プログラミング-言語・基礎・実践 14－3
（1993．10．29）

リアクティブ・データフロー型ドキュメントブラウザのための ドキュメント・プログラミング

福田 晴元　　高橋 直久
harumoto@nttspe.ntt.jp　　naohisa@nttspe.ntt.jp

NTT ソフトウェア研究所

ソフトウェア改造時におけるドキュメント分析作業を支援するため，我々は先に，リアクティブ・データフロー型ドキュメントブラウザを提案した．本ブラウザでは，ドキュメント分析過程をデータフロープログラムにより表現する．さらに，そのプログラムをユーザの違いを表すユーザレベルに適応させながら実行することによりドキュメント表示を行う．本稿では，ユーザレベルに応じてノードを選択的に実行させるために必要なレベル付きデータフロープログラム，および，その実行制御法について述べる．さらに，レベル付きデータフロープログラムにおける構造とデータフローに関する妥当性の検証法について述べる．

Document Programing for Reactive Data Flow Document Browser

Harumoto FUKUDA　　Naohisa TAKAHASHI
harumoto@nttspe.ntt.jp　　naohisa@nttspe.ntt.jp

NTT Software Laboratories

This paper describes the programming and the execution control mechanism of a document browser called a reactive data-flow document-browser. The browsing process uses a data-flow program that displays software and hardware documents specified in document analysis process. Program execution is based on a data-flow computation model with reactive functions. The model can be expanded to enable the execution control mechanism to adaptively control data flow according to user levels. This paper also presents a program analysis method that can detect illegal program structures and dataflow anomalies by interpreting the leveled dataflow programs with a generalized dataflow computing model.

1 はじめに

ソフトウェアの開発では，ソースコードをはじめ，設計文書等様々な技術文書，テスト結果などの生産物が，各種ツールにより作り出される．ソフトウェアの改造では，既存システムの機能を理解し，変更の影響を考察するために，これらの生産物（本稿では「ドキュメント」と呼ぶ）の中から必要なドキュメントを捜し出し，その内容を理解する作業が必要となる．

このようなドキュメントの分析を容易にするため，ハイパテキスト・システムがソフトウェア開発支援環境でも利用されている[1]．ドキュメントの作成時，あるいは，分析時に，ソースコードの断片に対して，関連のあるドキュメントへのリンクを付与しておけば，リンクを辿ることにより関連するドキュメントを読み進めることができる．しかし，リンクの辿り方は，ユーザに任されているため，ドキュメント間にリンクを張った者の意図が十分に伝わらず，ユーザはドキュメントの内容を理解する以外に，リンクの意味を正確に理解するための作業が必要となる．一方，ドキュメントの間には複雑な相互関係があるため，従来の多くのオーサリングシステムによりシナリオとして表示順にドキュメントを列挙することは困難である．

ハイパテキストのリンクの辿り方をペトリネットで表現し，表示順序を制御する方式[5]が提案されている．この方式では，ペトリネットの意味規則に従い，リンクが辿られ，ドキュメントが非同期並列に表示される．また，ペトリネットの性質を用いて，リンクの到達可能性解析を行ない，表示過程の誤りの検出作業を助けることができる．しかし，この方式では，ペトリネットは並列実行動作の性質を求めるためのモデルであるためプログラミング言語として記述性が低い．また，すべてのユーザに対して同一の表示を行うため，基礎知識を多く持つユーザには冗長な表示となり，基礎知識の少ないユーザには理解し難い表示となるという問題がある．

上記の問題を解決するため，我々は，リアクティブ・データフローモデルと呼ぶドキュメント・ブラウジング・モデルを提案し，それに基づくブラウザの作成を進めている[3][8][9]．このモデルでは，データフロー計算モデルに基づいてドキュメント間のリンクの辿り方に関する意味付け法を与え，構造的な並列プログラムとしてドキュメントの表示過程を記述する．さらに，記述されたプログラムをデータフロー計算モデルにより並列実行することによりドキュメントの表示過程を再現する．このとき，リアクティブ制御機能[6][7]と呼ぶ実行制御機能により，ユーザが置かれた局面に応じて再現法を変化させている．

本稿では，上記ブラウザのためのプログラム，すなわち，ドキュメントの表示過程を制御するプログラムを記述するドキュメント・プログラミングについて述べる．さらに，記述されたプログラムの実行制御法と検証法を提案する．ここでは，一般化したデータフロー計算モデル[2]に対して発火規則と実行規則を与えることにより実行制御法と検証法を実現する手法を示す．

2 リアクティブ・データフロー型ブラウザ

リアクティブ・データフロー型ブラウザは，表示するドキュメントをノードとし，ドキュメント間の依存関係をアークとしたデータフローグラフをデータフロー計算モデルに従って実行することによりドキュメントの表示順序を制御する[9]．

図 1にデータフローグラフの実行例を示す．図において小さい黒丸はトークン，網掛けの丸は分配ノード，白丸はドキュメントノード，斜線の丸はドキュメント表示中のドキュメントノードを表す．四角は画面イメージを表す．図ではトークンが揃ったノードが実行され，そのノードに対応するドキュメントが画面に表示される様子を表している．実行機構の詳細については 4章で述べる．

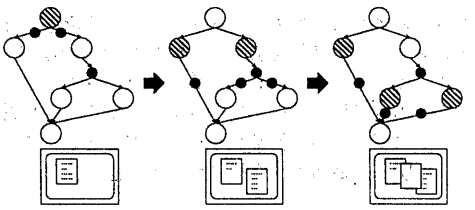

図 1: 実行例

3 ドキュメントプログラミング

3.1 ドキュメントプログラミングの概要

ドキュメントを処理対象とするプログラムを記述することは，ドキュメントプログラミングと呼ばれている[4]．ここでは，特に，ドキュメントの表示順序制御を対象とする処理記述をドキュメントプログラミングと呼ぶ．以下で単にプログラムという場合には，このドキュメントプログラミングにより記述されたプログラムをいう．以下ではドキュメントプログラミングの概要を示す．ノードとアークについては 3.2, 3.3節で述べる．

(1) **ドキュメント選択** ドキュメント群の中より表示に必要なドキュメントを選択する．

(2) **依存関係** (1) で選択したドキュメントに対し，3.3節で述べる依存関係を表すアークを張る．

(3) **ドキュメントへのレベル付け** どのレベルの人に対して表示を行うかを示すレベルをドキュメントに付与する．レベルについては 3.2節で述べる．

(4) **データフロープログラム作成 (関数化)** (2) で作成したグラフに実行の流れを制御するノードを付加して，データフロープログラムの関数に変換する．

(5) **レベル付きデータフローグラフ作成** ドキュメントに対応するノードに対して，ドキュメントと同じレベルを与える．

表 1: 環境レベルの満たすべき条件

ノード種別	入力アーク先ノードに起因する条件	出力アーク先ノードに起因する条件
ドキュメント, 演算, 開始, 分配, 環境読み取り, 環境設定, レベル読み取り, レベル設定	$level(n) \leq level_{in}(n,i)$ なる i が存在する	$level(n) \leq level_{out}(n,j)$ なる j が存在する
手続き呼び出し, 合流	$level(n) \leq level_{in}(n,1) = level_{in}(n,2)$	$level(n) \leq level_{out}(n,1)$
分岐	$level(n) \leq level_{in}(n,1) = level_{in}(n,2)$	$level(n) \leq level_{out}(n,1) = level_{out}(n,2)$
手続き復帰	$level(n) \leq level_{in}(n,2)$	-

3.2 ノード

プログラムで使用するノードには，以下に示す機能とレベルを記述する．

3.2.1 ノードの機能

プログラム実行時に，ノード間ではトークンと呼ぶデータの授受を行う．トークンは，演算値，環境値，実行レベルと呼ぶ 3 つのフィールドからなる．演算値を更新するノードを演算ノード，環境値，実行レベルを更新するノードを環境制御ノード，トークンの流れを制御するノードをフロー制御ノード，ドキュメントを表示するノードをドキュメントノードと呼ぶ．それぞれのノードの機能について以下に述べる．

演算ノード　入力トークンを与えるとトークンの演算値に対し処理を行い，その結果をトークンの演算値として返すノードであり，組み込みノードとユーザ定義ノードがある．組み込みノードはブラウザで予め用意した，論理演算，四則演算などの基本機能を持つ．ユーザ定義ノードはユーザが入力トークンの演算値に対する処理をノード機能として記述しブラウザに登録しておくノードである．

環境制御ノード　環境制御ノードは，環境読み取り・環境設定・レベル読み取り・レベル設定ノードからなる．環境読み取りノードは，入力トークンの環境値を出力トークンの演算値に設定する．環境設定ノードは，入力トークンの演算値を出力トークンの環境値に設定する．レベル読み取り・レベル設定ノードは，トークンの実行レベルについて同様の処理を行う．

フロー制御ノード　フロー制御ノードには，分配，分岐，関数呼び出し，開始，終了ノードがある．これらのノードは，通常のデータフロー計算モデルでの制御ノードと同様の機能[11]を持ち，トークンの伝幡経路を以下のように制御する．

分配ノードは入力アーク上のトークンのコピーを全ての出力アーク上に出力する．分岐ノードは，2 個の入力トークンを受信し，第 2 入力トークンの演算値（真偽値）により定められる出力アークに第 1 入力トークンを送り出す．関数呼び出しノードは，引数データを運ぶトークンを受信し，引数データ，環境値，出力先アークの組を演算値とし，新たに生成した値を環境値としたトークンを関数の開始ノードに送る．開始ノードは，受信したトークンが運ぶ演算値のうち，引数データを関数本体の最初のノードに送り，環境値と出力先アークとの対を終了ノードに送る．終了ノードは，これらの環境値と出力先アークに従い，関数本体のノードから送られたトークンの演算値を呼び出し元のノードに返す．

ドキュメントノード　入力トークンを与えると，ノードに与えられたドキュメント名に一致するドキュメントを表示し，入力トークンの演算値をトークンの演算値として返す．

3.2.2 ノードのレベル

以下のようにドキュメントをレベル分けし，ドキュメントノードに対して，対応するドキュメントのレベルと等しい値 (環境レベルと呼ぶ) を付与する．ドキュメントノード以外のノードに対しては，表 1に示す条件に従って環境レベルを定める．

レベルの高いユーザが必要とするドキュメントとレベルの低いユーザが必要とするドキュメントには違いがある．従って，レベルの高いユーザに合わせてプログラムを作成すると，レベルの低いユーザには理解し難い表示が行われることとなり，逆に，レベルの低いユーザに合わせてプログラムを作成すると，レベルの高いユーザには冗長な表示を行うこととなる．そこで，レベルの高いユーザほど少ないドキュメントで情報を取得し，レベルの低いユーザはレベルの高いユーザが必要とするドキュメントに加えて更に多くのドキュメントから情報を取得すると考えて，レベルの高いユーザが必要とするドキュメントの集合と，レベルの低いユーザが必要とするドキュメントの集合は包含関係にあると定める[8]．

ここで，ユーザに対してユーザレベル $l_i (i = 1, 2, \ldots, n)$ を与える．$j < k$ のとき $l_j > l_k$ とし，l_j は l_k よりレベルが高いという．ユーザレベル l_m の開発者が必要とするドキュメントの中でユーザレベル l_{m-1} 以上の開発者は必要としないドキュメントをレベル l_m のドキュメントと呼ぶ．

3.3 アーク

プログラムにおけるアークはノードの実行順序を表す．この順序関係は，ドキュメント相互の局所的な依存関係に基づいて定められる．依存関係としては，図 2(1),(2) に示す 2 つの関係を基本とする．ここで (1) は「ドキュメント A を理解してからドキュメント B を見る」という関係，もしくは，「ドキュメント B を見るためにはドキュメント A が必要」という関係を表す．次に (2) は「ドキュメント A を理解するためにはドキュメント B が必要であり，ドキュメント B を理解するためにはドキュメント A が必要」という相互依存関係を表す．(1),(2) を一般化すると (3),(4),(5) となる．(3) は「ドキュメント A を理解

したら，ドキュメントB1...Bnを見る」「ドキュメントB1...Bnを理解するためにはドキュメントAが必要」という関係である．(4)は「ドキュメントA1...Anを理解したらドキュメントBを見る」「ドキュメントBを理解するためには，ドキュメントA1...Anが必要」という関係である．(5)は「ドキュメントAiを理解するためにはドキュメントA1...Ai−1...Ai+1...Anが必要」という関係が$i = 1,\ldots,n$について存在することを意味する．

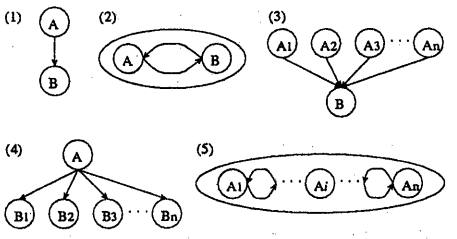

図 2: ドキュメントの依存関係

データフロープログラムでは，（1），（3），（4）の場合にはノード間に順序づけのアークを張り，（2），（5）の場合にはノードが同時に発火可能になるように記述する．

3.4 実行レベルの動的設定

ある実行レベルでプログラムを実行させたときに表示されるドキュメントに対して，ユーザが冗長な表示が行われていると感じたり，もしくは，表示されたドキュメントだけでは良く理解出来ない場合が考えられる．これは，実行レベルがユーザレベルを正しく反映していないためである．この場合には，プログラムの実行中に表示を見ているユーザのレベルを推定し，その推定結果に基づいて表示レベルを変更する必要がある．このための基本機能として，本ブラウザでは環境制御ノードにより実行レベルを実行時に変更させる手段を与える．

図 3に環境制御ノードを用いたプログラム例を示す．この図でDはドキュメントノード，Uはユーザ定義ノード，ELはレベル読み取りノード，SLはレベル設定ノードを示す．まず，ドキュメントノードDがテスト用のドキュメントを表示する．次に演算ノードUがそのドキュメントに関する質問を表示し，ユーザからの解答を得る．さらに，解答に従い真偽値を定め，演算値として出力する．分岐ノードはこの真偽値に従いELノードが読み出した実行レベルを分岐させ，1増加または減少させる．SLノードは送られた値を実行レベルとしてトークンに設定する．

4 プログラムの実行制御

通常のデータフロー計算機構では，その内部に発火制御の取り決めを記述した発火規則と，ノードの実行機能を記述した実行規則が埋め込まれている．このため，同

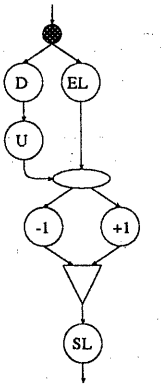

図 3: 実行レベルの動的設定プログラム例

じデータフローグラフを与えると同じ計算を行う．これに対して，一般データフロー計算[2]では，図 4に示すように，発火規則と実行規則を一般化し，処理に応じて実行機構の外部から与えることにより，同じデータフローグラフに対して種々の解釈実行を行なえるようにしている．以下で述べる発火規則と，3.2.1節で述べたノードの機能からなる実行規則を一般データフローインタプリタに与えることにより，リアクティブデータフロー型ドキュメントブラウザが実現される．

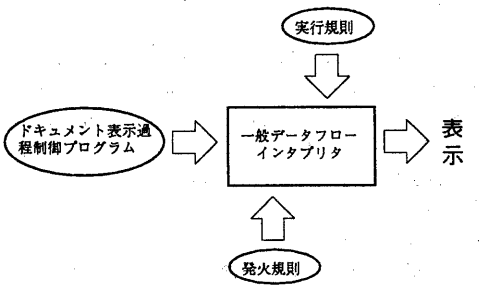

図 4: 一般データフロー計算によるドキュメント表示

表 2: ノードのタイプ

タイプ	特徴	ノード例
type1	全ての入力データが実行に必要なノード	分岐，関数呼び出し，開始，終了
type2	一部の入力データだけでも実行可能なノード	ドキュメントノード，演算（加算）
type3	いずれか一つの入力データだけで実行可能なノード	合流ノード

まず，実行に必要な入力データの違いに従って，ノードを表 2のように3つのタイプに分類する．次に，ノードn_1からn_2に向かうアークaを考える．l_1，l_2をn_1，

n_2 の環境レベルとし，l_n を l_1 と l_2 の最小値とする．このとき，a は $l \leq l_n$ なる l でアクティブであるといい，n_2 は，そのタイプに応じて次の条件を満たすとき，実行レベル l で発火可能となる．

- 発火規則

  type1 のとき　全ての入力アーク上に実行レベル $l(\leq l_n)$ のトークンが揃う．

  type2 のとき　$l \leq l_n$ の任意の l でアクティブな全ての入力アーク上に，実行レベル l のトークンがそろう．

  type3 のとき　任意の入力アークに実行レベル $l(\leq l_n)$ のトークンが到着する．

5　プログラムの性質

5.1　プログラム構造

データフロープログラムでは，任意の構造を許したとき，プログラムの構造が複雑になり，無限ループやトークンの衝突等の誤りがあっても検出できず，動作が理解困難であるようなプログラムが作られる場合がある．そこで，次のような簡明で良構造のプログラムを本ブラウザで扱うプログラムとして許し，これ以外の構造のプログラムを構造異常として排除する．

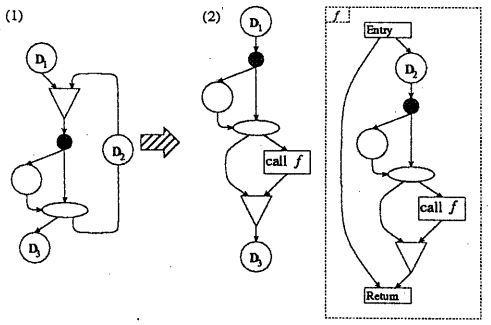

図 5: プログラムの良構造化の例

- 単一代入制約　同一のアーク上を流れるトークンは高々1つとし，繰り返し処理は再帰として記述する．例えば，図 5(1) に示すループ構造は，1つのアーク上を複数回トークンが流れて単一代入性を損なうので誤りである．このような構造は図 5(2) の関数で表現する．

- 分岐処理の構造化　分岐処理は分岐ノードと合流ノードが一対一に対応したネスト構造となる．例えば，図 6(1) の構造は分岐処理の構造化の構造化がなされていない．このような場合は図 6(2) のネストした構造として記述しなければならない．

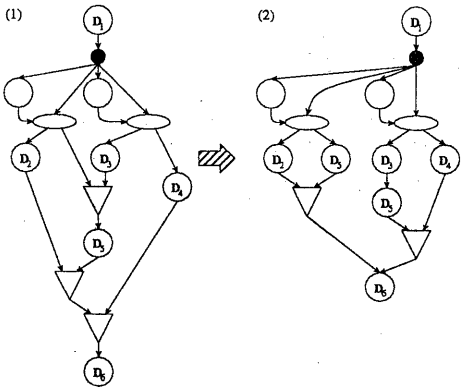

図 6: プログラムの良構造化の例

5.2　データフロー異常

前節で示した構造異常がない場合でも，プログラム構造上の誤りや，レベル付けの誤りにより，ある実行レベルで表示すべきドキュメントが表示されない場合がある．このような場合には，プログラムはデータフロー異常があるという．データフロー異常は表 3に示すような4種類の異常に分類される．図 7に各異常の発生例を示す．図では，異常を発生させたノードを斜線で示している．

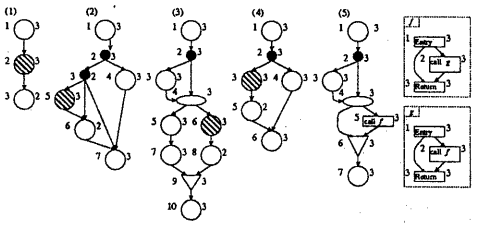

図 7: 異常プログラムの例

(1)　ノード2は出力アークにレベル3のアクティブアークを持たないためトークンの伝幡が停止する．

(2)　ノード5は環境レベル3であるが，入力アークにレベル3のアクティブアークをもたないので，決して発火しない．

(3)　ノード6は出力アークにレベル3のアクティブアークを持たないためトークンの伝幡が停止する．ただし，分岐ノードが真のとき，終了ノードまで到達可能であるので可到達性異常ではない．

(4)　ノード3は出力アークにレベル3のアクティブアークを持たないためトークンの伝幡は停止する．

(9)　関数 f と g の相互再帰が無限に続くので停止しない．

表 3: データフロー異常

異常名	異常内容	例
可到達性異常	開始ノードから終了ノードまでトークンが伝幡する経路が存在しない.	図 7(1)(4)
伝幡異常	実行を中断するノードが存在する.	図 7(3)(4)
全実行性異常	トークンが伝幡して来ないために実行されないノードが存在する.	図 7(2)
再帰異常	関数の相互再帰呼び出しによりサイクルが構成される.	図 7(5)

6 レベル付きプログラムの妥当性の検証

6.1 検証の概要

プログラムは，局所的な関係に基づいて断片情報をつなげて作成され，レベル付けされる．プログラムを作成したときに，プログラム全体として構造異常やデータフロー異常がないことを調べるのは必ずしも容易でない．プログラムを実行させて異常箇所を探す方法は，試行錯誤的になり，多大な時間を要することとなる．そこで，プログラムを解析し，異常箇所を検出する検証法を与える．

この検証では，妥当であると判定したプログラムに対しては，構造異常およびデータフロー異常がないことを保証する．また，妥当でないと判定したプログラムに対して少なくともひとつの異常発生箇所を提示する．

構造異常およびデータフロー異常は次の 2 つの解析により実現される．

- 関数内解析　関数内部を解析することにより構造異常，データフロー異常を検出する．
- 関数間解析　関数間に渡る可到達性を調べることにより再帰異常を検出する．

6.2 関数内解析

6.2.1 関数内解析の概要

図 4に示した一般データフローインタプリタに以下の発火規則と実行規則を与えて，関数本体を実行させることにより関数内解析を行う．

- 発火規則　ノードの種類に関わらず，全ての入力アーク上にトークンが揃った時点で発火可能にする．
- 実行規則　6.2.2節，6.2.3節に示す構造異常解析機能とレベル別データフロー異常解析機能によりトークンを処理する．

この検証のための実行 (以降では検証実行といい，4 章で述べた通常のプログラム実行（通常実行という）と区別する) は．実行規則の各機能に対する初期値を与えた初期トークンを，各関数の開始ノードに与えることにより開始され，発火可能なノードがなくなると停止する．

検証実行は，次のように実行が制御されるので必ず停止することが保証される．

- プログラム中にサイクルがあっても，発火規則によりサイクルの開始するノード，つまり合流ノードが発火しないため，サイクルは実行されない．
- 関数呼び出しがあっても 6.3節で述べるように，呼び出し先関数を展開する代わりに，その関数の解析結果の近似値を用いるので相互再帰が無限に続くことがない．

5.1節に示した単一代入規則に関する異常のうち，サイクルをなす異常は検証実行の停止時に発火しない合流ノードとして現れる．サイクルをなさない単一代入規則に関する異常，および，分岐処理の構造化に関する異常は 6.2.2節で述べる構造異常解析機能により検出される．また，データフロー異常のうち可到達性異常，伝播異常，全実行性異常は 6.2.3節で述べるレベル別データフロー異常解析機能により検出される．再帰異常は，6.3節に述べる関数間解析を収束するまで繰り返したとき，可到達性異常のある関数の存在として現れる．

6.2.2 構造異常検出機能

検証実行では，ノードの種別と環境レベルによらず，すべてのノードに対してすべての出力アークにトークンを出力させる．プログラム中にサイクルがないとき，分岐ノードが一つだけのプログラムでは，合流ノードで，全ての入力アーク上に同じ分岐ノードの異なる出力アークから送られたトークンが揃い，その他のノードで，全ての入力アーク上に同じ分岐ノードの同じ出力アークから送られたトークンが揃うか，分岐ノードを通らないトークンが揃った場合に，単一代入規則に関する異常，および，分岐処理の構造化に関する異常がないといえる．分岐ノードが複数あるプログラムでは，トークンが通過した経路を考慮する必要がある．このため，プログラムの経路解析手法[10]と同様に，次のような経路識別子を導入する．

まず，すべての分岐ノードの出力アークに対して，そのノードの識別子，および，出力アークを表す方向（真または偽）からなる 2 つ組（経路断片識別子と呼ぶ）を割り付ける．さらに，経路断片識別子の系列（経路識別子と呼ぶ）により，検証実行においてトークンが通過した経路を表す．たとえば，経路識別子 $c_1 = <(sw1_1, dl_1), (sw1_2, dl_2), \ldots, (sw1_n, dl_n)>$ は，分岐ノード $sw1_1, sw1_2, \ldots, sw1_n$ を順に辿り，各ノードで方向 $dl_1, dl_2, \ldots, dl_n$ を選択することにより一意に定まる経路を表す．ここで，経路識別子 $c_2 = <(sw2_1, d2_1), (sw2_2, d2_2), \ldots, (sw2_n, d2_n)>$ を考える．このとき，c_1 と c_2 がともに空であるか，c_1 と c_2 の対応する経路断片識別子が互いに等しいとき，c_1 と c_2 は等しいという．また，$sw1_1 = sw2_1, id1_1 = id2_1, \ldots, sw1_{n-1} = sw2_{n-1}, id1_{n-1} = id2_{n-1}, sw1_n = sw2_n, id1_n \neq id2_n$ ならば，c_1 と c_2 は兄弟の関係にあるという．これらの識別子は，$n-1$ 個の分岐ノードを同じ経路で辿り，第 n 番目の分岐ノードで異なる出力アークを辿ることを表している．

上記の経路識別子を用いて，次のようにノードの実行規則を定め，経路識別子を運ぶトークンを用いて検証実行させると，サイクルをなさない単一代入規則に関する異常，および，分岐処理の構造化に関する異常がある場合には検証実行を中断させるノードが出現する．

- 分岐ノード　入力トークンの経路識別子が等しくない場合には実行を中断させる．各出力アークに対して，それぞれノード断片識別子を求め，入力トークンの経路識別子に付与してできる新たな経路識別子を運ぶトークンを出力する．
- 合流ノード　入力トークンの経路識別子が兄弟の関係でない場合には実行を中断させる．兄弟の関係にある場合は実行を続け，入力トークンの経路識別子の最後の経路断片識別子を取り除くことによりできる経路識別子を運ぶトークンを出力する．
- その他のノード　入力トークンの経路識別子が等しくない場合には実行を中断させる．等しい場合は実行を続け，入力トークンと同じ経路識別子を運ぶトークンを出力する．

6.2.3　レベル別データフロー異常解析機能

開始ノードから該アークに至る経路を p としたとき，実行レベル m に対するアークの状態は次のように分類できる．

1. 実行レベル m の通常実行ではトークンは到達する．
2. 実行レベル m の通常実行では，該アークにトークンは到達しない．
   (a) 実行レベル m の通常実行の際に経路 p において異常中断するノードが存在する．
   (b) 実行レベル m の通常実行の際に経路 p において異常中断しない．これはさらに次の 2 つの場合に分けられる．
      i. $m \leq l$ なる環境レベル l を満たすノードが実行されない．
      ii. 上記以外である．

上記により分類される 4 つの状態，1，2-(a)，2-(b)-i，2-(b)-ii，をそれぞれ記号 T，R，E，N で表す．これらの記号を全ての実行レベルについて並べた一次元ベクトルをアーク状態ベクトルと呼ぶ．s_1,s_2 をアークの状態としたとき，$s_1 \oplus s_2$, $s_1 \otimes s_2$, $s_1 \odot s_2$ なる演算を次のように定義する．

$$s_1 \oplus s_2 = \begin{cases} T & (s_1 = T, s_2 \neq R) \text{ または} \\ & (s_2 = T, s_1 \neq R) \text{ のとき} \\ N & (s_1 = s_2 = N) \text{ のとき} \\ R & (s_1 = R) \text{ または } (s_2 = R) \text{ のとき} \\ E & (\text{その他}) \end{cases} \quad (1)$$

$$s_1 \otimes s_2 = \begin{cases} T & (s_1 = s_2 = T) \text{ のとき} \\ N & (s_1 = s_2 = N) \text{ のとき} \\ R & (s_1 = R),\ (s_2 = R),\ (s_1 = T, s_2 \neq T), \\ & (s_1 \neq T, s_2 = T) \text{ のいずれかのとき} \\ E & \text{その他} \end{cases} \quad (2)$$

$$s_1 \odot s_2 = \begin{cases} T & (s_1 = T) \text{ または } (s_2 = \text{T}) \text{ のとき} \\ R & (s_1 = R, s_2 \neq T) \text{ または} \\ & (s_1 \neq T, s_2 = R) \text{ のとき} \\ E & (s_1 = s_2 = E) \text{ のとき} \\ N & (\text{その他}) \end{cases} \quad (3)$$

アーク状態ベクトル S_1=(s_{11},s_{12},...,s_{1n})，S_2=(s_{21}, s_{22},...,s_{2n}) に対して，$S_1 \oplus S_2$,$S_1 \otimes S_2$,$S_1 \odot S_2$ なる演算を次の様に定義する．

$$S_1 \otimes S_2 = (s_{11} \otimes s_{21}, s_{12} \otimes s_{22}, \ldots, s_{1n} \otimes s_{2n}) \quad (4)$$

$$S_1 \oplus S_2 = (s_{11} \oplus s_{21}, s_{12} \oplus s_{22}, \ldots, s_{1n} \oplus s_{2n}) \quad (5)$$

$$S_1 \odot S_2 = (s_{11} \odot s_{21}, s_{12} \odot s_{22}, \ldots, s_{1n} \odot s_{2n}) \quad (6)$$

開始ノードの出力アークに対して，全て T からなるアーク状態ベクトルを運ぶトークン送ることにより検証実行を開始する．このとき，ノードの実行規則は，入力アークのアーク状態ベクトルから出力アークのアーク状態ベクトルを求める手順を与え，次のように記述される．

1. 全ての入力アーク上のトークンが運ぶアーク状態ベクトルの組に対して，ノードタイプ別に以下の演算を行う．この結果求まったベクトルを入力状態ベクトルと呼ぶ．これらの演算は通常実行の各ノードタイプの発火規則が対象としているアークについて条件が満されているか調べるものである．

   type1　$T_1 \otimes T_2 \otimes \ldots \otimes T_j \otimes \ldots$
   type2　$T_1 \oplus T_2 \oplus \ldots \oplus T_j \oplus \ldots$
   type3　$T_1 \odot T_2 \odot \ldots \odot T_j \odot \ldots$

2. 入力状態ベクトルの第 m 要素を i_m，ノードの環境レベルを l とする．i_m =N(または E) かつ $m \leq l$ のとき，このノードは決して実行レベル m で発火可能とならない．この場合には E，その他の場合には i_m を値とする要素からなるベクトルをノード状態ベクトルと呼ぶ．

3. ノード状態ベクトルの第 m 要素を r_m，第 j 出力アーク先のノードの環境レベルを $l_{out}(j)$ とする．このとき，分岐ノードのようにレベルによらず，全ての出力アークにトークンを送出すべきノードでは，r_m =T かつ $m > l_{out}(j)$ のときならば，そのノードまでトークンが到達したのに実行レベル m ではトークンを出力できないことを意味する．この場合には R，その他の場合には r_m を値とする要素からなるベクトルを第 j 出力アークのアーク状態ベクトルとする．また，アクティブとなる出力アークにのみトークンを送出すべきノードに対しては，r_m =T のときに，全ての出力アークが実行レベル m でアクティブでないならば R，第 j 出力アーク以外の出力アークが実行レベル m でアクティブであれば N，その他の場合には r_m を値とする要素からなるベクトルを第 j 出力アークのアーク状態ベクトルとする．以上により求めた出力アークのアーク状態ベクトルを運ぶトークンを，それぞれ対応する出力アーク上へ送出する．

各実行レベルにおける可到達性異常，伝幡異常，全実行性異常の存在は，検証実行を行い，それぞれ次の状態を検出することにより判定できる．

- 可到達性異常
  終了ノードにトークンが伝幡してこない場合に，全ての実行レベルにおいて可到達性異常である．

  終了ノードのノード状態ベクトルの第 m 要素がRである場合に，実行レベル m で可到達性異常である．

- 伝幡異常
  検証実行時に発火しない，もしくは，トークンを出力できないノードが存在する場合に，全ての実行レベルにおいて伝幡異常である．

  入力状態ベクトルの計算で，アーク状態ベクトルの第 m 要素がTであり，結果がRとなった場合に，実行レベル m で伝搬異常である．

  出力アークに対するアーク状態の計算で，ノード状態ベクトルの第 m 要素がTであり，結果がRとなった場合に，実行レベル m で伝搬異常である．

- 全実行性異常
  トークンが到達しないノードが存在する場合に，全ての実行レベルにおいて全実行性異常である．

  ノード状態ベクトルの計算で，入力状態ベクトルの第 m 要素がNかEであり，結果がEとなった場合に，実行レベル m で全実行性異常である．

6.3 関数間解析

再帰異常がある場合には，関数間に渡る可到達性異常が存在する．そこで，プログラムに含まれる関数を $f_1, f_2, \ldots, f_n$ とし，それぞれの関数内解析による結果 $r_1, r_2, \ldots, r_n$ を利用して以下のように関数内解析を繰り返すことにより，関数間に渡る可到達性異常を検出する．

1. $f_1, f_2, \ldots, f_m, \ldots, f_n$ の解析結果の初期値 $r_1^1, r_2^1, \ldots, r_m^1, \ldots, r_n^1$ を全ての要素がNである一次元のベクトルと定める．

2. 第 i 回目の解析結果 $r_1^i, r_2^i, \ldots, r_n^i$ を用いて，第 $i+1$ 回目の解析結果 $r_1^{i+1}, r_2^{i+1}, \ldots, r_n^{i+1}$ を求める．このとき各関数の関数内解析では，関数呼び出しが起こった際に，呼び出し先関数の第 i 回目の解析結果を用いて関数内解析を行う．

3. 全ての関数について，第 i 回目と第 $i+1$ 回目の解析結果が等しくなるまで上の処理を繰り返す．

7 おわりに

リアクティブデータフロー型ドキュメントブラウザにおいて，ユーザレベルに応じてノードを選択的に実行させるために必要なレベル付きデータフロープログラムおよび，その実行制御法について述べた．さらに、レベル付きデータフロープログラムにおける構造とデータフローに関する妥当性の検証法について述べた．

実行制御系と検証系では，それぞれの発火規則と実行規則を一般データフロー計算モデルに与えることにより実現される．検証系では必ず実行が停止することが保証されるように発火規則と実行規則を与えている．また検証系では単一代入制約と分岐処理の構造化に関する構造異常，および，可到達性異常，伝幡異常，全実行性異常，再帰異常と呼ぶデータフロー異常を検出し，妥当であると判定したプログラムに対してこれらの異常がないことを保証する．

現在，実行制御系と検証系の作成を進めている．今後，作成したシステムを用いて評価実験を行うとともに，ユーザレベルの設定法を与えるユーザモデルについて検討する予定である．

最後に，日頃御討論頂くソフトウェア基礎技術研究部後藤滋樹部長，伊藤正樹リーダはじめ，ソフトウェア基礎技術研究部の皆様に深謝します．

参考文献

1) 高田広章, ハイパテキストとそのプログラミング環境への応用, 情報処理, Vol.30 No.4, pp406-413(1989).

2) 高橋, 鈴木, 直井, 福田, データフロー計算の一般化 - ソフトウェア・リエンジニアリングにおける 複雑な情報の構造化と理解を目指して -, 日本ソフトウェア科学会第 10 回大会, C9-2(1993).

3) 福田, 高橋, リアクティブデータフローモデル: ソフトウェア設計ドキュメントの表示過程の記述と再現, 情処学会 ソフトウェア工学研究会 90-3, pp17-24(1993).

4) 京嶋, 上林,N.V. ゴパル, 文書プログラミングについての一考察 情処学会 オフィスシステム研究会 OS88-53, pp27-35(1988).

5) P.David Stotts and R.Furuta, *Petri-Net-Based Hypertext:Document Structure with Browsing Semantics*, ACM Trans. Information Systems Vol.7 No.1, pp3-29(1989).

6) D.Harel and A.Pnueli, *On the Dovelopment of Reactive System*, Logics and Models of Concurrent Systems, NATO ASI Series, Vol.F13, pp3-29(1985).

7) Frederic Boussinot, *Reactive C:An Extension of C to Program Reactive System*, SOFTWARE -PRACTICE AND EXPERIENCE, Vol.21(4), pp401-428(1991).

8) 福田, 高橋, リアクティブデータフロー・モデルに基づくドキュメントブラウザの設計と実現, 情処学会第 46 回全国大会 5J-2(1993).

9) 福田, 高橋, リアクティブ・データフロー型ドキュメントブラウザにおけるユーザレベルへの適応表示制御, 情処学会 第 47 回全国大会 7J-4(1993).

10) 直井, 高橋, 経路依存フローグラフを用いた Infeasible Path 検出法, 信学論 Vol J76-D-I, No 8, pp429-439(1993).

11) 雨宮 真人, データフロー・アーキテクチャについて, コンピュータソフトウェア, Vol.1 No.1, pp42-63(1984).