ネット理論の応用 —ワークフローネットとプログラムネット

Application of Net Theory
—Workflow Nets and Program Nets

山口真悟 Shingo YAMAGUCHI
葛 崎偉 Qi-Wei GE

アブストラクト ネット理論の応用として、ワークフローをモデル化するためのワークフローネットと、データフロープログラムをモデル化するための(データフロー)プログラムネットを紹介する。ワークフローネットの主要な解析問題として、健全性、振舞い継承、動的変更時間を取り上げ、その定義と解析方法を概説する。またプログラムネットの主要な解析問題として、停止性、不活性、トークン Self-clean 性、並列度を取り上げ、その定義と解析方法を概説する。
キーワード ネット理論、ペトリネット、ワークフローネット(データフロー)プログラムネット

1. はじめに

ネット理論はペトリネット⁽¹⁾⁽²⁾に代表されるネット表現モデルに基づいてシステムを探索する理論である。ネット理論に関する主要な国際会議は International Conference on Application and Theory of Petri Nets and Other Models of Concurrency であり、1980 年から毎年 1 回開催されている。国内では 1986 年に電子通信学会(現電子情報通信学会)ネット理論時限研究会が組織され、現在、その流れをくむ電子情報通信学会コンカレント工学研究会、計測自動制御学会離散事象システム研究会が中心となって研究を進めている。

グラフィカルな表現力と数学的な解析力を併せ持つネット表現モデルは、多くの事象が同時進行するコンカレントシステムのモデリングと解析に適している。これまでに通信システムの設計や生産システムのスケジューリング、計算機アーキテクチャの性能評価など、幅広い多様な応用分野で目覚ましい成果⁽³⁾を上げている。本解説ではネット理論の応用として、ワークフローネット⁽⁴⁾と(データフロー)プログラムネット⁽⁵⁾を紹介し、それらの解析問題と解析方法について、筆者らの近年の研究成果と適用例を交えながら概説する。

ワークフローネットはワークフローをモデル化するためのペトリネットである。ペトリネットの観点からいえば、ワークフローネットはペトリネットの一つのサブクラスに過ぎない。しかし、その単純明解な構造的制約は、ワークフローのモデル化に十分な能力を保ちつつ、その動的性質を構造的に検証可能に

するというバランスのとれた合理的なものとなっている。ワークフローネットはワークフローの諸問題に適用され、成果を上げている。具体例として、ワークフローの動的変更の解析⁽⁶⁾や性能評価⁽⁷⁾などが挙げられる。

プログラムネットはデータフロープログラム⁽⁸⁾⁽⁹⁾をモデル化するための有向グラフであり、一種の拡張ペトリネットとみなすことができる。データフロープログラムはデータ駆動方式の動作原理に従って実行され、命令水準の並列性を引き出すことができる。プログラムネットは、データフロープログラムの記述だけでなく、データフロープログラムの諸問題に適用され、成果を上げている。具体例として、並列性の解析⁽¹⁰⁾やマルチプロセッサスケジューリング⁽¹¹⁾などが挙げられる。

まず導入として、2. でワークフローネットとプログラムネットの概念と定義を紹介する。3. ではワークフローネットの解析問題と解析方法を適用例を交えながら解説する。4. ではプログラムネットの解析問題と解析方法を適用例を交えながら解説する。5. では本解説をまとめ、未解決の問題を整理する。

2. ワークフローネットとプログラムネット

2.1 ワークフローネット

(a) 定 義

ワークフローネットはワークフロー(厳密にいえば、その制御フロー)をモデル化するためのラベル付きペトリネットである。ワークフローネットは van der Aalst⁽¹²⁾によって提案された。

ワークフローネットの各トランジションはワークフローにおける一つのアクションを表す。アクションにはその実行が観測できるものと、できないものがある。前者を外部アクション、後者を内部アクションという。外部アクションを表すトランジションは、そのアクションの名前でラベル付けされ、内部アクションを表すトランジションは特別な記号 τ でラベル付けされる。

[定義 1](ラベル付き)ワークフローネット⁽¹³⁾ 以下の二

山口真悟 正員 山口大学大学院理工学研究科
E-mail shingo@yamaguchi-u.ac.jp
葛 崎偉 正員 山口大学教育学部
E-mail gqw@yamaguchi-u.ac.jp
Shingo YAMAGUCHI, Member (Graduate School of Science and Engineering, Yamaguchi University, Ube-shi, 755-8611 Japan), Qi-Wei GE, Member (Faculty of Education, Yamaguchi University, Yamaguchi-shi, 753-8513 Japan).
Fundamentals Review Vol.3 No.3 pp.52-63 2010 年 1 月

図1 ワークフローネット N_1 . このワークフローネットは、ある会社における注文処理のワークフローを表している。

つの条件がすべて成り立つラベル付き正規ペトリネット $N=(P, T, A, \ell)$ を (ラベル付き) ワークフローネットという。

- (1) N には唯一のソースプレース p_I (すなわち $\bullet p_I = \emptyset$ かつ $\forall p \in P - \{p_I\} : \bullet p \neq \emptyset$) と唯一のシンクプレース p_O (すなわち $p_O \bullet = \emptyset$ かつ $\forall p \in P - \{p_O\} : p \bullet \neq \emptyset$) がある。
- (2) すべてのノードは p_I から p_O への有向道上に存在する。

ここで P と T ($\cap P = \emptyset$), A ($\subseteq (P \times T) \cup (T \times P)$) は、それぞれプレースとトランジション、アークの有限集合である。 $\ell: T \rightarrow \mathcal{A} \cup \{\tau\}$ はトランジションのラベル付け関数であり、 $\mathcal{A}$ は外部アクションのラベルの全体集合である。またプレースあるいはトランジション x ($x \in P \cup T$) に対して、 $\bullet x = \{y | (y, x) \in A\}$, $x \bullet = \{y | (x, y) \in A\}$ である。

ワークフロー管理連合 (Workflow Management Coalition; WfMC) の定義⁽¹⁴⁾によれば、ワークフローは直列 (sequential), 並列 (parallel), 条件付き (conditional), 繰り返し (iteration) という4種類の経路からなる。それらはワークフローネットで次のようにモデル化できる⁽¹⁵⁾。直列経路は複数のアクションが決まった順に実行される部分である。直列経路はプレースとトランジションを交互に接続することによって表現される。並列経路は複数のアクションが同時にあるいは任意の順に実行される部分である。その開始点を AND 分岐, 終了点を AND 合流と呼ぶ。AND 分岐, AND 合流はそれぞれ、複数の出力プレースを持つトランジション, 複数の入力プレースを持つトランジションによって表現される。条件付き経路はアクションが選択的に実行される部分である。その開始点を OR 分岐, 終了点を OR 合流と呼ぶ。OR 分岐, OR 合流はそれぞれ、複数の出力トランジションを持つプレース, 複数の入力トランジションを持つプレースによって表現される。繰り返し経路はある条件が満たされるまで、複数のアクションが繰り返し実行される部分であ

る。それは有向閉路によって表現される。

ワークフローネット $N=(P, T, A, \ell)$ のマーキングは、しばしば P 上の多重集合 (bag) として表記される。多重集合は同じ要素が重複して現れることを許すように集合を一般化した概念である。 N のマーキング M とプレース p ($p \in P$) に対して、プレース p のマーキング (トークン数) を $M(p)$ と表記し、ネット全体のマーキングを $[p^{M(p)} | p \in P, M(p) > 0]$ と表記する。例えば $[p_1, p_3^2]$ は p_1 に一つのトークン, p_3 に二つのトークン, それ以外のプレースにはトークンがないマーキングを表す。多重集合の演算は、通常の集合の演算と同様に定義される。 $[p_I]$ と $[p_O]$ はそれぞれワークフローの開始状態と終了状態を表すマーキングであり、通常断りなく初期マーキングと最終マーキングとして扱われる。

ワークフローネットのトランジションの発火可能則と発火則はペトリネットと同じである。すなわちマーキング M とプレース t ($t \in T$) に対して、 $M \geq \bullet t$ ならば、 t は M において発火可能であるという。また t が発火すると、 M は $M - \bullet t \cup t \bullet$ に変換される。マーキング M を別のマーキング M' へ変換するトランジションの発火系列 σ が存在するとき、 M' は M から到達 (reachable) であるといい、これを $M[N, \sigma]M'$ と記す。またマーキング M から到達可能なマーキング全体の集合を $R(N, M)$ と記す。ワークフローネット N に対して、 p_O から p_I へ追加のトランジション t^* を通じて接続することによって得られるペトリネットを N の Short-circuited ネットと呼び、 $\bar{N}$ と記す。

(b) 例

ワークフローネットの一例を図1に示す。このワークフローネットは、ある会社における注文処理のワークフローを表している。顧客から商品の注文書が届くと、このワークフローは開始する。まず注文書の登録が行われ、次に在庫確認とクレジットチェックが並行に実行される。評価の結果、注文を承諾した場合、顧客へ商品の発送と請求書の発送が並行に実行される。却下した場合は、そのことが顧客へ通知される。最後に、処理結果の記録が行われると、このワークフローは終了する。

(c) サブクラス

ワークフローネットのサブクラスとは、ワークフローネットのグラフ的な構造に制約を加えたものである。ワークフローネットの主要なサブクラスには以下のものがある。

- (1) 状態機械ワークフローネット: すべてのトランジション t に対して、 $|\bullet t| = |t \bullet| = 1$ が成り立つワークフローネット。
- (2) マークグラフワークフローネット: $|p_I \bullet| = |p_O \bullet| = 1$, p_I と p_O 以外のすべてのプレース p に対して、 $|\bullet p| = |p \bullet| = 1$ が成り立つワークフローネット。
- (3) 自由選択ワークフローネット: すべてのプレース p_1, p_2 に対して、 $p_1 \bullet \cap p_2 \bullet \neq \emptyset \Rightarrow |p_1 \bullet| = |p_2 \bullet| = 1$ が成り立つ

ワークフローネット。

- (4) 拡張自由選択ワークフローネット^(注1): すべてのプレース p_1, p_2 に対して, $p_1 \bullet \cap p_2 \bullet \neq \emptyset \Rightarrow p_1 \bullet = p_2 \bullet$ が成り立つワークフローネット。
- (5) 非対称選択ワークフローネット: すべてのプレース p_1, p_2 に対して, $p_1 \bullet \cap p_2 \bullet \neq \emptyset \Rightarrow p_1 \bullet \subseteq p_2 \bullet$ または $p_1 \bullet \supseteq p_2 \bullet$ が成り立つワークフローネット。
- (6) Well-structured ワークフローネット: Short-circuited ネットに TP-ハンドルや PT-ハンドル^(注2)がないワークフローネット。
- (7) アサイクリック (acyclic) ワークフローネット: 有向閉路を持たないワークフローネット。

Well-structured ワークフローネットと自由選択ワークフローネットは比較不能であるが, アサイクリック Well-structured ワークフローネットはアサイクリック自由選択ワークフローネットのサブクラスである。

実用的なワークフローの多くは拡張自由選択ワークフローネットでモデル化できる⁽¹²⁾ことが知られている。

2.2 プログラムネット

(a) 定義

プログラムネットはデータフロープログラムをモデル化するための有向グラフである。プログラムネットは, Dennis ら⁽¹⁶⁾のデータフロープログラムグラフを元にして, 翁長ら⁽⁵⁾によって提案された。ノードはデータフロープログラムの命令を表し, 有向枝はデータの依存関係若しくは制御の依存関係を表す。ノードは発火則の違いによって AND-node, OR-node, SWITCH-node の3種類に分けられる。葛ら⁽¹⁷⁾は各有向枝に重み α, β を導入している。これらの値はそれぞれ, その有向枝の終点に対応した命令へ入力されるデータの数, 同じく始点の命令から出力されるデータの数を表す。 α, β はデータフロープログラムにおける, 例えばベクトルのような, 多次元のデータを扱うことを可能にする。

[定義2](プログラムネット⁽¹⁷⁾) プログラムネットは $PN = (N, E, \alpha, \beta)$ で表される4項組である。その要素は以下のとおりである。

- (1) N はノードの有限集合^(注3)である。各ノード $z \in N$ は

(注1): 自由選択ワークフローネットと総称されることも多いが, 本解説では区別する。

(注2): ベトリネットの有向閉路 c に関して, c 上のトランジションからプレースへの有向道を TP-ハンドルといい, c 上のプレースからトランジションへの有向道を PT-ハンドルという。

(注3): ノード z に対して, その入力, 出力の集合をそれぞれ $\bullet z = \{x | (x, z) \in E\}$, $z \bullet = \{x | (z, x) \in E\}$ と定義する。

AND-node, OR-node, SWITCH-node のいずれかである。開始ノード (start node) s という唯一のソースノード (すなわち $\bullet s = \emptyset$ かつ $\forall z \in N - \{s\} : \bullet z \neq \emptyset$) と終了ノード (termination node) という唯一のシンクノード t (すなわち $t \bullet = \emptyset$ かつ $\forall z \in N - \{t\} : z \bullet \neq \emptyset$) があり, すべてのノードは s から t への有向道上に存在する。

- (2) E は先入れ先出し機能を有する有向枝の有限集合である。各有向枝 $e \in E$ はデータフロー枝か制御フロー枝のいずれかである。
- (3) $\alpha : E \rightarrow \{1, 2, 3, \dots\}$ と $\beta : E \rightarrow \{1, 2, 3, \dots\}$ は枝の重みである。枝 e に対して, α_e はその終点に対する重み, β_e はその始点に対する重みである。

□

開始ノード s と終了ノード t は AND-node である。OR-node には2本の入力データフロー枝と1本の出力データフロー枝がある。SWITCH-node には1本の入力データフロー枝と1本の入力制御フロー枝, 2本の出力データフロー枝がある。SWITCH-node の出力枝の一方は True とラベル付けされ, 他方は False とラベル付けされる。制御フロー枝 e の重み α_e と β_e は常に $\alpha_e = \beta_e = 1$ である。

プログラムネットのトークンは一つのデータを表し, 有向枝に置かれる。プログラムネットのトークン分布は $|E|$ 次元ベクトル d で表される。各要素 d_e は e に置かれたトークンの数を表す。初期トークン分布を d_0 と記す。開始ノード s は d_0 において1回だけ発火する。トークン分布は以下の発火則に従って変化する。

- AND-node は各入力枝 e_{in} に少なくとも $\alpha_{e_{in}}$ 個のトークンがあるとき, 発火可能になる。発火可能な AND-node が発火すると, 各入力枝 e_{in} から $\alpha_{e_{in}}$ 個のトークンが取り去られ, 各出力枝 e_{out} に $\beta_{e_{out}}$ 個のトークンが加えられる。
- OR-node は入力枝のどれか一つ (e_{in} とする) に少なくとも $\alpha_{e_{in}}$ 個のトークンがあるとき, 発火可能になる。発火可能な OR-node が発火すると, 入力枝 e_{in} から $\alpha_{e_{in}}$ 個のトークンが取り去られ, 出力枝 e_{out} に $\beta_{e_{out}}$ 個のトークンが加えられる。
- SWITCH-node はデータフロー枝 e_{in} に少なくとも $\alpha_{e_{in}}$ 個のトークンがあり, 制御フロー枝に少なくとも一個のトークンがあるときに発火可能になる。発火可能な SWITCH-node が発火すると, データフロー枝 e_{in} から $\alpha_{e_{in}}$ 個のトークンが取り去られ, 制御フロー枝から1個のトークンが取り去られる。制御フロー枝から取り去ったトークンの値が True だった場合, True とラベル付けされた出力枝 e_{True} に $\beta_{e_{True}}$ 個のトークンが加えられる。制御フロー枝から取り去ったトークンの値が False だった場合, False とラベル付けされた出力枝 e_{False} に $\beta_{e_{False}}$ 個のトークンが加えられる。

グラフとして表現する場合, AND-node は $\bigcirc$, OR-node は Δ , SWITCH-node は ∇° として描かれ, データフロー枝は $\rightarrow$, 制御フロー枝は $\dashrightarrow$ として描かれる。枝の重みはその枝に付されるが, 重みが1である場合は省略されることが多い。SWITCH-node の出力枝に付された True と False のラベルはそれぞれ T と F に略記される。トークンは有向枝の上に点とし

図2 プログラムネット PN_1 . このプログラムネットは与えられた西暦 y 年 m 月 d 日の曜日を求めるデータフロープログラムを表している .

て描かれる .

(b) 例

プログラムネットの一例を図2に示す . このプログラムネットは与えられた西暦 y 年 m 月 d 日の曜日を求める . ただし1月と2月はそれぞれ前年13月と14月とみなす . 終了ノードに入力されるトークンの値が0なら日曜日 , 1なら月曜日 , $\dots$, 6なら土曜日である .

(c) サブクラス

プログラムネットのサブクラスとは , プログラムネットのグラフ的な構造に制約を加えたものである . プログラムネットの主要なサブクラスには以下のものがある .

- (1) AND プログラムネット : AND-node だけから成るプログラムネット .
- (2) SWITCH-less プログラムネット : AND-node と OR-node だけから成るプログラムネット .
- (3) アサイクリックプログラムネット : 有向閉路を持たないプログラムネット .

この章ではワークフローネットの主要な解析問題として , 健全性 (soundness) , 振舞い継承 (behavioral inheritance) , 動的変更時間 (dynamic change time) を取り上げ , その定義と解析方法を紹介する . ワークフローネットには , それら以外に , 直列化可能性 (serializability)¹⁸⁾ , 最大スループット (maximum throughput)⁷⁾ (動的変更) 過渡時間 (transient time)¹⁹⁾ などの解析問題がある . 詳細はそれぞれの文献を参照されたい .

3.1 健全性

(a) 定義

ワークフローネットはワークフローの制御フローを表し , ワークフローネットの振舞いはワークフローインスタンスのライフサイクルを表す . ワークフローインスタンスは , 制御フローの最初で開始し , 最後で終了しなければならない . 途中で行き詰まったり , 何かを残して終了してはならない . また , すべてのアクションは何らかのワークフローインスタンスの処理に役立つべきである . 換言すれば , 決して実行されないアクションはあってはならない . ワークフローネットの形式で , この性質は , $[p_I]$ から可達な任意のマーキングは p_O にトークンがあるマーキングへ可達であり , p_O にトークンがあるマーキングは $[p_O]$ であり , そして不活性なトランジションが存在しないこと意味する . この性質を健全性という . 健全性はワークフローネットの論理的な正しさの一つの基準である . 健全性はそれ自身重要な性質であるが , ワークフローネットのあらゆる動的性質を研究する上で , その基礎になっているという意味でも重要である .

健全性は以下のように定義される .

[定義3] (健全性²⁰⁾) ワークフローネット $N = (P, T, A, \ell)$ は以下をすべて満たすならば , 健全であるという .

- (1) $\forall M \in R(N, [p_I]) : \exists M' \in R(N, M) : M' \geq [p_O]$
- (2) $\forall M \in R(N, [p_I]) : M \geq [p_O] \Rightarrow M = [p_O]$
- (3) $(N, [p_I])$ に不活性なトランジションは存在しない

□

健全性には別の等価な定義¹²⁾がある . また , ワークフローネット N が健全であるとき , かつそのときに限り , $(\bar{N}, [p_I])$ が有界かつ活性である¹²⁾という重要な性質がある .

(b) 判定法

状態機械ワークフローネットは健全である . 有向閉路のあるマークグラフワークフローネットは不健全である²¹⁾ .

拡張自由選択ワークフローネット $N = (P, T, A, \ell)$ の健全性の必要十分条件は , 次の条件がすべて成り立つこと²²⁾である . $\bar{N}$ のプレース-トランジション接続行列²⁾を $A_{\bar{N}}$ とすると ,

- (1) $A_{\bar{N}}x = 0, x > 0$ なる有理数ベクトル x が存在する .
- (2) $A_{\bar{N}}^T y = 0, y > 0$ なる有理数ベクトル y が存在する .
- (3) $\text{rank } A_{\bar{N}} = |C_{\bar{N}}| - 1$. ここで $\text{rank } A_{\bar{N}}$ は $A_{\bar{N}}$ の階

数, $C_{\overline{N}}$ は $\overline{N}$ のクラスタ^(注4)の集合である.

(4) $[p_I]$ においてトークンなしのサイフォン⁽²⁾が存在しない.

この必要十分条件は多項式時間で検証可能である⁽²³⁾. ゆえに拡張自由選択ワークフローネットの健全性は多項式時間で判定可能である.

アサイクリック Well-structured ワークフローネットは健全である⁽²¹⁾. これはアサイクリック Well-structured ワークフローネットがアサイクリック自由選択ワークフローネットのサブクラスであり, 更に TP-ハンドルと PT-ハンドルを持たないからである. 一方, 有向閉路のある Well-structured ワークフローネットの健全性も多項式時間で判定可能である. 詳細は文献(22)を参照されたい. なお健全な拡張自由選択ワークフローネットと健全な Well-structured ワークフローネットは安全 ($\forall M \in R(N, [p_I]) : M \leq P$) である⁽⁴⁾.

複数のワークフローを結合したインタワークフロー⁽²⁴⁾を表すワークフローネットは, 非対称選択になる場合がある⁽²⁵⁾. 非対称選択ワークフローネットの健全性問題は NP 困難である⁽²⁶⁾. そのようなネットの健全性を効率良く判定するために, 筆者ら⁽²⁷⁾はモデル検査による判定方法を明らかにしている. アサイクリックワークフローネット $N = (P, T, A, \ell)$ の健全性の線形時相論理 (linear temporal logic) による必要十分条件は, 次の条件がすべて成り立つことである.

- (1) $\Diamond(M \geq [p_O])$ in $(N, [p_I])$
- (2) $\Box((M \geq [p_O]) \Rightarrow (M = [p_O]))$ in $(N, [p_I])$
- (3) $\neg(\Box(\exists t \in T : X(t) = 0))$ in $(\overline{N}, [p_I])$

ここで $\Diamond\phi$ は ϕ が将来いつか必ず成り立つこと, $\Box\phi$ は ϕ が今後常に成り立つこと, $X(t)$ は t の発火回数を意味する.

更に筆者らは, 有向閉路のある一般的なワークフローネットの健全性を効率良く判定するために, 線形時相論理の検査と計算木論理 (computational tree logic) の検査を組み合わせた判定法を明らかにしている. 詳細は文献(28)を参照されたい.

3.2 振舞い継承

(a) 定義

ワークフローは, 論理的に正しいだけでなく, 与えられた仕様を満たしていなければならない. 前者のチェックには健全性の解析技法が利用できる. 一方, 後者のチェックはコンテキスト依存であるため, シミュレーションによる検証が一般的である. しかしワークフローの規模が大きくなれば, シミュレーションにかかる時間は膨大になって検証は困難になる. この検証に対する有望な技法が振舞い継承である. あるワークフローが別の

ワークフローの振舞い (ワークフローインスタンスのライフサイクル) を含むならば, 前者 (サブクラスという) は後者 (スーパークラスという) の振舞いを継承するという. スーパークラスは, サブクラスの仕様を指定するために使うことができる.

ワークフローネットの形式で, 振舞い継承は等価性の一つである Branching bisimilarity の緩和として定義される. 直観的に Branching bisimilarity は, $[p_I]$ から実行可能な外部アクションの実行系列全体の集合が同じワークフローネットを等しいものとみなす. サブクラスのワークフローネットは, スーパークラスのワークフローネットにはない外部アクションの実行系列を持つことができる. あるワークフローネット N_X は, それに追加された外部アクションの実行をブロックした場合, 別のワークフローネット N_Y と Branching bisimilarity になるならば, N_Y の振舞いを Protocol 継承するという. また N_X に追加された外部アクションを内部アクションへ変更した場合, N_Y と Branching bisimilarity になるならば, N_Y の振舞いを Projection 継承するという. そして N_X に追加された一部の外部アクションの実行をブロックし, その他の外部アクションを内部アクションへ変更した場合, N_Y と Branching bisimilarity になるならば, N_Y の振舞いを Life-cycle 継承する (あるいは単に, N_Y を振舞い継承する) という.

Branching bisimilarity の形式的な定義を与えるために, まずワークフローネット $N = (P, T, A, \ell)$ のラベル付き可達グラフ $G = (V, E)$ を以下のように定義する: (i) $V = R(N, [p_I])$, (ii) $E = \{(M, \ell(t), M') \mid M, M' \in V, t \in T, M[N, t]M'\}$. ノード v' がノード v から α とラベル付けされた枝を通して到達できる ($(v, \alpha, v') \in E$) ならば $v[\alpha]v'$ と表記し, 任意の本数の τ とラベル付けされた枝を通して到達できるならば $v[\tau^*]v'$ と表記する. また (i) $\alpha = \tau \wedge v = v'$, あるいは (ii) $v[\alpha]v'$ のいずれかであることを $v[(\alpha)]v'$ と略記する.

[定義 4] (Branching bisimilarity⁽²⁹⁾) $G_X = (V_X, E_X)$ をワークフローネット N_X のラベル付き可達グラフ, $G_Y = (V_Y, E_Y)$ をワークフローネット N_Y のラベル付き可達グラフとし, $\mathcal{R} \subseteq V_X \times V_Y$ をグラフ間の 2 項関係とする. 関係 $\mathcal{R}$ は以下を満たすときかつ, そのときに限り, G_X と G_Y に関する Branching bisimulation と呼ばれる.

- (1) $u\mathcal{R}v \wedge u[\alpha]u'$ であるとき, $v[\tau^*]v'' \wedge v''[(\alpha)]v' \wedge u\mathcal{R}v'' \wedge u'\mathcal{R}v'$ となるような $v', v'' \in V_Y$ が存在する.
- (2) $u\mathcal{R}v \wedge v[\alpha]v'$ であるとき, $u[\tau^*]u'' \wedge u''[(\alpha)]u' \wedge u''\mathcal{R}v \wedge u'\mathcal{R}v'$ となるような $u', u'' \in V_X$ が存在する.
- (3) $u\mathcal{R}v$ であるとき, $(u = [p_O^X])$ ならば $v[\tau^*][p_O^Y]) \wedge (v = [p_O^Y])$ ならば $u[\tau^*][p_O^X])$. ここで p_O^X と p_O^Y はそれぞれ N_X と N_Y のシンクブレースである.

$(N_X, [p_I^X])$ と $(N_Y, [p_I^Y])$ は, G_X と G_Y の間に $[p_I^X]\mathcal{R}[p_I^Y]$ となるような Branching bisimulation $\mathcal{R}$ が存在するとき, かつそのときに限り, Branching bisimilar である. ここで p_I^A と p_I^B はそれぞれ N_A と N_B のソースブレースである. $\square$

振舞い継承の形式的な定義を与えるために, ワークフ

(注4): ペトリネットのブレースとトランジションの空でない部分集合の組 (P', T') は次の条件をすべて満たすならば, クラスター (cluster) であるという.
(i) $\forall p \in P' : p^* \subseteq T'$ かつ $\forall t \in T' : t^* \subseteq P'$, (ii) P', T' は条件 (i) を満たすもののうち集合の包含関係の意味で極小である.

ローネット $N=(P, T, A, \ell)$ 中の外部アクションに対して、その実行をブロックする Encapsulation 操作と、内部アクションへ変更する Abstraction 操作を導入する。 N における外部アクションの部分集合 H に対する Encapsulation 操作は $\partial_H: N \mapsto (P, T', A', \ell')$ であり、 $T'=\{t \in T | \ell(t) \notin H\}$ 、 $A'=A \cap ((P \times T') \cup (T' \times P))$ 、 $\ell'=\ell \cap (T' \times \mathcal{A})$ である。また N における外部アクションの部分集合 I に対する Abstraction 操作は $\tau_I: N \mapsto (P, T, A, \ell')$ であり、各トランジション $t \in T$ に対し、 $\ell(t) \in I$ ならば $\ell'(t)=\tau$ 、そうでなければ $\ell'(t)=\ell(t)$ である。

[定義 5] (振舞い継承⁽¹³⁾) N_X と N_Y をワークフローネット、 p_I^X と p_I^Y をそれぞれ N_X と N_Y のソースプレースとする。

Protocol 継承: $(\partial_H(N_X), [p_I^X]) \sim_b (N_Y, [p_I^Y])$ となる $H (\subseteq \mathcal{A})$ が存在するとき、かつそのときに限り、 N_X は N_Y の振舞いを Protocol 継承する。

Projection 継承: $(\tau_I(N_X), [p_I^X]) \sim_b (N_Y, [p_I^Y])$ となる $I (\subseteq \mathcal{A})$ が存在するとき、かつそのときに限り、 N_X は N_Y の振舞いを Projection 継承する。

Life-cycle 継承: $(\tau_I \circ \partial_H(N_X), [p_I^X]) \sim_b (N_Y, [p_I^Y]) \wedge I \cap H = \emptyset$ となる $I (\subseteq \mathcal{A})$ と $H (\subseteq \mathcal{A})$ が存在するとき、かつそのときに限り、 N_X は N_Y の振舞いを Life-cycle 継承する。

□

Protocol 継承ならば Life-cycle 継承、Projection 継承ならば Life-cycle 継承である。また Protocol 継承、Projection 継承及び Life-cycle 継承はいずれも半順序関係である⁽¹³⁾。

(b) 判定法

Protocol 継承、Projection 継承及び Life-cycle 継承を判定する問題は決定可能である⁽³⁰⁾。しかしながら、この問題は状態空間爆発 (state-space explosion) のため、手に負えないと考えられている。Verbeek⁽²⁹⁾らは Life-cycle 継承を判定するためのバックトラック法を用いたアルゴリズムを提案している。

また判定の複雑さを回避するため、振舞い継承を維持したままワークフローネットを合成する規則が提案されている。Van der Aalst ら⁽¹³⁾は振舞い継承を維持したままワークフローネットとペトリネットを合成する四つの規則を提案している。また筆者ら⁽³¹⁾は、WfMC が定めるインターワークフローのプロトコル⁽³²⁾が振舞い継承を維持することを明らかにしている。

3.3 動的変更時間

(a) 定義

ワークフローは法律の改正や市場の変化、ビジネスプロセス・リエンジニアリングなどの結果として実行中 (on the fly) に変更されることがある。これはワークフローの動的変更と呼ばれている。動的変更において、ワークフローインスタンスは変更前のワークフローから変更後のワークフローへ移行されなければならない。いうまでもなく、このワークフローインスタンスが行き詰まることや、いずれのワークフローにも従わないこと (それらを動的変更バグという) があってはならない。動的変更には以下の方式が提案されている。

- (1) **Flush 方式**⁽³³⁾: 変更前のワークフローのインスタンスはそのまま変更前のワークフローで処理され、それらがすべて処理された後、ワークフローが変更される。変更開始後に発生したワークフローインスタンスはワークフローが変更されるまで待たされる。
- (2) **Abort 方式**⁽³³⁾: ワークフローは直ちに変更される。変更前のワークフローのインスタンスは変更後のワークフローですべてやり直される。変更開始後に発生したワークフローインスタンスは、変更前のワークフローのインスタンスに続いて処理される。
- (3) **SCO (Synthetic Cut-Over) 方式**⁽³³⁾: 変更前のワークフローと変更後のワークフローの両方が同時に使われる。ただしリソース (ワークフローの作業員) は増えない。変更前のワークフローのインスタンスはそのまま変更前のワークフローで処理され、変更開始後に発生したワークフローインスタンスは変更後のワークフローで処理される。
- (4) **Migrate 方式**⁽³⁴⁾: ワークフローは直ちに変更される。変更前のワークフローのインスタンスは変更後のワークフローでやり直されるが、変更前のワークフローで実行済みのアクションは変更後でも実行済みとみなされ、やり直しはまだ実行されていない最初のアクションから行われる。変更開始後に発生したワークフローインスタンスは、変更前のワークフローのインスタンスに続いて処理される。
- (5) **Selective shift 方式**⁽³⁵⁾: 変更前のワークフローと変更後のワークフローの両方が同時に使われる。ただしリソースは増えない。変更前のワークフローのインスタンスは一部をそのまま変更前のワークフローで処理され、残りは変更後のワークフローでやり直される。変更前のワークフローで実行済みのアクションは変更後でも実行済みとみなされ、やり直しはまだ実行されていない最初のアクションから行われる。

これらの方式はいずれも動的変更バグを生じないが、動的変更の開始時刻より後に発生したワークフローインスタンスに幾らかの待ち時間を生じる場合がある。この待ち時間は動的変更時間⁽⁶⁾と呼ばれ、短い方がよい。動的変更時間は変更方式によって異なり、その時間を知ること、効率の良い変更方式を選ぶことが可能となる。

動的変更のモデル化にはトランジションの発火に遅延時間を導入したワークフローネット (時間ワークフローネットという) を用いる。ワークフローネットの形式で、動的変更とはワークフローネット内の部分ワークフローネット N_{old} (古い変更領域と呼ぶ) を別のワークフローネット N_{new} (新しい変更領域と呼ぶ) によって置き換えて、新しいワークフローネットを得ることである。変更領域 N_{old} と N_{new} は p_I と p_O をそれぞれ共有するものとし、以下では変更領域のみを考察する。またワークフローインスタンスは繰り返し発生するものとし、その間隔

は一定かつ、どのトランジションの発火時間よりも長いとする。
動的変更時間は以下のように定義される。

[定義 6] (動的変更時間⁽³⁶⁾) 動的変更の開始時刻 τ_{cs} に、古い変更領域 N_{old} 中に存在するワークフローインスタンスの数を μ 、 μ 個のワークフローインスタンスを $c_1, c_2, \dots, c_\mu, \tau_{cs}$ 以降に最初に発生するワークフローインスタンスを $c_{\mu+1}$ とする。
 $c_1, c_2, \dots, c_{\mu+1}$ を表すマーキングをそれぞれ $[p_O]$ へ変換する発火系列を $\sigma^{c_1}, \sigma^{c_2}, \dots, \sigma^{c_{\mu+1}}$ に限定した場合、 N_{old} を新しい変更領域 N_{new} へ変更するための動的変更時間は、以下の式で与えられる。

$$\gamma(\delta, \tau_{cs}, \sigma^{c_1}, \dots, \sigma^{c_{\mu+1}}) \stackrel{\text{def}}{=} \max \{ (\tau_{p_O}^{c_{\mu+1}} - \tau_{p_I}^{c_{\mu+1}}) - \Delta \tau^{c_{\mu+1}}, 0 \}$$

ここで $\tau_{p_O}^{c_{\mu+1}}$ は $c_{\mu+1}$ が p_O へ到着する時刻、 $\tau_{p_I}^{c_{\mu+1}}$ は $c_{\mu+1}$ が p_I へ到着する時刻、 $\Delta \tau^{c_{\mu+1}}$ は $c_{\mu+1}$ が単独で p_I から p_O へ移動する時間である。

N_{old} を N_{new} へ変更するための (最悪) 動的変更時間は、以下の式で与えられる。

$$\gamma(\delta, \tau_{cs}) \stackrel{\text{def}}{=} \max_{(\sigma^{c_1}, \dots, \sigma^{c_{\mu+1}}) \in \Sigma^{c_1} \times \dots \times \Sigma^{c_{\mu+1}}} \{ \gamma(\delta, \tau_{cs}, \sigma^{c_1}, \dots, \sigma^{c_{\mu+1}}) \}$$

ここで Σ^{c_i} は c_i を表すマーキングを $[p_O]$ へ変換する発火系列全体の集合である。

□

(b) 計算法

アサイクリックマークグラフワークフローネットの計算法を示す。

Flush 方式の動的変更時間は、「 N_{old} における最長パスの長さ」-「ワークフローインスタンスの発生間隔」によって計算できる。マークグラフワークフローネットの最長パスの長さは、深さ優先探索に基づいて求められるので、Flush 方式の動的変更時間は多項式時間で計算できる。

Abort 方式の動的変更時間は、「 N_{new} において $[p_I^{\mu+1}]$ を $[p_O^{\mu+1}]$ へ変換するのにかかる最短時間」-「 N_{new} における最長パスの長さ」- $(\tau_{p_I}^{c_{\mu+1}} - \tau_{cs})$ によって計算できる。

SCO, Migrate, Selective shift 方式の動的変更時間はシミュレーションにより求められる。シミュレーションのアルゴリズムは、それぞれ文献 (6) (35) (37) を参照されたい。

SCO 方式は Flush 方式の緩和であり、Migrate 方式は Abort 方式の緩和、Selective shift 方式は SCO 方式と Migrate 方式の緩和なので、動的変更時間の観点からは、Selective shift 方式が五つの方式の中で最も良い方式である⁽³⁵⁾。

アサイクリック状態機械ワークフローネットの計算法は文献⁽³⁶⁾を参照されたい。

3.4 解析例

ワークフローネットの解析例として、図 1 に示すワークフローネット N_1 と、その変形 N_2 の健全性を判定する。 N_2 は N_1 にアーク (p_{10}, t_{11}) を追加することによって得られるワークフロー

ネットである。 N_2 は拡張自由選択ワークフローネットでない ($p_7 \bullet \cap p_{10} \bullet = \{t_{11}\}$ であるが $p_7 \bullet = p_{10} \bullet$ でない) ため、その健全性を判定する問題は手に負えない。そこで、これらの健全性の判定にモデル検査ツール SPIN⁽³⁸⁾ (39) を用いた判定方法⁽²⁷⁾を適用する。

まず $(N_1, [p_I])$ に対する状態遷移を SPIN のモデル記述言語 PROMELA で記述する。その記述を図 3 に示す。PROMELA の言語仕様により添字の番号がトランジションの番号とずれているので、注意されたい。 $(\overline{N_1}, [p_I])$ に対する状態遷移も同様に記述できる。次に N_1 の健全性の判定条件を線形時相論理式として記述する。

$$\langle \rangle (M[13] > 1) \quad (1)$$

$$[] ((M[13] > 1) \rightarrow (M[0] == 0 \ \&\& \ M[1] == 0 \ \&\& \ \dots \ \&\& \ M[12] == 0 \ \&\& \ M[13] == 1)) \quad (2)$$

$$[] (X[0] == 0 \ || \ X[1] == 0 \ || \ \dots \ || \ X[11] == 0) \quad (3)$$

式 (1) と (2) の判定には $(N_1, [p_I])$ の PROMELA 記述を用い、式 (3) の判定には $(\overline{N_1}, [p_I])$ の PROMELA 記述を用いる。SPIN は式 (1) と (2) に対して真、式 (3) に対して偽を出力し、 N_1 は健全と判定される。

N_2 に対する PROMELA 記述も N_1 と同様に記述できる。 N_2 に対する健全性の判定条件は N_1 と同一である。SPIN は式 (1) と (2) に対して真、式 (3) に対して真を出力し、 N_2 は不健全と判定される。更に SPIN は式 (3) の反例として、 t_{11} が不活性であることを示す発火系列 $(t_1 t_2 t_3 t_4 t_5 t_6 t_7 t_8 t_9 t_{10} t_{12} t^*)^\infty$ を出力する。ここで σ^∞ は σ の無限な繰返しを表す。

4. プログラムネットの解析問題と解析方法

4. ではプログラムネットの主要な解析問題として、停止性 (terminacy)、不活性 (dead)、トークン Self-clean 性 (token self-cleanness)、並列度 (parallel degree) を取り上げ、その定義と解析方法を紹介する。プログラムネットには、それら以外に、到達性 (reachability)⁽⁴⁰⁾、計算決定性 (computation determinacy)⁽¹⁷⁾、マルチプロセッサスケジューリング⁽⁴¹⁾などの解析問題がある。詳細はそれぞれの文献を参照されたい。

4.1 停止性

(a) 定義

プログラムネットの各ノードはデータフロープログラムの命令を表しており、ノードの発火系列は命令の実行系列を表す。その長さが無限ならば、実行結果は得られないかもしれない。また、そのような実行系列はシステムの資源を無駄に消費し続けることになるだろう。それゆえ命令の実行系列が有限の長さであることを調べることは極めて重要である。プログラムネットの形式で、この性質は初期トークン分布から実行可能なすべての発火系列が有限の長さであることを意味する。この性質を停


```

#define Place 14
#define Transition 12
#define inp1(x1) (x1>0) -> x1--
#define inp2(x1,x2) (x1>0 && x2>0) -> x1--;x2--
#define out1(x1) x1++
#define out2(x1,x2) x1++; x2++
int M[Place];
int X[Transition];
init
{
  M[0]=1;
  do
  :: atomic{inp1(M[0]) -> X[0]++;out1(M[1])}
  :: atomic{inp1(M[1]) -> X[1]++;out2(M[2],M[4])}
  :: atomic{inp1(M[2]) -> X[2]++;out1(M[3])}
  :: atomic{inp1(M[4]) -> X[3]++;out1(M[5])}
  :: atomic{inp2(M[3],M[5]) -> X[4]++;out1(M[6])}
  :: atomic{inp1(M[6]) -> X[5]++;out1(M[7])}
  :: atomic{inp1(M[7]) -> X[6]++;out2(M[8],M[10])}
  :: atomic{inp1(M[8]) -> X[7]++;out1(M[9])}
  :: atomic{inp1(M[10]) -> X[8]++;out1(M[11])}
  :: atomic{inp2(M[9],M[11]) -> X[9]++;out1(M[12])}
  :: atomic{inp1(M[7]) -> X[10]++;out1(M[12])}
  :: atomic{inp1(M[12]) -> X[11]++;out1(M[13])}
  od
}

```

図 3 図 1 に示すワークフローネット $(N_1, [p_I])$ に対する状態遷移の PROMELA 記述

止性という。停止性はそれ自身重要な性質であるが、プログラムネットのあらゆる動的性質を研究する上で、その基礎になっているという意味でも重要である。

停止性は以下のように定義される。

[定義 7] (停止性¹⁷⁾) 初期トークン分布 d_0 のプログラムネット $PN=(N, E, \alpha, \beta)$ は、 d_0 から実行可能な、すべての発火系列の長さが有限ならば、停止するという。□

初期トークン分布 d_0 のプログラムネット $PN=(N, E, \alpha, \beta)$ において、 d_0 から実行可能な発火系列 σ は、それが発火した後、すべてのノードが発火できないならば、停止するという。

(b) 判定法

アークの重みを $\forall e \in E : \alpha_e = \beta_e = 1$ と仮定する。

初期トークン分布 d_0 のアサイクリック SWITCH-less プログラムネットは停止する。一方、有向閉路がある場合、 d_0 の SWITCH-less プログラムネットの停止性の必要十分条件⁽⁴²⁾ は、各閉路に d_0 から実行可能なすべての発火系列において (i) 1 回も発火できない (すなわち、不活性 (後述) な) ノード、または (ii) 有限回だけ発火するノードから AND-node と OR-node だけからなる有向道に到達可能なノードの少なくとも一方が存在することである。そのようなノードが存在するかどうかは深さ優先

探索に基づいて計算できる。したがって、 d_0 の SWITCH-less プログラムネットの停止性は多項式時間で判定できる。

SWITCH-node がある一般的なプログラムネットの停止性を判定する問題は手に負えないと考えられる。これは一般的なプログラムネットのモデル化能力が、アークの重みを $\forall e \in E : \alpha_e = \beta_e = 1$ と仮定した場合でも、チューリング機械のモデル化能力と同等であり⁽⁴³⁾、チューリング機械の停止性問題が決定不能だからである。

4.2 不活性

(a) 定義

プログラムネットの各ノードはデータフロープログラムの命令を表している。各命令はデータフロープログラムの何らかの実行の中で、少なくとも 1 回は実行されるべきである。なぜなら、全く実行されない命令は、その命令自体が実行されないだけでなく、その出力データを必要とする命令の実行も妨げる恐れがあり、その結果、データフロープログラムの実行が結果を出力せずに停止してしまう恐れもある。更に、そのような命令はメモリを浪費するだけでなく、プログラムのテストにおいて unnecessary コストを要求することになる。

プログラムネットのノードが、初期トークン分布 d_0 から実行可能な、どの発火系列においても発火できなければ、不活性であるという。またプログラムネットは不活性なノードがあれば、部分的不活性 (partially dead) であるという。ノードの不活性とプログラムネットの部分的不活性は以下のように定義される。[定義 8] (不活性⁽⁴⁴⁾) 初期トークン分布 d_0 のプログラムネット $PN=(N, E, \alpha, \beta)$ において、ノード $z (\in N)$ が d_0 から実行可能な、どの発火系列においても発火できないならば、 z は不活性であるという。また PN に不活性なノードがあるならば、 PN は部分的不活性であるという。

□

(b) 判定法

初期トークン分布を $d_0 = 0$ 、アークの重みを $\forall e \in E : \alpha_e = \beta_e = 1$ と仮定する。

アサイクリック SWITCH-less プログラムネットは部分的不活性でない。一方、有向閉路のある SWITCH-less プログラムネットが部分的不活性かどうかは多項式時間で判定可能である⁽⁴⁴⁾。判定には各ノードが少なくとも 1 回発火可能になるかどうかを調べればよい。発火則とアークの重みの仮定から、AND-node が発火可能になるのは、すべての親ノードが発火可能なときであり、OR-node が発火可能になるのは一つ以上の親ノードが発火可能なときである。したがって、開始ノード s から探索を始め、発見されたノードが発火可能ならばその子ノードを探索していくことを繰り返していく。探索できるノードがなくなったとき、すべてのノードが発火可能なら、そのネットは部分的不活性ではないと判定し、そうでなければ、そのネットは部分的不活性であると判定できる。

更に筆者ら⁽⁴⁴⁾は、アサイクリックプログラムネットが部分的不活性かどうか判定する問題が手に負えないことを示した。ア

サイクリックプログラムネットにおいて、 d_0 から始まる発火系列全体の集合を求めるには、各 SWITCH-node の制御フロー枝上のトークンの値を調べる必要がある。SWITCH-node が多ければ、すべてのトークンの値を調べることは現実的ではない。そこで SWITCH-node の発火則を制御フロー枝上のトークンの値によらず、出力枝のいずれかにトークンを出力するように変更した場合について検討した。そして、その場合にアサイクリックプログラムネットのサブクラスが部分的不活性かどうかを判定する問題が coNP 完全であることを示した。本来の問題は、この問題以上に難しいので、手に負えないといえる。

4.3 トークン Self-clean 性

(a) 定義

プログラムネットの各トークンはデータフロープログラムのデータを表している。データフロープログラムの実行が終了した時、データが残留しないことが望ましい。これは残留したデータがあると、そのプログラムは再入可能 (reentrant) でなくなる恐れがあるからである。プログラムネットの形式で、この性質はすべてのノードが発火できなくなったとき、すべての有向枝上にトークンがないことを意味する。この性質をトークン Self-clean 性という。

トークン Self-clean 性は以下のように定義される。

[定義 9] (トークン Self-clean 性⁽⁴⁵⁾) 初期トークン分布 d_0 のプログラムネット $PN=(N, E, \alpha, \beta)$ は、(i) 停止し、かつ (ii) 任意の停止発火系列によって d_0 から変換されたトークン分布 d が $d=0$ ならば、トークン Self-clean であるという。

□

(b) 判定法

初期トークン分布を $d_0 = 0$ と仮定する。

アサイクリック SWITCH-less プログラムネット $PN=(N, E, \alpha, \beta)$ がトークン Self-clean であるための必要十分条件⁽⁴⁵⁾は、実行中に有向枝 e に現れるトークンの総数を δ_e とすると、(i) 各有向枝 $e \in E$ に対し δ_e / α_e が整数であり、かつ (ii) 各 AND-node z に対し、その任意の二つの入力ノードを x と y とすると、 $\delta_{(x,z)} / \alpha_{(x,z)} = \delta_{(y,z)} / \alpha_{(y,z)}$ であることである。すべての有向枝 $e \in E$ に対する δ_e は幅優先探索に基づいて計算できる。したがって、アサイクリック SWITCH-less プログラムネットがトークン Self-clean であるかどうかは多項式時間で判定できる。一方、有向閉路のある SWITCH-less プログラムネットはトークン Self-clean でない。

一般的なプログラムネットに対しては、必要条件だけが知られている。詳細は文献⁽⁴⁵⁾を参照されたい。

4.4 並列度

(a) 定義

プログラムネットが表すデータフロープログラムはデータ駆動方式に従って実行される。この方式はプログラム自身が本来持つ潜在的な並列性を引き出すことが期待される⁽⁹⁾。しかしなが

ら、その並列性を発揮するには、その並列度に応じた十分な数のプロセッサが必要である⁽⁴⁶⁾。Gurd ら⁽⁴⁷⁾はデータフローコンピュータ上の実験を通じてデータフロープログラムの並列度を測定している。Patnaik⁽⁴⁸⁾らと Ghosal ら⁽⁴⁹⁾はシミュレーションを通じてデータフロープログラムの並列度を測定している。データフロープログラムが大規模になったり、複雑になると、実験やシミュレーションを通じて並列度を測定することは難しくなる。したがってデータフロープログラムや、それを表すプログラムネットの並列性を解析的に測定することが求められている。

並列度 $PARAdeg$ は以下のように定義される。

[定義 10] (並列度 $PARAdeg$ ⁽¹⁰⁾) 初期トークン分布 d_0 のプログラムネット $PN=(N, E, \alpha, \beta)$ ($\forall e \in E : \alpha_e = \beta_e = 1$) に対し、ノード z の発火時間を $\tau(z)$ と記すと、

(1) d_0 から実行可能な停止する発火系列 σ に対する $PARAdeg$ を次のように定義する。

$$PARAdeg(PN, \sigma) \stackrel{\text{def}}{=} \frac{\sum_{z \in N} \tau(z) \bar{\sigma}(z)}{T_{\sigma}(\text{inf})}$$

ここで $\bar{\sigma}(z)$ は σ の中で z が発火する回数、 $T_{\sigma}(\text{inf})$ は σ が十分なプロセッサによって実行された場合の最短発火時間である。

(2) PN の $PARAdeg$ を次のように定義する。

$$PARAdeg(PN) \stackrel{\text{def}}{=} \max_{\sigma \in \Sigma} \{PARAdeg(PN, \sigma)\}$$

ここで Σ は d_0 から実行可能な停止する発火系列全体の集合である。

□

(b) 計算法

初期トークン分布を $d_0 = 0$ 、アークの重みを $\forall e \in E : \alpha_e = \beta_e = 1$ と仮定する。

AND プログラムネットの $PARAdeg$ は多項式時間で計算可能である⁽⁵⁰⁾。これは AND プログラムネットの各ノードの発火回数は高々 1 回であり、幅優先探索に基づいて開始ノードから各ノードの発火完了時刻を計算していけばプログラムネット全体の最短発火時間が計算できるからである。

アサイクリック SWITCH-less プログラムネットの $PARAdeg$ は準多項式時間で計算可能⁽¹⁰⁾である。これはアサイクリック SWITCH-less プログラムネットのノードは複数回発火する可能性があるため、プログラムネット全体の最短発火時間を計算するには、最大発火回数だけ繰り返す必要があるからである。

更に筆者ら⁽⁵¹⁾は、SWITCH-node がある一般のアサイクリックプログラムネットの $PARAdeg$ を計算する問題が手に負えないことを示した。具体的には、順次、分岐、並列という基本的な制御構造からなるアサイクリックプログラムネットのサブクラスを定義し、その $PARAdeg$ を計算する問題が NP 完全であることを示した。更に、そのクラスに対して、その $PARAdeg$ を多項式時間で計算可能なヒューリスティックアルゴリズムを構築した。

図 4 アサイクリックプログラムネットに対するヒューリスティックアルゴリズム⁽³⁶⁾による並列度 $PARAdeg$ の計算例

4.5 解析例

プログラムネットの解析例として、図 4(a) に示すプログラムネット PN_2 の $PARAdeg$ を計算する。 PN_2 の各ノードの発火時間は 1 とする。 PN_2 は SWITCH-node を持つため、その $PARAdeg$ を計算する問題は手に負えない。しかし、 PN_2 は順次、分岐、並列という基本的な制御構造からなるアサイクリックプログラムネットのサブクラスに属する。そこで、文献⁵¹⁾のヒューリスティックアルゴリズムが適用できる。

そのヒューリスティックアルゴリズムは、分岐構造が入れ子になっていることに着目し、入れ子の最も深い部分から部分的な $PARAdeg$ の値が大きくなるように、発火系列を選択していく。分岐構造の開始点である SWITCH-node と終了点である OR-node の組を C-pair という。C-pair は深さ優先探索に基づいて、入れ子の深さが同じ SWITCH-node と OR-node を発見することにより求められる。各ノードの入れ子の深さを図 4(b) に示す。 PN_2 には二つの C-pair (z_3, z_{16}) と (z_6, z_9) がある。次に、再び開始ノード s から深さ優先探索を行い、訪れたノードが C-pair の OR-node ならば、対応する SWITCH-node から OR-node までの部分的な $PARAdeg$ の値が大きくなるように、

発火系列を選択する。 z_9 を訪問したとき (図 4(c))、その C-pair (z_6, z_9) の間には、二つの発火系列 $\sigma_{True}=z_7$ と $\sigma_{False}=z_8$ が存在する。

$$PARAdeg(PN_2, z_7) = 1.0 = PARAdeg(PN_2, z_8)$$

したがって σ_{True} が選択される。同様に、 z_{16} を訪問したとき (図 4(d))、その C-pair (z_3, z_{16}) の間には、二つの発火系列 $\sigma_{True}=z_4z_5z_6z_7z_9$ と $\sigma_{False}=z_{10}z_{11}z_{13}z_{12}z_{14}z_{15}$ が存在する。

$$PARAdeg(PN_2, z_4z_5z_6z_7z_9) = 5/5 = 1.0 <$$

$$PARAdeg(PN_2, z_{10}z_{11}z_{13}z_{12}z_{14}z_{15}) = 6/4 = 1.5$$

したがって σ_{False} が選択される。最終的に、発火系列 $sz_1z_2z_3z_{10}z_{11}z_{13}z_{12}z_{14}z_{15}z_{16}t$ が選択される。

$$\begin{aligned} PARAdeg(PN_2, sz_1z_2z_3z_{10}z_{11}z_{13}z_{12}z_{14}z_{15}z_{16}t) \\ = 12/10 = 1.2 \end{aligned}$$

この値が $PARAdeg(PN_2)$ の近似値として出力される。

5. おわりに

本解説ではネット理論の応用として、ワークフローネットとプログラムネットを紹介した。まずワークフローネットとプログラムネットの概念と定義を説明した。次にワークフローネットに関する解析問題として、健全性と振舞い継承、動的変更時間を取り上げ、その解析方法について紹介した。またプログラムネットに関する解析問題として、停止性と不活性、トークン Self-clean 性、並列度を取り上げ、その解析方法について紹介した。

最後にワークフローネットとプログラムネットに関して挑戦すべき課題を紹介しよう。ワークフローネットに関して、van der Aalst⁽⁵²⁾は 10 個の課題を提起した。その中には本稿で紹介した解析問題のほかに、デザインパターンやプロセスマイニングなどの興味深い課題が含まれている。またプログラムネットに関しては、マルチコアプロセッサの資源割当てへの応用や並列コンパイラ的设计への応用など、実用的な課題も挙げることができる。本解説がネット理論に対する興味を持つきっかけになれば幸いである。

文 献

- (1) J.L. Peterson, Petri Net Theory and the Modeling of Systems, Prentice Hall, 1981.
- (2) T. Murata, "Petri nets: Properties, analysis and applications," Proc. IEEE, vol.77, no.4, pp.541-580, April 1989.
- (3) S. Kumagai, "Activities on net theory in Japan," IEICE Trans. Fundamentals, vol.E77-A, no.7, pp.1125-1131, July, 1994.
- (4) W.M.P. van der Aalst and K. van Hee, Workflow Management: Models, Methods, and Systems, MIT Press, Jan. 2002.
- (5) K. Onaga and W.K. Chen, "Data flow analysis of program nets," J. Franklin Inst., vol.313, issue 4, pp.219-231, Oct. 1982.

- (6) S. Yamaguchi, Q.W. Ge, and M. Tanaka, "Performance evaluation on change time of dynamic workflow changes," IEICE Trans. Fundamentals, vol.E83-A, no.11, pp.2177-2187, Nov. 2000.
- (7) S. Yamaguchi, K. Kuniyoshi, Q.W. Ge, M. Tanaka, "Computation methods of maximum throughput for MG/ SMWF-nets with conflict-free resources," IEICE Trans. Fundamentals, vol.E87-A, no.11, pp.2868-2877, Nov. 2004.
- (8) T. Yuba, "Research and development efforts on dataflow computer architecture in Japan," J. Inf. Process., vol.9, no.2, pp.51-60, Sep. 1986.
- (9) W.M. Johnston, J.R.P. Hanna, and R.J. Millar, "Advances in dataflow programming languages," ACM Comput. Surv., vol.36, no.1, pp.1-34, March 2004.
- (10) Q.W. Ge, T. Watanabe, and K. Onaga, "Analysis of parallelism in autonomous execution of data-flow program nets," IEICE Trans. Fundamentals, vol.E74-A, no.10, pp.3008-3017, Oct. 1991.
- (11) Q.W. Ge, and N. Yoshioka, "An optimal two-processor scheduling for a class of program nets via a hybrid priority list," 情処学論, vol.40, no.5, pp.2064-2071, May 1999.
- (12) W.M.P. van der Aalst, "The application of Petri nets to workflow management," J. Circuits, Syst., Comput., vol.8, no.1, pp.21-65, June 1998.
- (13) W.M.P. van der Aalst and T. Basten, "Inheritance of workflows: An approach to tackling problems related to change," Theor. Comput. Sci., vol.270, no.1-2, pp.125-203, Jan. 2002.
- (14) WfMC, Terminology & Glossary, WfMC-TC-1011, 1999.
- (15) 山口真悟, 葛 崎偉, 田中 稔, "ペトリネットによるワークフローのモデル化と解析," システム/制御/情報, vol.45, no.8, pp.480-489, Aug. 2001.
- (16) J. B. Dennis, "First version of a data flow procedure language," LNCS, vol.19, pp.362-376, May 1974.
- (17) Q.W. Ge, T. Watanabe, and K. Onaga, "Execution termination and computation determinacy of data-flow program nets," J. Franklin Inst., vol.328, no.1, pp.123-141, 1991.
- (18) K. van Hee, N. Sidorova, and M. Voorhoeve, "Soundness and separability of workflow nets in the stepwise refinement approach," LNCS, vol.2679, pp.337-356, Nov. 2003.
- (19) S. Yamaguchi, Y. Shiode, Q.W. Ge, and M. Tanaka, "Performance evaluation on transient time of dynamic workflow changes," IEICE Trans. Fundamentals, vol.E84-A, no.11, pp.2852-2864, Nov. 2001.
- (20) H.M.W. Verbeek, "Verification of WF-nets," PhD Thesis, Technische Universiteit Eindhoven, 2004.
- (21) 金子侑史, 山口真悟, 葛 崎偉, 田中 稔, "ワークフローネットの健全性と並列度について," 信学技報, CST2005-41, pp.13-18, Jan. 2006.
- (22) W.M.P. van der Aalst, "Verification of workflow nets," LNCS, vol.1248, pp.407-426, Aug. 1997.
- (23) J. Desel and J. Esparza, Free choice Petri nets, Cambridge University Press, 2005.
- (24) H. Hayami and K. Okada, "From workflow to inter-workflow - Supporting to business to business electronic commerce," 情処学論, vol.42, no.3, pp.432-447, March 2001.
- (25) S. Yamaguchi, H. Matsuo, Q.W. Ge, and M. Tanaka, "WF-Net based modeling and soundness verification of interworkflows," IEICE Trans. Fundamentals, vol.E90-A, no.4, pp.829-835, April 2007.
- (26) 太田 淳, 辻 孝吉, "ネット理論-ペトリネットとその解析問題-, 信学 FR 誌, vol.2, no.4, pp.56-67, April 2008.
- (27) S. Yamaguchi, M. Yamaguchi, and M. Tanaka, "A model checking method of soundness for acyclic workflow nets using the SPIN model checker," Int. J. Information, vol.12, no.1, pp.163-172, 2009.
- (28) M. Yamaguchi, S. Yamaguchi, and M. Tanaka, "A model checking method of soundness for workflow nets," IEICE Trans. Fundamentals, vol.E92-A, no.11, pp.2723-2731, Nov. 2009.
- (29) H.M.W. Verbeek, and T. Basten, "Deciding life-cycle inheritance on Petri nets," Proc. of ICATPN 2003, pp.44-63, 2003.
- (30) W.M.P. van der Aalst, and T. Basten, "Life-cycle inheritance: A Petri-net-based approach," LNCS, vol.1248, pp.62-81, Aug. 1997.
- (31) 山口真悟, 成井徹志, 葛 崎偉, 田中 稔, "引継型と請負型のインターネットワークフローの振る舞い継承について," 第 49 回自動制御連合講演会, SU5-1-7, Nov. 2006.
- (32) WfMC, Interface 4 - Interoperability - Abstract Specification, WfMC-TC-1012, 1996.
- (33) C. Ellis, K. Keddera, and G. Rozenberg, "Dynamic change within workflow systems," Proc. of ACM Conference on Organizational Computing Systems, pp.10-21, 1995.
- (34) S.W. Sadiq, O. Marjanovic, and N.E. Orłowska, "Managing change and time in dynamic workflow processes," Int. J. Coop., Inf. Syst., vol.9, no.1&2, pp.93-116, March and April 2000.
- (35) S. Yamaguchi, A. Mishima, Q.W. Ge, and M. Tanaka, "A flexible and efficient workflow change type: Selective shift," IEICE Trans. Fundamentals, vol.E88-A, no.6, pp.1487-1496, June 2005.
- (36) S. Yamaguchi, K. Miyauchi, Q.W. Ge, and M. Tanaka, "Performance evaluation on worst change time of Flush and SCO dynamic changes for state machine WF-nets," IEICE Trans. Fundamentals, vol.E89-A, no.6, pp.1701-1704, June 2006.
- (37) S. Yamaguchi, A. Mishima, Q.W. Ge, and M. Tanaka, "Modeling and performance evaluation on change time for Migrate dynamic workflow changes," IEICE Trans. Fundamentals, vol.E86-A, no.6, pp.1466-1475, June 2003.
- (38) G.J. Holzmann, The Model Checker Spin, Addison-Wesley, 2003.
- (39) SPIN model checker, <http://spinroot.com/>
- (40) S. Yamaguchi, K. Yamada, Q.W. Ge and M. Tanaka, "State equation of program nets and its application to reachability for SWITCH-less nets," Int. J. Information, vol.7, no.4, pp.515-526, 2004.
- (41) Q.W. Ge and H. Yanagida, "A method of computing minimum firing time for self-cleaning SWITCH-less program Nets," J. Franklin Inst., vol.335B, no.5, pp.877-895, July 1998.
- (42) S. Yamaguchi, K. Komiya, Q.W. Ge, and M. Tanaka, "On verification of marking-dependent terminacy for data-flow program nets," Proc. of ITC-CSCC 2008, pp.973-976, 2008.
- (43) 山田紘輔, 山口真悟, 葛 崎偉, 田中 稔, "データフロープログラムネットの標準形およびその表現能力," 信学技報, CST2003-34, pp.5-10, Dec. 2003.
- (44) S. Yamaguchi, K. Yamada, Q.W. Ge, and M. Tanaka, "Dead problem of program nets," IEICE Trans. Fundamentals, vol.E89-A, no.4, pp.887-894, April 2006.
- (45) Q.W. Ge and K. Onaga, "On verification of token self-cleanness of data-flow program nets," IEICE Trans. Fundamentals, vol.E79-A, no.6, pp.812-817, June 1996.
- (46) Q.W. Ge, "PARAdeg-processor scheduling for acyclic SWITCH-less program nets," J. Franklin Inst., vol.336, no.7, pp.1135-1153, Sep. 1999.
- (47) J.R. Gurd, C.C. Kirkham, and I. Watson, "The Manchester prototype dataflow computer," Commun. ACM, vol.28, no.1, pp.34-52, 1985.
- (48) L.M. Patnaik, R. Govindarajan, and N.S. Ramdoss, "Design and performance evaluation of EXMAN: An extended Manchester dataflow computer," IEEE Trans. Comput., vol.C-35, no.3, pp.229-243, March 1986.
- (49) D. Ghosal and L.N. Bhuyan, "Performance evaluation

of a dataflow architecture,” IEEE Trans. Comput., vol.39, no.5, pp.615–627, July 1990.

- (50) T. Watanabe, S. Yamaguchi, Q.W. Ge, and M. Tanaka, “On computation of PARAdeg of data-flow program nets with SWITCH-nodes,” 信学技報, CST2004, pp.7–12, 2005.
- (51) S. Yamaguchi, T. Takai, T. Watanabe, Q.W. Ge, and M. Tanaka, “Complexity and a heuristic algorithm of computing parallel degree for program nets with SWITCH-nodes,” IEICE Trans. Fundamentals, vol.E89-A, no.11, pp.3207–3215, Nov. 2006.
- (52) W.M.P. van der Aalst, “Making work flow: On the application of Petri nets to business process management,” LNCS, vol.2360, pp.1–22, Oct. 2002.

(平成 21 年 10 月 30 日受付 平成 21 年 11 月 13 日最終受付)

山口真悟 (正員)

平 4 山口大・工・電子工卒。平 6 同大学院工学研究科博士前期課程了。平 9 同大学院理工学研究科博士後期課程退学，同大学工・助手。現在，同大学院理工学研究科准教授。工博。ネット理論，コンカレント工学，ソフトウェア工学などの研究に従事。平 19 米国イリノイ大シカゴ校訪問研究員。平 19 からコンカレント工学研究専門委員会幹事。

葛崎偉 (正員)

昭 58 中国復旦大・電子工程系卒。昭 62 広島大学院工学研究科博士前期課程了。平 3 同大学院工学研究科博士後期課程了。工博。同年富士通テン(株)入社。平 5 山口大・助教授。以来，ネット理論，並列処理システム，ソフトウェア解析等の研究に従事。平 16 同大学教授，現在に至る。平 19–20 コンカレント工学研究専門委員会委員長。