディスコードが$\mathbf{p^{-1}}$となるのは $PNC(s_1, r_2)$=0 の時，かつその時に限る．

$m_1 = (s_1, r_1) \in C$ と m_2=(s_2，r_2) $\in C'$ のディスコードが **d** または$\mathbf{o^{-1}}$かどうかを求めるためには $PNC(e, e')$ を用いる他に，s_1 から s_2 へ連鎖がなく，かつ r_1 から r_2 へ連鎖がない C から C' へのパスが存在するかを調べればよい．そのようなパスがある時 1，ないとき 0 となる関数を $PNCP(m1, m2)$ と記す．$PNCP(m1, m2)$ を用いると，

- $PNCP(m1, m2)$=0，かつ PNC(s_1，s_2)=1 の時かつその時に限りディスコード=**d**
- $PNCP(m1, m2)$=1，かつ PNC(s_1，r_2)=0 の時かつその時に限りディスコード=$\mathbf{o^{-1}}$

である．

3. 提案アルゴリズム

2.4 節で $PNC(e, e')$ を用いてディスコードを計算する方法について述べたが，文献[1] の手法では頂点数が n，辺の数が $|E|$，プロセス数が $|P|$ の HMSC に対して $PNC(e, e')$ の計算量が $O(n|E|2^{4|P|}|P|^2)$ になることが示されている．すなわち，「プロセス数が大きい HMSC で膨大な時間がかかってしまう」という問題がある．そこで，任意の HMSC を論理式にエンコードすることで，2.4 節で述べた関数 $PNC(e, e')$ を充足可能性判定によって計算する方法を提案する．充足可能性判定は SAT ソルバを用いることで高速で解ける可能性がある．提案手法では関数 $PNC(e, e')$ を SAT ソルバを用いた充足可能性判定によって計算することで，高速に計算することを目的としている．

3.1 関数 $PNC(e_S, e_D)$

2 つのイベント e_S，e_D に対して PNC を判定する関数 $PNC(e_S, e_D)$ の計算をブール式の充足可能性判定で行う方法を示す．具体的には，$PNC(e_S, e_D)$=1 のとき，その時に限り充足可能となる論理式を構成する．連鎖のないパスの有無を調べるには単純閉路と単純経路だけを考えれば十分である (文献[1] の主張 1 を参照) ので，関数 $PNC(e_S, e_D)$ では単純閉路と単純経路だけを扱う．

関数 $PNC(e_S, e_D)$ の入出力は次の通りである．

- 入力：HMSC H と，2 つのイベント (e_S, e_D)
- 出力：PNC があるなら 1，そうでないなら 0

n 個の頂点からなる HMSC の各頂点に対応する MSC $C_i (i = 1, ..., n)$ 上の各プロセス $j (j = 1, ..., m)$ について，j 中の全てのイベントに先行する min イベント $e^{\min}_{i,j}$ および j 中の全てのイベントの後にある max イベント $e^{\max}_{i,j}$ を新たに導入する．

イベント e_S に対応する論理変数を S，e_D に対応する論理変数を D とする．イベント $e^{\min}_{i,j}$ に対応する論理変数をそれぞれ $gmin^k_{i,j} (k = 1, ..., n)$ と書く．同じように，イベント $e^{\max}_{i,j}$ に対応する論理変数をそれぞれ $gmax^k_{i,j} (k = 1, ..., n)$ と書くことにする．また，HMSC のそれぞれの頂点 $v_i (i = 1, .., n)$ に対応する論理変数をそれぞれ $b^k_i (k = 1, ..., n)$ と書く．さらに，頂点 v_i が $v_{i1}, v_{i2}, ..., v_{ip_i}$ を子として持つとき，それぞれの辺に対応する論理変数をそれぞれ $t^k_{i,iq}$ $(q = 1, .., p_i)$, $(k = 1, ..., n)$ と書く．直観的には論理式が充足するとき，HMSC 上

Vol.2010-SLDM-144 No.63
Vol.2010-EMB-16 No.63
Vol.2010-MBL-53 No.63
Vol.2010-UBI-25 No.63
2010/3/28

の長さ k のパスと対応付けられて各論理変数には次のように値が割り当てられる．

- 論理変数 S の値は常に 1
- 論理変数 D の値は常に 0
- 論理変数 $gmin^k_{i,j}$ は頂点 v_i がパスの k 番目にあり，かつ $(e_S, e^{\min}_{i,j})$ で連鎖があるなら 1，$(e^{\min}_{i,j}, e_D)$ で連鎖があるなら 0，それ以外の場合はドントケア
- 論理変数 $gmax^k_{i,j}$ は頂点 v_i がパスの k 番目にあり，かつ $(e_S, e^{\max}_{i,j})$ で連鎖があるなら 1，$(e^{\max}_{i,j}, e_D)$ で連鎖があるなら 0，それ以外の場合はドントケア
- 論理変数 b^k_i は v_i がパスの k 番目にあるなら 1，そうでないなら 0
- 論理変数 $t^k_{i,iq}$ は v_i がパスの k 番目にあり，かつ v_{iq} がパスの $k+1$ 番目にあるなら 1, そうでないなら 0

$PNC(e_S, e_D)$ において構成される論理式は，e_S を含む頂点 v_S から e_D を含む頂点 v_D までのパスと，そのパス上のイベントに対する e_S からの連鎖の有無を表現している．v_S から数えて k 番目の頂点とその頂点からの遷移枝についての論理条件を記述するために，k の上添え字を持つ論理変数を用いている．以下で，論理式のエンコードの詳細について述べる．

論 理 式 I

パスに関わらず論理変数の値が決められる，イベント e_S と e_D に対する論理変数と，イベント e_S と e_D を含む頂点に対する論理変数について扱う論理式 I を考える．論理変数 S は常に 1 で論理変数 D は常に 0 とする．

$$S \wedge \neg D \tag{1}$$

さらに，e_S を含む頂点を v_S，e_D を含む頂点を v_D とするとその 2 つの頂点は必ずパス上にあり，かつ v_S はパスの 1 番目，v_D はパスの終端にあるので，パスの長さを k として，

$$b^1_S \wedge b^k_D \tag{2}$$

とする．式 (1) と (2) の論理積をとった式を I とする．

$$I = S \wedge \neg D \wedge b^1_S \wedge b^k_D \tag{3}$$

論 理 式 S

頂点間の遷移関係を表す論理式 $S^k_i (k = 1, ..., n)$ を考える．パスの終端以外の頂点では，次の頂点への辺が必ず一つだけパス上に含まれる．それを示す論理式 S^k_{1i} は第 k 番目に頂点 v_i が選択されていればすなわち (b^k_i=1 ならば)，v_i から出ている辺のどれか一つのみが選択される ($t_{i,iq}$=1) ときかつその時に限り充足されるよう構成する．

$$S^k_{1i} = (b^k_i \Rightarrow (\bigvee_{q=1,...,p_i} (t^k_{i,iq} \bigwedge_{r \neq q} \neg t^k_{i,ir}))) \tag{4}$$

ある辺がパス上にあるならその辺の先にある頂点もパス上にある．その制約を示す論理式 S^k_{2i} は第 k 番目の頂点 v_i からの辺 (v_i, v_{iq}) が選ばれていれば ($t^k_{i,iq} = 1$ ならば)，v_{iq} のみが第 $k+1$ 番目の頂点として選択される (b^{k+1}_{iq} =1) ときかつその時に限り充足されるよう構成する．

$$S^k_{2i} = \bigwedge_{q=1,...,p_i} (t_{i,iq} \Rightarrow (b^{k+1}_{iq} \wedge \bigwedge_{l \neq iq} \neg b^{k+1}_l)) \tag{5}$$

式 (4) と (5) の論理積をとったものが S^k_i である．

$$S^k_i = S^k_{1i} \wedge S^k_{2i} \tag{6}$$

全ての頂点に関して (6) の論理積をとったものが S^k である．

$$S^k = \bigwedge_{i=1,...,n} S^k_i \tag{7}$$

論 理 式 B

次に論理式 B^k_i を次のように構成する．パス上の隣接する頂点 v_i と v'_i に対応する MSC は連接されることになり，同じプロセス上にあるイベントである $e^{max}_{i,j}$ と $e^{min}_{i',j}$ は連鎖が成立する．以下の論理式は，パス上で k 番目の頂点 v_i から $k+1$ 番目の頂点 v'_i への辺が選択 ($t^k_{i,i'}$=1) されたとき，イベント $e^{\max}_{i,j}$ と $e^{\min}_{i',j}$ に対する論理変数が同じ値 ($gmax_{i,j}$=$gmin_{i',j}$) になるときかつその時に限り充足されるよう構成される．

$$B^k_i = \bigwedge_{i'=1,...,n} \bigwedge_{j=1,...,m} ((t^k_{i,i'} \Rightarrow (gmax^k_{i,j} \equiv gmin^{k+1}_{i',j})) \tag{8}$$

さらに全ての頂点に関して (8) の論理積をとったものが，B^k である．

$$B^k = \bigwedge_{i=1,...,n} B^k_i \tag{9}$$

論理式 SL

HMSC では，同じ頂点を 2 回以上通るパスには，そのパスよりもディスコードが大きい単純経路または単純閉路が必ず存在する．(文献[1] の主張 1 を参照) つまり，$PNC(e_S, e_D)$ は連鎖のないパスがあるかどうかを調べる際，単純閉路を除き同じ頂点を 2 回以上通るパスを調べる必要がない．この論理式は，探索しているパスが単純経路または単純閉路であることを保証する．具体的には論理式 SL はパスの k 番目 (k = 2,...,r) の頂点として v_i が選ばれているとき，$p(p < k)$ 番目の頂点は v_i ではないときかつその時に限り充足されるよう構成される．

Vol.2010-SLDM-144 No.63
Vol.2010-EMB-16 No.63
Vol.2010-MBL-53 No.63
Vol.2010-UBI-25 No.63
2010/3/28

$$SL_r = \bigwedge_{i=1,...,n} \bigwedge_{k=2,...,r} (b_i^k \Rightarrow \bigwedge_{p<k} \neg b_i^p) \tag{10}$$

論 理 式 T

論理式 T を構成する前に，まず文献[3] に記述された各イベントの半順序関係 (推移的閉包) を求めるアルゴリズムを用いて，各 MSC について，半順序関係を計算する．MSC C_i 中の任意のイベント e，e' について，次のような R_i を各 v_i に対して計算しておく．

$$(e, e') \in R_i, \Leftrightarrow e <^* e' \tag{11}$$

論理式 $T_{e_S,\max}$ はイベント e_S を含む MSC C_S において，e_S と $e^{\max}_{S,j}$ で連鎖があれば, $e^{\max}_{S,j}$ に対応する論理変数 $gmax^1_{S,j}$ が e_S に対応する論理変数 S と同じ値 ($S \Rightarrow gmax^1_{S,j'}$) になる時かつその時に限り充足する論理式である．

$$T_{e_S,\max} = \bigwedge_{(e_S, e^{\max}_{S,j}) \in R_S} (S \Rightarrow gmax^1_{S,j'}) \tag{12}$$

論理式 $T^k_{i,\min,\max}$ は各 MSC C_i において, イベント $e^{\min}_{i,j}$ と $e^{\max}_{i,j'}$ との間に連鎖があり，$e^{\min}_{i,j}$ が e_S から連鎖がある ($gmin^k_{i,j} = 1$) なら論理変数 $gmax^k_{i,j'}$ が $gmin^k_{i,j}$ と同じ値 ($gmin^k_{i,j} \Rightarrow gmax^k_{i,j'}$) になるときかつその時に限り充足する論理式である．

$$T^k_{i,\min,\max} = \bigwedge_{(e^{\min}_{i,j}, e^{\max}_{i,j'}) \in R_i} (gmin^k_{i,j} \Rightarrow gmax^k_{i,j'}) \tag{13}$$

論理式 $T^k_{\min,e_D}$ は MSC C_D において，イベント $e^{\min}_{D,j}$ と e_D との間に連鎖があり，$e^{\min}_{D,j}$ が e_S から連鎖がある ($gmin^k_{D,j} = 1$) なら論理変数 D は $gmin^k_{D,j}$ と同じ値 ($gmin^k_{D,j} \Rightarrow D$) になるときかつその時に限り充足する関数である．

$$T^k_{\min,e_D} = \bigwedge_{(e^{\min}_{D,j}, e_D) \in R_D} (gmin^k_{D,j} \Rightarrow D) \tag{14}$$

$i = 1$，...，n について式 (13) の論理積をとり，次の論理式を構成する．

$$T^k_{\min,\max} = \bigwedge_{i=1,...,n} (T^k_{i,\min,\max}) \tag{15}$$

論 理 式 F

式 (7),(9),(15) の論理式をの論理積をとり, U^k を構成する．

$$U^k = T^k_{min,max} \wedge S^k \wedge B^k \tag{16}$$

長さ m ($2 \le m \le n$) のパスについて PNC の有無を判定するため，式 (3), (7), (9), (10), (12), (14), (16) より式 F_m を構成する．

表 1 ディスコードごとの $PNC(e_D, e_S)$，$PNCP(m_1, m_2)$ の実行結果

	$PNC(r_1, s_2)$	$PNC(s_1, s_2)$	$PNC(r_1, r_2)$	$PNCP(m_1, m_2)$	$PNC(s_1, r_2)$
p	0	0	0	0	0
o	1	0	0	0	0
$\mathbf{d^{-1}}$	1	0	1	0	0
d	1	1	0	0	0
$\mathbf{o^{-1}}$	1	1	1	1	0
$\mathbf{p^{-1}}$	1	1	1	1	1

e_S が MSC C_S に含まれるイベントとすると，

$m = 2$ のとき

$$F_2 = I \wedge T_{e_S,\max} \wedge S^1 \wedge B^1 \wedge T^2_{\min,e_D} \tag{17}$$

$m > 2$ のとき

$$F_m = (I \wedge \bigwedge_{j \neq S} \neg b^1_j) \wedge T_{e_S,\max} \wedge S^1 \wedge B^1) \wedge \bigwedge_{k=2,..,m-1} U^k \wedge T^m_{\min,e_D} \wedge SL_m \tag{18}$$

論理式 F_m を SAT ソルバを用いて解き，論理式を充足させる割り当てがあれば，長さ m のパスで e_S から e_D へ連鎖のないパスが存在する．すなわち，$PNC(e_S, e_D)$=1 とする．頂点数 n のとき，m=2 から n までの論理式 F_m を SAT ソルバを用いて解き，m=2 から n までのうち一つでも F_m を充足させる割り当てがあれば，e_S から e_D へ連鎖のないパスが存在することになるので $PNC(e_S, e_D)$=1 とする．

3.2 関数 $PNCP(m_1, m_2)$

m_1=(s_1, r_1)，m_2=(s_2, r_2) とすると，$PNCP(m_1, m_2)$ はある 1 つのパスに対して s_1 と s_2 の連鎖の有無と r_1 と r_2 の連鎖の有無を一度に調べるという点のみが前節の関数 $PNC(e_S, e_D)$ と異なり，ほぼ同様に構成できる．紙面の都合上，ここでは説明を省く．

3.3 ディスコード計算

2.4 節から，メッセージ m_1=(s_1, r_1)，m_2=(s_2, r_2) の 2 つのメッセージのディスコードを計算するには，最大でも $PNC(r_1, s_2)$，$PNC(s_1, s_2)$，$PNC(r_1, r_2)$，$PNC(r_1, r_2)$，$PNCP(m_1, m_2)$ の 5 つを実行すれば求めることが可能である．表 1 はディスコードの計算結果に対する各関数の実行結果である．

ディスコード計算においては $PNC(e1, e2)$ を実行する回数が少ないほど全体の実行時間は短くなる．ディスコード計算の結果は { **p**, **o**, $\mathbf{d^{-1}}$, **d**, $\mathbf{o^{-1}}$, $\mathbf{p^{-1}}$ } のうちのいずれか 1 つになるが，ここでは, ディスコード計算の結果が同じ確率でこの 6 つのうちの 1 つになると仮定して，ディスコード計算に必要な時間ができるだけ短くなるよう考える．

Vol.2010-SLDM-144 No.63
Vol.2010-EMB-16 No.63
Vol.2010-MBL-53 No.63
Vol.2010-UBI-25 No.63
2010/3/28

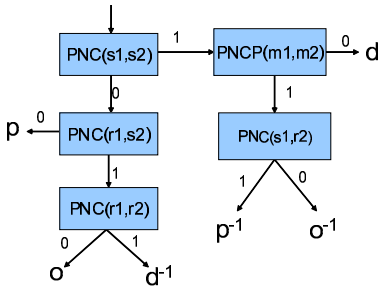

図 4 ディスコード計算の流れ

より短い時間でディスコードを計算するために図 4 の手順で $PNC(e, e')$ と $PNCP(m_1, m_2)$ を実行することにした．図 4 は m_1，m_2 のディスコードを求める場合の処理の流れを示している．まず，$PNC(s_1, s_2)$ を実行し，$PNC(s_1, s_2)$=0 なら次に $PNC(r_1, s_2)$ を実行し，$PNC(s_1, s_2)$=1 なら次に $PNCP(m_1, m_2)$ を実行する，といった順序で処理を行うことを示している．

4. 実　　験

本章では 3 章で述べた提案手法と文献[1] に記述された手法の 2 つの方法を Java プログラムで実装し，HMSC でのディスコード計算を行い， 結果を比較する．実験環境は以下のとおりである．

- CPU: Intel Core 2 Duo T8300 2.4GHz
- メモリ: 2.99GB
- OS: WindowsXP

3 章で述べたように提案した手法では SAT ソルバを利用してディスコードを計算する．SAT ソルバには minisat[7] や GRASP[5]，zchaff[6] などが存在するが，この実験では minisat を用いた．実験ではディスコードを計算するために人工的に HMSC を作成した．各 MSC はメッセージを 1 つか 2 つ含んでいる．

4.1 実　験　1

プロセス数が同じで頂点数が異なるいくつかの HMSC において，2 つの方法でディスコード計算を行い，実行時間を測定した．表 2 および図 5 はプロセス数が 12 で頂点数が異なる 5 つの HMSC で，ディスコードが **p** である 2 つのメッセージに対して，ディスコード計算を行った結果である．

表 2 実験 1(プロセス数を 12 で固定)

頂点数	プロセス数	実行時間 (提案手法)	実行時間 (既存手法)
10	12	0.578 秒	132.469 秒
20	12	3.672 秒	416.438 秒
30	12	13.094 秒	956.793 秒
40	12	29.250 秒	1462.171 秒
50	12	61.344 秒	2461.891 秒

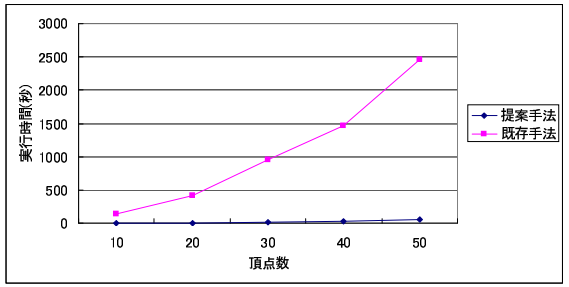

図 5 実験 1 の実行時間の変化

4.2 実　験　2

頂点数が同じでプロセス数が異なるいくつかの HMSC において，2 つの方法でディスコード計算を行い，実行時間を測定した．表 3 および図 6 はディスコードが **o** である 2 つのメッセージで，頂点数が 30 でプロセス数が異なる 8 つの HMSC に対してディスコード計算を行った結果である．

4.3 考　　察

プロセス数が大きいほどディスコード計算にかかる時間が長くなっているが，文献[1] の手法 (既存手法) ではプロセス数が大きい場合に特に多く時間がかかっている．プロセス数が小さい場合では既存手法の方が短い時間でディスコードを計算できているが，プロセス数が大きい場合は提案手法の方が短い時間でディスコード計算ができており，プロセス数が大きくなるほど 2 つの方法の実行時間の差が大きくになっている．また，頂点数が大きい場合ほどディスコード計算にかかる時間が長くなっているが，既存手法の方が多く時間がかかっており，頂点数が大きいほど 2 つの方法での実行時間の差が大きくなっている．これは既存手法では連鎖を調べる際に，全ての頂点について $2^{2|P|}$ 通り ($|P|$ はプロセス数) を調べるた

Vol.2010-SLDM-144 No.63
Vol.2010-EMB-16 No.63
Vol.2010-MBL-53 No.63
Vol.2010-UBI-25 No.63
2010/3/28

表 3 実験 2(頂点を 30 で固定)

頂点数	プロセス数	実行時間 (提案手法)	実行時間 (既存手法)
30	5	9.610 秒	0.109 秒
30	6	10.109 秒	0.344 秒
30	7	10.750 秒	1.265 秒
30	8	11.203 秒	4.938 秒
30	9	11.828 秒	20.109 秒
30	10	12.343 秒	81.297 秒
30	11	13.000 秒	326.016 秒
30	12	13.734 秒	1316.657 秒
30	13	14.266 秒	5236.953 秒
30	14	14.765 秒	20875.312 秒

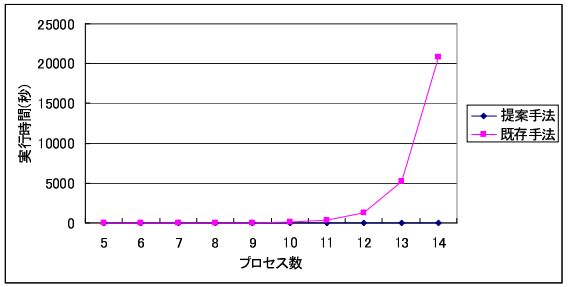

図 6 実験 2 の実行時間の変化

め，プロセス数が多い HMSC では時間がかかっていると考えられる．一方，提案手法ではパスを決めると論理式から推論によって連鎖の有無を導き出せるため，各頂点で連鎖の仕方を確かめる必要がない．そのため, 連鎖の有無を調べるのに比較的時間がかからず, プロセス数によって実行時間があまり変化しないと考えられる．

5. ま と め

HMSC のメッセージ間の実行順序をより短い時間で解析することを目的とし, HMSC を論理式にエンコードして，SAT ソルバを利用してディスコードを計算する手法を提案した．また，文献[1] の手法と，提案した手法の 2 つの手法でディスコードを求めた場合の計算時間の比較を行った．実験の結果，頂点数やプロセス数が大きい HMSC では提案した手法の方がより短い時間でディスコード計算を行えるということが分かった．

今後の課題として，本手法の計算量の解析やメッセージに条件式や代入文を追加した HMSC でもディスコードを計算できるように拡張する，などが考えられる．これが実現可能になれば，さらに幅広く HMSC に対する実行順解析を行うことができ，検証にかかる時間の削減に貢献できると考えられる．

参 考 文 献

1) E. Elkind,B. Genest, D. Peled, P. Spoletini: “Quantifying the Discord: Order Discrepancies in Message Sequence Charts,”ATVA2007.
2) A. Muscholl, D. Peled, ”Message Sequence Graphs and Decision Problems on Mazurkiewicz Traces,” In MFCS ’ 99, pp.81-91, 2001.
3) Rajeev Alur, Gerard J. Holzmann, Doron Peled: “An Analyzer for Message Sequence Charts”, Software Concepts and Tools, Vol.17, No.2, pp.70-77, 1996.
4) James F. Allen, ”Maintaining Knowledge about Temporal Intervals,” Communications of ACM, vol.26, No.11, pp.832-843,1983.
5) J.P. Marques-Silva and K.A. Sakallah, ”GRASP New Search Algorithm for Satisfiability,” Digest of IEEE Int’l Conf. Computer-Aided Design (ICCAD), pp.220-227, San Jose, Calif., 1996.
6) Y. S. Mahajan, Z. Fu, and S. Malik, “Zchaff2004: An efficient sat solver,” Lecture Notes in Computer Science, Theory and Applications of Satisfiability Testing, 7th International Conference, Invited Paper, vol.3542, pp.360-375, 2005.
7) MiniSat, http://www.cs.chalmers.se/Cs/Research/FormalMethods/MiniSat/MiniSat.html
8) Telecommunication Standardization sector of International Telecommunication Union, Z.120 Message Sequence Chart, 2004.
9) OGM, ”OMG United Modeling Language (OMG UML) Infrastructure V2.1.2,” OMG Available Specifcation, 2007