# メッセージリプレイのためのフロー情報合成とシステムテスト改善への応用

森拓郎[†1]　小川秀人[†1]

**概要**：計算機システムの複雑化に伴い，計算機間の連携を確認するシステムテストはテスト対象の周辺機器の手配および操作コストにより高コストとなる．この解決のため，報告者はネットワーク上で収集したパケット情報を用いるメッセージリプレイによる周辺機器のシミュレートについて検討している．シミュレートのためには，時系列での再現のみならず分岐やループ処理の再現が必要である．そこで，複数のフロー情報を解析して一つの合成フロー情報を出力するフロー情報合成技術を開発した．本技術により，複数のパケット情報からシミュレートのためのフロー情報を自動的に構築できる．実データを用いた試行では，開発技術によりループ部分が抽出され，シミュレーションに利用できることを確認した．今後の課題は情報解析の詳細化による精度向上および処理効率化である．

**キーワード**：フロー情報合成，テスト，リプレイ，オートマトン

# Flow Composition Technique for Message Replay Testing

TAKURO MORI[†1]　HIDETO OGAWA[†1]

## 1. はじめに

情報システムの構築では，テストに費やすコストが製品の初期開発コストの 45%を占める[1]ため，開発コスト削減のためにはテストコストの削減が必須課題である．特に複数の計算機ノードが連携するシステムテストでは，テスト対象であるシステムの動作が単体テストと比較して複雑となるためテスト件数の増大や問題解析の困難化，修正後の確認における再現性の低下などコスト増加の要因が発生しやすい．

そこで，エンタープライズ分野では履歴情報に基づくテストの自動実行ソリューションが盛んに開発されている．CA 社は Web サーバおよびクライアントの模擬とリプレイ[2]，Oracle 社は DB サーバの模擬[3]，Test Plant 社は GUI のリプレイ[4]を対象とした製品を展開している．OSS によるツールでは， Selenium[5]および Robot Framework[6]が Web クライアントの模擬とリプレイ，Sikuli Script[7]が GUI のリプレイ技術として開発されている．

報告者らは，計算機システムのうち，通信に独自プロトコルを採用しているシステムのテスト自動化を実現するために，メッセージリプレイによるテストに着目し，通信履歴情報であるパケット情報を入力情報としたメッセージリプレイ技術を開発し，メッセージリプレイソフトウェア PARROT(Packet Analysis and Respondent Replay Operation Toolset)を実現した．

図 1 および図 2 に例を示す．PARROT の動作は二つのフェイズに分かれている．解析フェイズでは入力したパケ

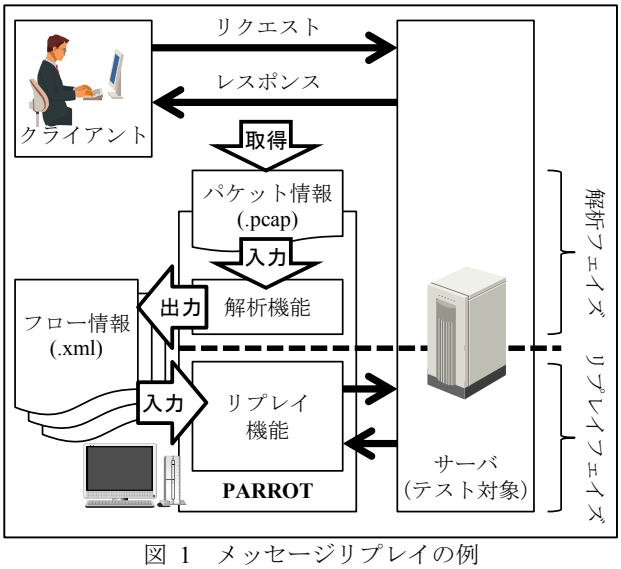


図 1　メッセージリプレイの例

```
<message id="request">
  <protocol name="ethernet" order="0">
    <src>00:11:22:33:44:55</src>
    <dst>aa:bb:cc:dd:ee:ff</dst>
  </protocol>
  <protocol name="ipv4" order="1">
    <src>192.168.0.100</src>
    <dst>192.168.0.200</dst>
  </protocol>
  <protocol name="tcp" order="2">
    <src>12345</src>
    <dst>54321</dst>
  </protocol>
  <protocol name="ap" order="3">
    <data>"application data"</data>
  </protocol>
</message>
```

図 2　フロー情報ファイルの例

†1 (株)日立製作所
Hitachi Ltd.

ット情報を解析してリプレイシナリオに相当するフロー情報を出力する．リプレイフェイズでは入力したフロー情報に基づき通信をリプレイする．2014 年には，本ツールに計算機システムのクライアント端末を模擬させることで，システムテストにおける 300 万件のテストを自動実行してテスト工程のコスト削減を実現した[8].

## 2. メッセージリプレイの問題

PARROT の利用においては，入力情報であるフロー情報の作成作業が煩雑となり，リプレイフェイズにおけるボトルネックとなる問題が発生する．

例として，クライアント・サーバシステムにおいて，クライアントを PARROT により模擬して，サーバに対して負荷テストを実施するケースを考える．PARROT には，サーバが正常に応答した場合と，タイムアウトした場合で異なる処理を実施させたいが，サーバの応答がどちらとなるかは実際にリプレイを行うまで分からない．そこで，正常応答したフロー情報と，タイムアウトでのフロー情報を組み合わせ，条件分岐を加えたフロー情報を作成して PARROT に入力する必要がある．このフロー情報は手作業で作成するほかなく，リプレイテストにおけるボトルネックとなる．

## 3. 課題

本稿では，問題解決のために，フロー情報合成技術を提案し，フロー情報合成アルゴリズムの開発を課題とする．フロー情報合成とは，本稿により定義するフロー情報処理技術であり，複数のフロー情報を入力として共通部分や分岐部分を抽出し，一つの合成フロー情報を構築する．本技術が確立すれば，PARROT はこれまでの単一のフローを再現するツールから，複数のフローをシミュレートする装置シミュレータへの発展が期待できる．

次章以降では，フロー情報合成アルゴリズムと，実際の計算機システムを用いて有効性を検証した結果を示す．

## 4. フロー情報および要件

本章では，フロー情報の定義およびフロー情報合成の要件について示す．本稿ではフロー情報とその関連情報について以下のように定義する．フロー情報とは，計算機ノードにおける，メッセージ送受信もしくは時間経過に伴う状態の変化を示す情報であり，有限オートマトンでして表現する．時間経過やメッセージ受信といったメッセージ送信の契機をトリガ，メッセージ送信およびメッセージの更新をアクションとする．本稿におけるフロー情報 Flow は，一般的な有限オートマトンの定義[9]におけるラベルの集合 Σをトリガおよびアクションに分け，以下の条件を満たす七つ組みで表す．

$Flow=<T,A,S,S_0,F,\delta,\omega>$

T　：トリガの集合
A　：アクションの集合
S　：状態の集合
$S_0$　：初期状態の集合 ($S_0 \in S$)
F　：終了状態の集合 ($F \in S$)
$\delta$　：状態遷移関数 ($S \times T \rightarrow S$)
$\omega$　：出力関数 ($S \times T \rightarrow A$)

Flow の動作を表す，トリガ t とアクション a の二つ組みラベルの順序つき集合をフロー情報の語(w)と定義する．w の i 番目のラベルを w(i)として(t/a)で表す．フロー情報 Flow によって受理される全ての語の集合を受理言語として Lang(Flow)と定義する．すなわち $w \in Lang(Flow)$である．

本稿の目的であるフロー情報の合成処理 Comp は，二つのフロー情報「受容フロー情報($Flow_{dst}$)」および「入力フロー情報($Flow_{src}$)」を入力として，二つのフロー情報の差異を抽出して，一致する部分は統合し，異なる部分は分岐を作成することで，両方のフロー情報の受理言語を受理可能な合成フロー情報 $Flow_{cmp}$ を構築することである．なお構築された合成フロー情報には，余分な語が発生しても良いとする．

$Flow_{cmp} = Comp(Flow_{dst}, Flow_{src})$
$\forall w_{dst} \in Lang(Flow_{dst})$ $(w_{dst} \in Lang(Flow_{cmp}))$
$\forall w_{src} \in Lang(Flow_{src})$ $(w_{src} \in Lang(Flow_{cmp}))$
$Lang(Flow_{dst}) \cup Lang(Flow_{src}) \subseteq Lang(Flow_{cmp})$

受理は必要条件であり，十分条件ではない．次節以降では，その他の合成フロー情報に必要な要件について述べる．

### 4.1 要件:分岐抽出

二つのフロー情報 $Flow_{dst}$ と $Flow_{src}$ から分岐部分を抽出した合成フロー情報 $Flow_{cmp}$ を構築する定義について示す．まず，$Flow_{dst}$ と $Flow_{src}$ における語 $w_{dst}$ および $w_{src}$ について，対応する状態 $S_{cmp}$，状態遷移関数 $\delta_{cmp}$，出力関数 $\omega_{cmp}$ を構築する．

$(s_{cmp1},t_{dst},s_{cmp2}) \in \delta_{cmp}$　$(s_{cmp1},t_{dst},a_{dst}) \in \omega_{cmp}$
$(s_{cmp3},t_{src},s_{cmp4}) \in \delta_{cmp}$　$(s_{cmp3},t_{src},a_{src}) \in \omega_{cmp}$
$s_{cmp1},s_{cmp2},s_{cmp3},s_{cmp4} \in S_{cmp}$
$w_{dst}(i)=(t_{dst} / a_{dst})$　$(w_{dst} \in Lang(Flow_{dst}))$
$w_{src}(j)=(t_{src} / a_{drc})$　$(w_{src} \in Lang(Flow_{src}))$
$t_{dst} \in T_{dst} \in T_{cmp}$　$t_{src} \in T_{src} \in T_{cmp}$
$a_{dst}=\omega_{dst}(s_{dst} \in S_{dst},t_{dst})$ $\in A_{dst} \in A_{cmp}$
$a_{src}=\omega_{src}(s_{src} \in S_{src},t_{src})$ $\in A_{src} \in A_{cmp}$

次に，$w_{dst}$ および $w_{src}$ に含まれるラベルの対のうち同一である対を抽出する．同一とは，ラベルのトリガおよびアクションがそれぞれ等しい，すなわち $w_{dst}(i)=w_{src}(j) \equiv t_{dst}=t_{src} \wedge a_{dst}=a_{src}$ と定義する．同一であるラベル対について，その遷移元状態は合成して一つの状態 $s_{cmp\_from}$ とする．遷移先状態は合成して状態 $s_{cmp\_to}$ とする．この合成した状態について，ラベル対に定義されているトリガは状態遷移関数 $\delta_{cmp}$ に，アクションは出力関数 $\omega_{cmp}$ に設定する．

$t_{cmp}=t_{dst}=t_{src}$　$(t_{cmp} \in T_{cmp})$

$a_{cmp}=a_{dst}=a_{src}$　　$(a_{cmp}\in A_{cmp})$

$(s_{cmp\_from},t_{cmp},s_{cmp\_to})\in \delta_{cmp}$

$(s_{cmp\_from},s_{cmp\_to}\in S_{cmp})$

$(s_{cmp\_from},t_{cmp},a_{cmp})\in \omega_{cmp}$

合成条件を満たさない対が残った場合は分岐として抽出する．図 3 に分岐抽出を行うフロー情報の例を示す．このフロー情報では，フローA とフローB の合成により，$S_{A1}$ と $S_{B1}$，$S_{A2}$ と $S_{B2}$，$S_{A3}$ と $S_{B4}$，$S_{A4}$ と $S_{B5}$，$S_{A5}$ と $S_{B6}$ が合成された結果，フローC が得られる．フローC では，$S_{C2}$ で分岐が抽出されており，二つのトリガ t2 および t5 で異なる状態へ遷移する．

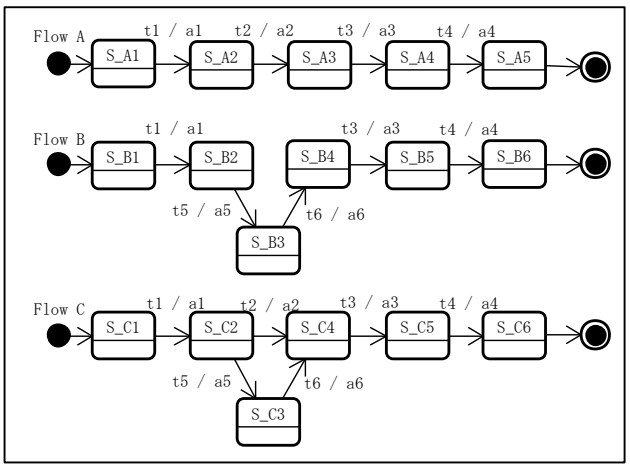


図 3　合成対象フロー情報の例 1

### 4.2 要件:ループ抽出

ループ抽出とは，あるフロー情報において，二回以上出現する部分をループとして抽出する処理である．フロー情報の語 w の i 番目から l 個のラベルを取り出し順序つき集合を作成する関数 sublist (w,i,l)を定義する．ループ抽出とは，以下を満たす i および l の抽出と定義できる．

$$sublist(w,i_1,n)=sublist(w,i_2,l)\ldots sublist(w,i_n,l)$$

ループの抽出パターンが複数考えられる場合には，長いループすなわち l が最大となるループを選択する．なぜならば，短いループを優先して抽出した場合，長いループの場合と比べて，入力したフロー情報には存在しない経路が多く発生すると考えられるためである．

入力情報の例と合成結果の例を図 4 に示す．合成フローF は，短いループを優先的に抽出した場合の合成結果であり，語{t3/a3,t5/a5}や{t1/a1,t2/a2,t1/a1,t2/a2,t3/a3,t5/a5}が受理されるが，これらはいずれも入力フロー情報であるフローD，E いずれにも存在しない経路である．一方で，長いループを優先的に抽出した場合のフロー情報 G では，このような経路の発生が少なく抑えられている．

### 4.3 要件:不正な合成の回避

フローの合成においては，合成を回避すべき状態の組が存在する．入力情報の例フローH および，成結果の例フローI を図 5 に示す．$S_{A4}$ および $S_{H5}$ が合成され $S_{I5}$ となった場合，$S_{I5}$ の持つ二つの遷移におけるトリガは t4 で同様だが，アクションおよび遷移先状態は異なる．これは非決定的なオートマトンであるため，メッセージリプレイのためのシナリオとしては不適切である．そのため本稿においては，$S_{I5}$ のような合成の回避が必要である．例では，問題の合成は $S_{A4}$ と $S_{H5}$ であるが，この合成へ至ったのはその前の状態 $S_{A3}$ と $S_{H4}$ の合成による，二つのフローの合流に起因する．そこで，$S_{A3}$ と $S_{H4}$ および $S_{A4}$ と $S_{H5}$ の合成を回避できれば，図 5 に示すフローJ が得られる．フローJ には不正な合成は存在せず，本稿の要件を満たす．

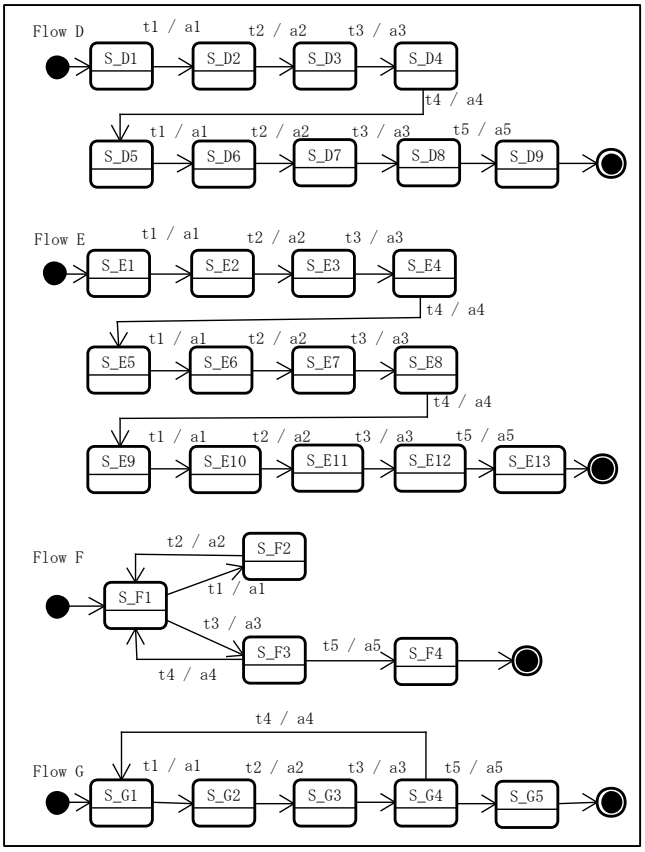


図 4　合成対象フロー情報の例 2

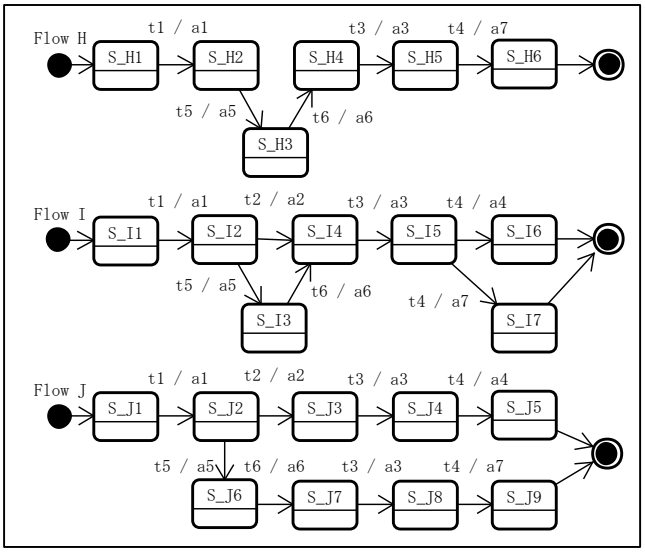


図 5　不正なフロー合成およびその回避の例

## 5. 関連研究

文からの正規文法の抽出や，状態遷移系のような分岐や順序関係を持つ情報の統合技術は様々に検討されている．

Lorenzoli ら[10]は 4.3 節で示したトリガが同様な遷移について，アクションが操作するパラメータに着目して丸め込むアプローチで状態の合成を行った．このような手法は計算機の動作の解析において有効な一方で，本稿の目的であるリプレイにおけるシナリオの作成という視点では，アクションの内容が一意に決まらないため課題が残る．

van der Aalst ら[11]は，ワークフローのログ情報からワークフローモデルを抽出する技術を Workflow Mining と位置付け，ERP(Enterprise Resource Planning)や，CRM(Customer Relationship Management)の情報解析を提案した．この提案では，順序性の伴わない並列実行可能なプロセスの記述を用いて現実のワークフローにより近い抽出を実現した．一方で，提案手法は解析対象となる並列実行タスクの数 N に対して計算量が O(N!)となるため，タスク数の増大に伴い計算量が増大する課題が残る．

フロー情報のような情報を合成するためには情報の同一性を検査するマッチング技術が必要である．文字列のマッチングについてはツリー構造を応用した Karp ら[12]の手法が古典的に知られている．橋本ら[13]はツリー構造をソースコードの変化の解析に応用しており，大規模な情報への適用へ有効であることが示されている．

これらを参考とし，本稿の次章以降で示す手法では，大規模なフロー情報の合成を前提として，文字列の解析に関するアルゴリズムに着想を得て開発した．

## 6. フロー情報合成アルゴリズム

4 章で述べた要件を満たす解決アプローチとして，レーベンシュタイン距離に着目した．レーベンシュタイン距離とは，ある文字列 S1 と文字列 S2 が与えられた際に，S1 から S2 への変更に必要な編集作業（追加，削除）の回数である．動的計画法により，O(|S1|×|S2|)の計算量でレーベンシュタイン距離および編集作業が算出できることが知られている[14]. 本稿で扱う二つのフロー情報の合成では，フロー情報は一次元の情報であるため文字列の比較処理応用可能であり，またある程度のフロー情報の規模増大にも対応可能であると考えた．

また他の解決手法の候補として，文字列の検索で用いられる Suffix Array[15]が考えられたが，Suffix Array は計算量やメモリ使用量の点で優れループ検出には有効である一方で，4.3 節で示した不正な合成の回避するために必要な状態に関する情報の扱いが困難であると考え，不採用とした．

### 6.1 フロー合成手順グラフの作成

本手法では，二つのフロー情報の合成を考える．二つのフローを受容フロー$Flow_{dst}$と入力フロー$Flow_{src}$とし，フロー合成処理とは受容フローに対して入力フローを合成する作業であると定義する．

フロー合成はフロー合成の手順を導出する有向グラフ「合成手順グラフ」を用いる．合成手順グラフのノードである合成手順ノードは合成作業の状態過程を示し，$Flow_{dst}$と $Flow_{src}$の各状態の組み合わせと定義する．合成手順ノードを得る関数を $Node(s_{dst},s_{src})$と定義する．

合成手順グラフのエッジである合成手順エッジは合成処理における作業である．合成手順エッジは作業内容に応じた三種の種別いずれかを設定する．種別はレーベンシュタイン距離の算出における文字列編集作業である追加, 削除, 置換と対応する追加エッジ $E_{add}$，パスエッジ $E_{pas}$，合成エッジ $E_{cmp}$と定義する．各エッジは{遷移元状態,イベント,遷移先状態}で表す．

追加エッジ $E_{add}$は，分岐を新たに作成する処理である．合成手順ノードの入力フロー側に遷移が存在する場合に作成する．

パスエッジ $E_{pas}$は，文字列編集作業では文字の削除に該当するが，フロー合成では受容フローの状態を削除する必要がないため，パスエッジはエッジは存在するが処理は何も行わない．合成手順ノードの受容フロー側に遷移が存在する場合に作成する．

合成エッジ $E_{cmp}$は，文字列比較の置換に相当し，二つの状態を同一であると判断して合成処理すなわち状態の合流を行う．レーベンシュタイン距離のアルゴリズムでは，エッジの一致条件に文字要素の一致を用いるが，本アルゴリズムではトリガおよびエッジの一致とする．

図 6 に各エッジを作成する擬似コードを示す．図 3 で示したフローA および B を用いて，合成手順グラフを作成した例を図 7 に示す．

### 6.2 フロー合成経路の算出と合成

このグラフにおいて，左上の合成前状態から右下の合成終了状態に対する最もコストの低い経路を算出することで合成手順が得られる．エッジのコストは，追加エッジを $C_{add}$，パスエッジを $C_{pas}$，合成エッジを $C_{cmp}$とすると，以下の関係が成り立つように設定する．これは，処理の優先度を合成→パス→追加の順序とするためである．

$$0 < C_{cmp} < (C_{pass} \times |S_{dst}|) < C_{add}$$

コストを設定し，動的計画法による最短経路を導出した例を図 7 の 1〜6 に示す．導出した最短経路に 6.1 節で示したエッジごとの処理を適用すると，合成処理の手順として以下のように取り出せる. この処理をフローA に施すと，合成結果としてフローC が得られる．

1. 合成エッジ　($S_{A1}$ と $S_{B1}$，$S_{A2}$ と $S_{B2}$ を合成)
2. 追加エッジ　(Node($S_{A2}$,$S_{B2}$)に $S_{B3}$ を追加)
3. 追加エッジ　(Node($S_{A2}$,$S_{B3}$)に $S_{B4}$ を追加)
4. パスエッジ
5. 合成エッジ　($S_{A3}$ と $S_{B4}$，$S_{A4}$ と $S_{B5}$ を合成)

6. 合成エッジ　　($S_{A4}$ と $S_{B5}$，$S_{A5}$ と $S_{B6}$ を合成)

本節で述べた手法により，4.1 で示した分岐抽出を含むフロー合成は達成できる．

```
# 追加エッジ
function makeAddEdge{
 for sdst in Sdst{ for ssrc in Ssrc{ for tsrc in Tsrc{
    if δ src(ssrc,tsrc)!={} {
     Eadd+={Node(sdst,ssrc),{tsrc, ω (ssrc,tsrc)},
            Node(sdst, δ src(ssrc,tsrc))} }}}}}
# パスエッジ
function makePassEdge{
 for sdst in Sdst{ for ssrc in Ssrc{ for tdst in Tdst{
    Epas+={Node(sdst,ssrc),{tdst, ω (sdst,tdst)},
           Node( δ dst(sdst,tdst),ssrc)}
    Epas+={Node( δ dst(sdst,tdst),ssrc),{tdst, ω (sdst,tdst)},
           Node(sdst,ssrc)} }}}}
# 合成エッジ
function makeCompositionEdge{
 for sdst in Sdst{ for ssrc in Ssrc{
   for tdst in Tdst{ for tsrc in Tsrc{
     if δ src(ssrc,tsrc)!={} and δ src(ssrc,tsrc)!={}
     and tdst=tsrc and ω dst(sdst,tdst)= ω src(ssrc,tsrc) {
      Ecmp+={Node(sdst,ssrc),{tdst, ω dst(sdst,tdst)},
             Node( δ dst(sdst,tdst), δ src(ssrc,tsrc))} }}}}}}
```

図 6　基本的な合成手順グラフのエッジ作成

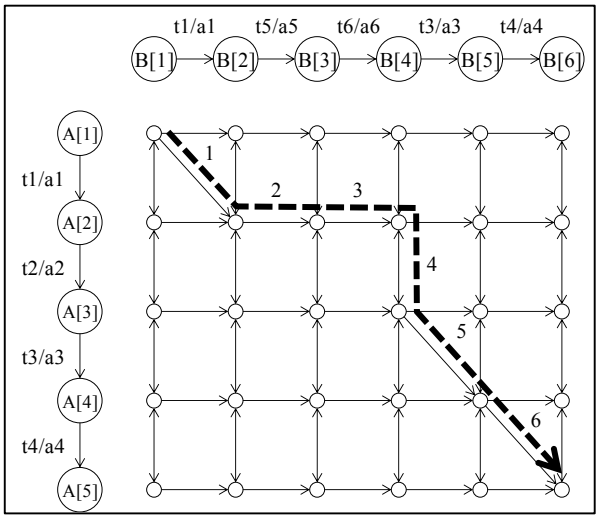


図 7　合成手順グラフ 1 および最短経路

### 6.3 三個以上のフロー情報の合成

三個以上のフロー情報を合成する場合については，フローが ABCD の四つ存在すると仮定して以下の三つの合成手法が考えられる．

第一の手法は，全てのフローを一つの合成手順グラフで合成する手法 Comp(A,B,C,D)である．二つのフロー合成では二次元であった合成手順グラフを「合成対象フローの数」次元に拡張する手法である．本方式は，合成手順グラフが二次元の場合と同様の手法で利用可能な一方で，計算量とメモリ使用量は爆発的に増加する．

第二の手法は，合成したフローをさらに合成する手法 Comp(Comp(A,B),Comp(C,D))である．合成したフローの合成とは，分岐を持つフロー同士の合成を意味する．この場合，動的計画法による合成経路の算出ができず，代わりに組み合わせ問題を解かなければならない．そのため合成の計算量は増加する．

第三の手法は，分岐のある受容フローに対して分岐のない入力フローの合成を繰り返す手法 Comp(Comp(Comp(A,B),C),D)である．受容フローに分岐が存在したとしても，入力フローには分岐が存在しければ動的計画法の適用が可能であるため，前の二つの手法のような計算量の増大は発生しない．

第一の手法は最適解を得られる唯一の手法であり，第二および第三の手法で得られる結果は局所解である．一方で大規模な適用においては第三の手法が最も効果的である．いずれの手法を選択するかは要件や入力情報により選択すべきである．

### 6.4 ループ抽出

本節以降では，4.2 節および 4.3 節で示した要件を満たすためのアルゴリズム拡張について述べる．

ループ抽出のためには，二つの処理を順に適用する．まず，受容フロー$Flow_{dst}$について，受容フロー内にループが存在するかどうか解析する（単体フローループ抽出）．次に，受容フローに対して，入力フロー$Flow_{src}$を合成することで入力フローのループを抽出する（複数フローループ抽出）．

#### 6.4.1 単体フローループ抽出

単体フローループ解析は，受容フローと入力フローに同じフロー情報を設定したフロー合成により実現する．そのためアルゴリズムの拡張を行う．図 4 のフローD の単体フローループ解析による合成手順グラフの例を図 8 に示す．図 8 の実線で示す合成経路は，受容フローの特定の経路を複数回通過しており，この経路に基づく合成処理がループ抽出となる．しかし，6.1 節で示したアルゴリズムでは破線で示す経路が最短経路となるため，そのままではループが抽出されない．そこでコストを以下のように拡張する．

ある合成エッジのコストを計算する際に，そのエッジの始点を終点とするような合成エッジ（後方合成エッジ）が存在する場合，合成エッジのコストは後方合成エッジのコストに $C_{cmpinc}$ を加えた値とする．

$\forall e_1=\{s_{1from},ev_1,s_{1to}\}\in E_{cmp}($
　$\forall e_2=\{s_{2from},ev_2,s_{1from}\}\in E_{cmp}($
　　$cost(e_1)>cost(e_2)+C_{cmpinc})))$

後方合成エッジが存在しない合成エッジのコストはこれまで通り $C_{cmp}$ とする．言い換えれば，合成エッジのコストはその後方に連続している合成エッジの個数を N とし

て，$C_{cmp}+C_{cmpinc}\times N$ とする．これに伴い，変数の関係性は以下のように拡張する．

$$0 < C_{cmp} < (C_{pass}\times|S_{dst}|) < (C_{cmp}+C_{cmpinc}) < C_{add}$$

これを反映した合成経路算出では，合成エッジのみ通過する経路と比較して，ループ処理を含む経路のコストが低くなるため，最短経路法の適用により図 8 の実線で示したループを抽出した合成経路が作成される．この合成経路に従い合成処理を行うと，図 4 のフローG が得られる．

#### 6.4.2 複数フローループ抽出

複数フローループ抽出は，単体フローループ抽出で得られた合成フロー情報を受容フローとするフロー合成により実現する．

図 4 のフローG を受容フロー，フローE を入力情報とした際の合成手順グラフを図 8 に示す．このグラフには追加エッジが存在しない．これは，入力フローであるフローE に含まれる全てのエッジが受容フローであるフローG に存在していることを示す．よってこの合成結果は変わらずフローG である．

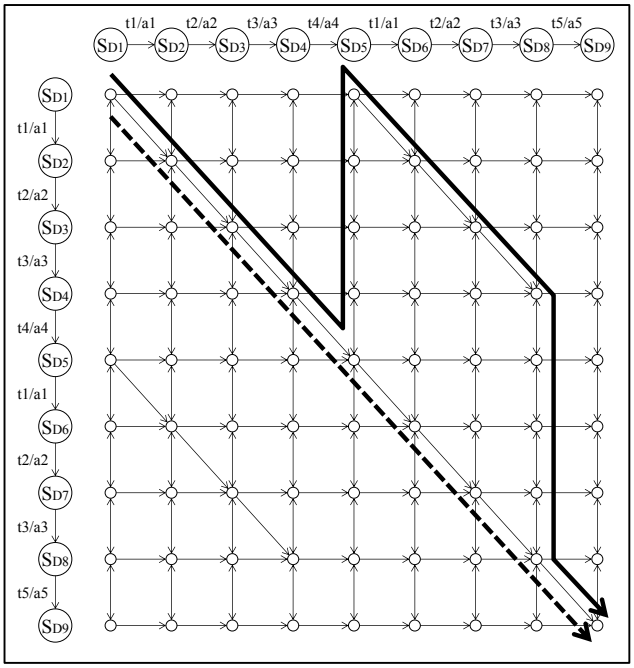


図 8　合成手順グラフの例 2

### 6.5 不正な合成の回避

4.3 節で示したような，フローによっては不正な状態遷移モデルが合成される問題は，基本的な合成手順グラフの作成においても発生する．4.3 節のフローA および H の合成手順グラフの例を図 9”Path A”に示す．

問題発生の原因は，基本的な合成手順グラフの生成アルゴリズムでは，同じ状態に存在してはならない二つの遷移が共存するような合成経路の算出が行われる点にある．本アルゴリズムのアプローチでは，この解決のために不正な合成がなされた遷移について，その合成されるまでの経路を遡り，不正な合成へ至らないよう経路を分岐させる手法を取る．そのため，この問題を解決するため合成手順グラフに次のような拡張を加える．

“「トリガは同じだがアクションが異なる」遷移が存在するノード(図 9 “N”)について，そのノードを終点とする合成のエッジ(図 9 “E”)が存在するならば，そのエッジを削除する．この処理は，削除したエッジの始点ノードに移動して再帰的に繰り返す”

本拡張を加えた合成手順グラフを図 9”Path B”に示す．問題の原因であった合成は行われなくなっており，このグラフに基づく合成処理を行うと，問題を回避したフローであるフローJ が得られる．

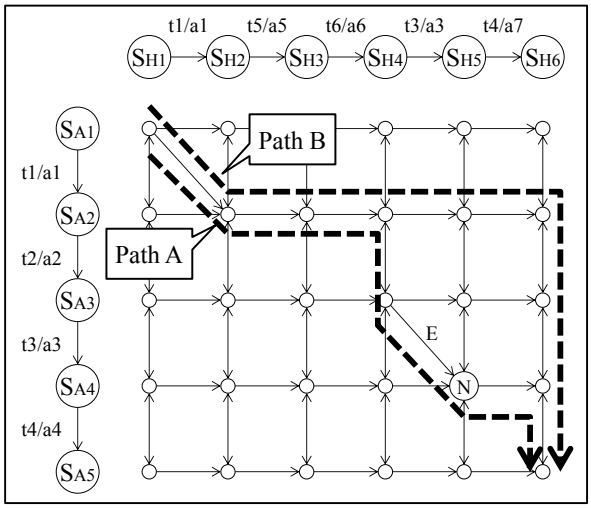


図 9 フローA と H の合成手順グラフ

## 7. 適用結果および評価

本稿で提案したアルゴリズムに関する，適用および評価について示す．

### 7.1 実験環境

実験環境には一般的な PC を用いた．スペックの概要を以下に示す．

| | |
|---|---|
| OS: | Windows7 Professinal Service Pack1 |
| CPU: | Intel® Core™ i7-4790 3.60GHz |
| RAM: | 16GB |

アルゴリズムは Java 言語を用いて，43 クラス，約 2300 ステップで実装した．なお入力となるフロー情報を作成するための，パケット情報解析機能は除いている．

### 7.2 フロー合成結果

実験では，実際の情報制御システムを構成する計算機から取得したパケット情報を用いる．パケット情報の解析結果であるフロー情報を入力として合成処理を適用する．

フロー情報におけるトリガは，メッセージ送信処理および閾値異常の待機処理については時間経過を，メッセージの更新については累積の経過時間を適用した．アクションはあらかじめプロトコル解析により得られたメッセージの同一性に関する情報を採用する．

メッセージ数を 10 メッセージから 690 メッセージまで増加させた場合について，図 10 に合成後のノード数およびエッジ数の推移を，図 11 に処理時間の推移を示す．合成ノードの数の増減が少ない一方で，エッジの数は入力メッセージ数に比例して増加している．また処理時間はメッセージ数に対して指数的な増加を確認できる．

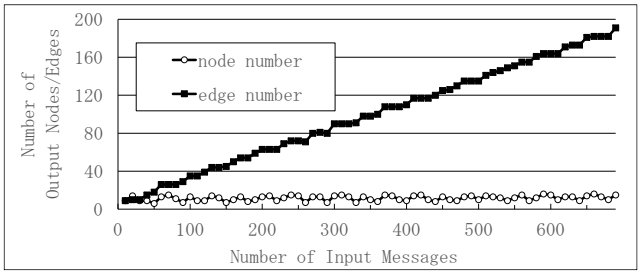


図 10　ノード数およびエッジ数の推移

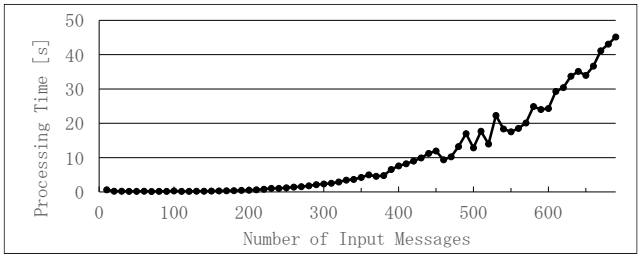


図 11　処理時間の推移

### 7.3 評価

本研究で提案したアルゴリズム拡張による分岐およびループの検出について，サンプルの処理結果を抽出して評価する．図 12 に示す例は，75 メッセージから抽出した 76 ノード，75 エッジのフロー情報を，13 ノード，26 メッセージに合成した結果である．

フロー合成の結果ループが検出されており，本稿で提案および拡張したアルゴリズムが有効に機能していることが確認できる．合成対象の増加に伴う結果の変動について，図 10 では，合成エッジの数がメッセージ数に比例して増加する一方で，合成ノードの数が上下しており，これらについて詳細を調査した．

合成エッジについて詳細を調査したところ，一定周期ごとにデータの更新に要因があることが分かった．メッセージの更新処理は，プロトコル解析機能によってその更新内容を抽出しているが，メッセージの内容は複雑に変化するため，大規模なデータの更新の同一性の解析は困難であり，更新ごとに異なる更新処理がなされたと判断している．この結果として一定時間ごとに遷移が増加した．

合成ノードについて詳細を調査したところ，この理由は，入力したフロー情報の末尾に原因が存在することが分かった．本実験において，末尾のフローの数は入力情報の増加に伴い変化するため末尾部分は周期処理一回分と一致しない場合がほとんどである．そのため，解析アルゴリズムは合成手順グラフ作成時に，末尾部分をループ処理ではなく，異なる処理へ分岐であると解析した．ここで発生した分岐は，実際にシステムの内部状態が分岐したわけではないため，システムの挙動を正確に抽出しているとはいえない．リプレイでの利用に問題は無いが解析や見える化の点では課題である．

図 11 で示したように，処理時間が指数的に増加する点は本アルゴリズムの元となったレーベンシュタイン距離のアルゴリズムの計算量が二つの文字列の長さの積となる点からも当然の結果である．ここで課題となるのは，690 メッセージの合成に一般的な PC で 45 秒を費やす点である．今後大規模な適用を実施した場合には現実的な時間で処理が行えない懸念があるため，この削減手法については検討が必要である．

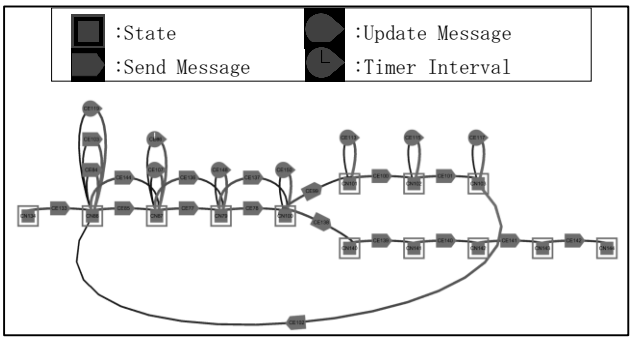


図 12 合成結果から得られる状態遷移図の例

### 7.4 効果

本研究で提案した手法により，メッセージ情報を入力として，長いループを優先したループ抽出ができた．また本研究で抽出したフローを PARROT への入力としてリプレイ処理を行ったところ，キャプチャ内容と同様の通信処理が再現され，リプレイのための入力情報として利用可能であることを確認した．これにより，本稿で提案したアルゴリズムは，PARROT のための入力情報作成として有効である．

### 7.5 メッセージリプレイの応用可能性

メッセージリプレイは計算機ノードの挙動再現やバリエーションテストの容易化に有効である．一方で，他の応用として Metamorphic Testing[16]を参考とした利用が考えられる．Metamorphic Testing は入力データを変化させてテストを実行しその結果を比較するテスト手法で，テストオラクルが無いシステムへのテスト手法として提案されている．本稿で実現したフロー情報合成により，合成フロー情報から，入力したフロー情報には存在しなかった語 $W=Lang(Flow_{cmp})-Lang(Flow_{dst})-Lang(Flow_{src})$を得られる．

W を擬似的なテストオラクルとして，メッセージリプレイによりテストする．もしも，テストの結果として合成フロー情報には存在しないフローが発生した場合には，合成フロー情報がシステムの挙動を十分に解析できなかったことを意味する．そこで，新たに得られたフロー情報を用いてフロー情報合成を行えば，より精度を向上した合成フロー情報の構築が期待できる．

### 7.6 フロー情報合成の応用可能性

本稿で示したフロー情報合成アルゴリズムによって，メッセージ情報の解析結果から，フロー情報として合成処理を行い，計算機ノードの状態遷移を推測できた．この合成結果は必ずしも計算機が内部に持つ状態遷移と一致するとは限らないが，実際に発生した自称に対して矛盾の無い合成結果であるため，リプレイのみならず，その挙動解析において他の情報についても有効であると考えられる．

応用例としては，差異抽出によるシステムの問題解析が挙げられる．例として，ある計算機について定常的にメッセージ情報を収集し，問題の発生していないフロー情報について合成フロー情報を算出しておく．計算機に何らかの問題が発生した際には作成しておいた合成フロー情報と，問題が発生する前後のフロー情報を合成する．合成フローに新たに状態遷移 $\delta_{err}$( $\delta_{err}=\delta_{cmp}\cap\neg\delta_{dst}$)が追加されるならば，$\delta_{err}$ は問題発生時特有の計算機の挙動であるため，原因究明の参考情報として有用である．

用いる情報は，今回用いたメッセージ情報のみならず計算機システムに関連するその他の情報が利用可能である．例としては，アプリケーションの関数実行履歴が挙げられる．関数の実行履歴そのものはログの出力やアスペクトのウィーブによって容易に得られる一方で，動的な解析ではその情報の膨大さから挙動の把握や可視化に課題があった．そこで本稿のアルゴリズムを適用することで状態を合成し，処理の分岐条件など必要な情報にフォーカスしやすい情報を提供できる．また，システムの仕様に関する情報が別途得られている場合には，合成フロー情報との突合せによる整合性の確認が可能である．

これら応用は，本稿のアルゴリズムのみならず，正規文法の抽出や状態遷移機械に関する他の技術においても応用可能であるが，本稿のアルゴリズムは不正なく分岐やループを抽出できるため，精度の点で有利と考える．

### 7.7 今後の課題

本稿で示した合成手順グラフに関するアルゴリズムにおける最大の課題はその一般性の定式化である．合成手順グラフの提案およびその適用についてはある程度示したものの，不正な合成の回避やループの抽出については，結果的に望ましい出力が得られたことを示したに過ぎない．本アルゴリズムの有効性を定式的に検討できているとは言えず，今後の課題である．

アルゴリズムの拡張に関する課題は二点あり，利用する情報の追加と，計算量増大への対応である．

メッセージ情報に含まれる情報には，プロトコルによってはリクエスト・レスポンスに関する情報や，CRUD(Create, Read, Update, Delete)のようにデータに対する機能に関する情報が含まれる．本稿ではこれらの情報の活用については検討していないが，活用により合成の精度や効率をさらに改善できる可能性があり，今後の検討課題である．

計算量の増大は，今後の大規模な適用を検討するにあたり顕現する可能性が高い課題である．この解決のためには，本稿のアルゴリズムの基礎である状態遷移モデルのさらなるブラッシュアップも有効であると考えられえる一方で，今回検討していないデータ形式についてもブレイクスルーが存在する可能性があり，検討すべきである．

## 8. おわりに

本稿で示したフロー情報合成アルゴリズムは，メッセージリプレイを行うテストにおける有効性のみならず，システムの挙動解析による効率化が望まれる様々な情報の解析において有意義な可能性があることを確認した．今後は本アルゴリズムの一般化と，適用結果を参考とした改良や適用範囲の拡大を実施し検証作業の効率化に取り組む．

## 参考文献

[1] Cem Kaner, Jack Falk, Hung Quoc Nguyen. 基本から学ぶソフトウェアテスト, テスト技術者交流会 訳.
[2] CA LISA. Service Virtualization. Online at http://www.ca.com/jp/devcenter/ca-lisa.aspx
[3] Oracle. Database Replay. Online at http://docs.oracle.com/cd/B28359_01/server.111/e12253/dbr_intro.htm
[4] TestPlant. eggPlant Tools. Online at http://www.testplant.com/eggplant/testing-tools/
[5] SeleniumHQ Browser Automation. Online at http://www.seleniumhq.org/
[6] Robot Framework. Online at http://robotframework.org/
[7] Sikuli Script. Online at http://www.sikuli.org/
[8] 森拓郎, 秦野康生, 臼井崇文, 田中修一. ネットワークリプレイツール PARROT によるテスト自動化のケーススタディ. 研究報告ソフトウェア工学（SE）. 2015-SE-187(39), 2015.
[9] 磯部祥尚, 粂野文洋, 櫻庭健年, 田口研治, 田原康之. ソフトウェア科学基礎. 田中譲 監修. 近代科学社.
[10] Davide Lorenzoli, Leonardo Mariani and Mauro Pezzè. Inferring Statebased Behavior Models. Proceedings of the 2006 International Workshop on Dynamic Systems Analysis(WODA '06). 2006.
[11] W.M.P. van der Aalst, B.F. van Dongen, J. Herbst, L. Maruster, G. Schimm: Workflow mining: A survey of issues and approaches. Data & Knowledge Engineering 47 (2003) 237-267.
[12] Richard M. Karp, Raymond E. Miller, and Arnold L. Rosenberg. Rapid identification of repeated patterns in strings, trees and arrays. In Proceedings of the fourth annual ACM symposium on Theory of computing (STOC '72), 1972.
[13] Masatomo Hashimoto and Akira Mori. A Method for Analyzing Code Homology in Genealogy of Evolving Software. Fundamental Approaches to Software Engineering(FASE 2010). 2010.
[14] Robert A. Wagner, Michael J. Fischer. The String-to-String Correction Problem, Journal of the ACM (JACM) Volume 21 Issue 1. 168-173. 1974.
[15] Juha Kärakkäinen, Peter Sanders and Stefan Burkhardt. Linear Work Suffix Array Construction. Journal of the ACM, Vol. 53, No. 6. November 2006.
[16] T. Y. Chen, T. H. Tse and Zhiquan Zhou. Fault-based testing in the absence of an oracle. Computer Software and Applications Conference. 2001.