

FDTって何？

東野輝夫

大阪大学

●FDT（形式記述技法）とは？●

近年の計算機ネットワークの発展に伴い、さまざまなネットワーク関連製品が開発されている。また、これらのシステムは年々高度化され、複雑になってきている。一般には、このような複雑なシステムの仕様は、英語や日本語などの自然語で書かれていたり、理解を深めるために図や表を交えて記述されている場合が多い。

自然語や図表による表現は、多くの人にとって馴染みやすく、理解しやすいものである。しかし一方では、あい矛盾するような記述をしてしまう場合や、記述に漏れのある場合や、記述に曖昧さが残る場合など、記述した仕様に問題がある場合も多い。しかも、そのような問題がある場合でも、自然語の記述からこれらの不具合などを見つけ出すのはそれほど容易ではない。その結果、作成したシステムに検出困難なバグが内在したり、同じ仕様に基づいて作成された通信装置であっても異なるメーカーの装置間ではうまく通信できないなどといった問題が生じる。

FDT（Formal Description Technique：形式記述技法）は、このような複雑なシステムの仕様を、あらかじめその意味が定義されている適当な言語で記述し、その記述から仕様の無矛盾性や曖昧性、完全性（記述に漏れないこと）などを検証や試験の手法を用いて解析することにより、開発するシステムの信頼性を向上させようとするものである。

従来、これらの解析にはスーパーコンピュータを何日も占有しなければならない場合も多く、人工衛星のソフトウェアのように高度の信頼性を要求されたり、電話の交換機のように多人数の利用者があるような特殊なシステムを除いて、時間とコストの面で採算が合わないことも多かった。

しかし、最近は安価で高速なワークステーションやパソコンが登場してきたため、それらの計算機を何台か使って、検証や試験を効率よく行おうとする試みも数多くなされている。また、検証・試験手法自身の高速化も進んでおり、市販の計算機を数時間から数日間

動かすことにより、検証や試験が行えるようになってきた。たとえば、(1) AT&Tで開発した通信ソフトウェアやATM用の通信システム、(2) NASAの人工衛星用プログラム、(3) ジャンボジェットなどの航空機のフライト制御システム、などといった実用システムの仕様をFDTで記述し、機械的な検証・試験手法を用いることにより、「数多くのバグが発見できた」とか、「システムの重要な部分の安全性の検証が行えた」、などといったFDTの適用事例が近年数多く報告されている。

●FDTの鍵は？●

通信システムやグループウェアなどの並行分散システムやソフトウェア開発プロセスなど、複数のプロセスや人間がお互いに通信しながら並行に動作するような協調システムを設計する場合には、各プロセスの仕様をどのように記述するかのみならず、複数プロセス間の通信やシステム全体をどのようにモデル化し、それらの仕様をどのように記述すればよいかが重要になる。また、そのモデル化の善し悪しによって、無矛盾性や完全性などの解析が容易になったり難しくなったりする。

このように、FDT（形式記述技法）には、(1) 開発したいシステムのタイプに応じたモデルの選択と、(2) そのモデルに適した形式仕様記述言語を用いた仕様記述、の2つの側面がある。

通信システムなどの仕様を記述する場合、(1) のモデル化は、たとえば、図-1のように複数のノード（計算機）間のメッセージのやりとりを記述したシーケンス図などから各計算機の動作を有限状態機械などでモデル化し、システム全体をそれら有限状態機械群の並行動作としてモデル化する場合が多い。また、計算機間の通信は、FIFOキューのような非同期通信としてモデル化される場合が多いが、送信と受信が同時に実行されるような同期通信モデルや、より抽象的な通信モデルを用いて、排他制御やマルチキャスト通信のメカニズムを簡潔に記述しようとする場合もある。

一方、(2) の形式仕様記述については、一般に広く利用されている形式仕様記述言語を用いて記述するこ

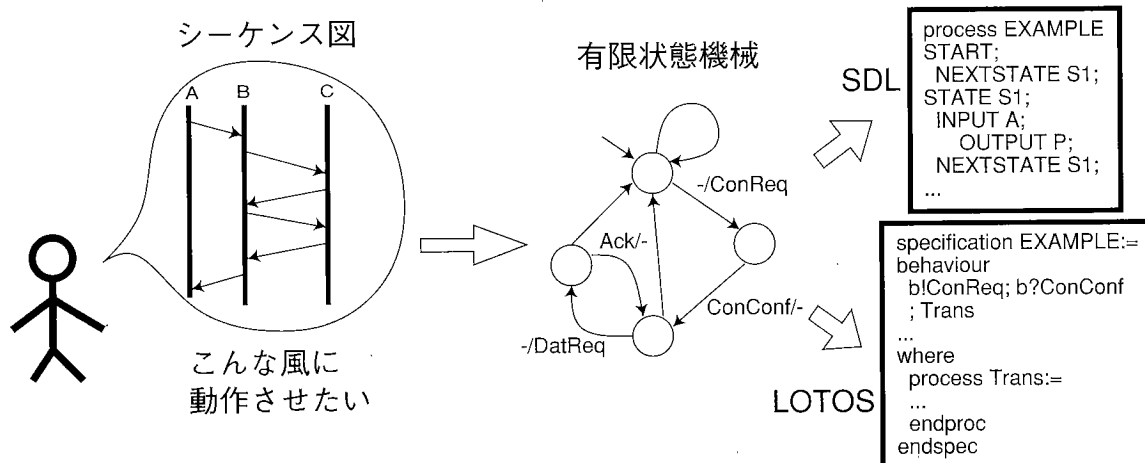


図-1 通信システムのモデル化と仕様記述

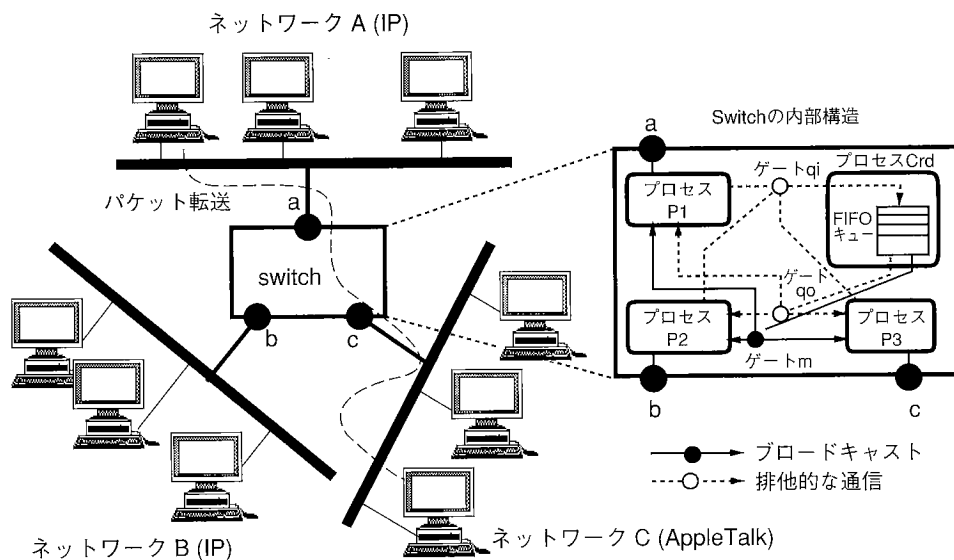


図-2 ネットワークスイッチの概念図

とが望ましい。これは、汎用の形式仕様記述言語には、仕様記述のためのエディタや検証、試験、実装のためのツールが数多く開発されていることが多いのである。たとえば、通信システムの汎用的な形式仕様記述言語としては、SDLやEstelle, LOTOS, MSC (メッセージシーケンスチャート) などが考案されている^{1), 2)}。

●FDTの記述例●

■簡単なネットワークスイッチの例■

図-2に示すような簡単なネットワークスイッチ (ルータ) の仕様をFDTで記述する方法について考えてみよう。図-2のスイッチは、3つの入出力ポート (ゲート) a, b, cを持ち、それぞれ異なるネットワークA, B, Cに接続されている。各パケットはその宛先のアドレスに従って該当するネットワークに転送される。スイッチに到着したブロードキャストメッセージは送信元のネッ

トワークを除くすべてのネットワークにブロードキャストされる。また、ここでは、ネットワークA, BがIPプロトコルを使用し、ネットワークCが異なるプロトコル (AppleTalk) を使用していると仮定している。

各ゲートの動作は他のゲートの動作とは独立して実行できることが望ましい。このため、a, b, cの3つのゲートをそれぞれP1, P2, P3の並行プロセスによって制御することを考える。また、このスイッチでは、各ゲートから入力されたパケットは共通のFIFOキューにいったん蓄えられ、キューの先頭から取り出されたパケットが順次該当するネットワークに送出されるものとする。キューは1つのコーディネータプロセスCrdによって制御され、キューの入力、出力のための内部ゲートとして、それぞれqi, qoを用いる。さらに、ブロードキャストの場合の出力ゲートとしてゲートmを用いる。ゲートmではコーディネータプロセスCrdから送られたパケ

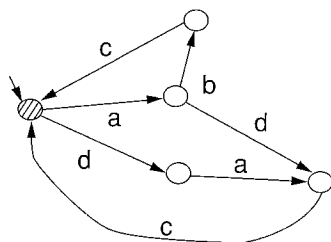
表-1 LOTOSのオペレータとその意味

オペレータの種類	表記	オペレータの説明
1. アクションプレフィクス	$\alpha; B$	イベント α の実行後、 B を実行。
2. 選択	$B_1 \square B_2$	B_1, B_2 のいずれかを選択実行。
3. 非同期並列	$B_1 \parallel B_2$	B_1, B_2 を非同期に並列実行。
4. 同期並列	$B_1 \parallel [G] B_2, B_1 \parallel B_2$	B_1, B_2 を同期並列実行。
5. 逐次	$B_1 >> B_2$	B_1 の実行後 B_2 を実行。
6. 割込	$B_1 \triangleright B_2$	B_1 の実行中 B_2 による割込可能。

P := Q || [b, c] || R
Q := a; (b; c; exit || c; exit) >> Q
R := (d; c; exit || b; c; exit) >> R

LOTOS仕様の例 (プロセスP)

- ・2つのプロセスQとRがゲートb, cで同期実行される。
- ・プロセスQのイベントaを最初に実行したら, b; c; exitのイベント列はすぐに同期実行できるが, c; exitの方のイベントcはすぐに同期実行できないので, プロセスRのイベントdを実行した後cが実行される。
- ・プロセスRのイベントdを最初に実行すると, すぐにはイベントcは同期実行できないので, プロセスQのイベントaを実行した後, cが実行される。



プロセスPで実行されるイベントの実行順
(●印が初期状態)

図-3 LOTOS仕様の動作

であり, “x” や “E” は入出力値を表す. 入出力動作をイベントと呼び, 各イベントは $a?x$ や $b!E$ のようにゲート名と入出力データの組で構成される. また, イベントの実行順序を定めるオペレータとして, 表-1のようないくつかのオペレータがある.

このうち, 同期並列実行を表す動作式 $B_1 \parallel [G] B_2$ では, B_1 と B_2 の動作の中でゲート名の集合Gに属するイベントの実行は同期して並列に(同時)実行されなければならない. また, たとえば, 動作式 “ $a?x; B_1 \parallel [a] a!f(z); B_2$ ” でイベントaが同期実行される際には, 右辺の出力動作 $a!f(z)$ の出力値 $f(z)$ が左辺の入力動作 $a?x$ の変数 x に受け渡される. これらの機構を用いてプロセス間のデータ通信を同期的に行う.

簡単なLOTOS仕様とその動作については図-3を参照されたい.

■LOTOSを用いた仕様記述の例■

上記のようなネットワークスイッチのLOTOS仕様 Switch[a,b,c]は, おおまかには表-2のように記述される.

●C言語で仕様は書けないの?●

アルゴリズムやデータ構造の教科書ではCやPascal, あるいは, それらに近い疑似言語を用いてアルゴリズムを説明している場合が多い. その意味では, CやPascalも立派な仕様記述言語である. それらの言語の意味もほぼ形式的に定義されているし, 何より, 言語の意味定義を具体的に示さなくても多くの人が同じように理解できるという利点がある.

それでは, CやPascal, あるいは, それらに近い疑似言語で仕様を書くことが望ましいのであろうか. もちろん, 教科書に載っているような簡単なプログラムの仕様を書く場合は, CやPascalで書くことが望ましい場合もある. しかし, ちょっと複雑なシステムを記述する場合, 複雑なデータ構造を用いることが多く, データ構造を抽象化する機能を持った言語が必要になってくる. たとえば, 上記のLOTOS仕様の例では, IPパケットのデータを抽象的にデータ型IPで表現し, その内容やIPパケットに対する関数(たとえば関数dst(odt)やsrc-address(odt)の定義)は階層的に別途記述する

ットがプロセスP1, P2, P3にブロードキャストされるものとする. また, コーディネータプロセスCrdは, ゲートqi, qoでは同時にはP1, P2, P3のいずれか1つのプロセスからしかアクセスできないものとする.

このレベルでのネットワークスイッチの仕様を考える場合, ゲートmでのブロードキャストやゲートqi, qoでの排他的な通信のメカニズムの実現法はもっと実装に近いレベルで考慮すべきものとして隠蔽し, 上述のような抽象的な通信機構を持つ仕様記述言語を用いて簡潔に仕様を書くことが望まれる. このような方針で仕様を書く場合, 以下で紹介するLOTOSなどでシステム全体の仕様を記述することが望ましい.

■仕様記述言語LOTOS■

LOTOS (Language Of Temporal Ordering Specification)³⁾ は, プロセス代数と呼ばれる数学的なモデルに基づく言語で, 1989年にISOの国際標準として規格制定された. LOTOSでは, 入力 $a?x$ のように “?”を用いて表現し, 出力 $b!E$ のように “!”を用いて表現する. “a” や “b” は入出力ポート (ゲート) の名前

表-2 ネットワークスイッチのLOTOS仕様

LOTOS プログラム	プログラムの説明
<pre> specification Switch[a,b,c] := (P1[a,qi,qo,m](address-A) [m] P2[b,qi,qo,m](address-B) [m] P3[c,qi,qo,m](address-C)) [qi,qo,m] Crd[qi,qo,m] where P1[a,qi,qo,m](address-A) := ... P2[b,qi,qo,m](address-B) := In[b,qi] Out[b,qo,m](address-B) In[b,qi] := b?idt:IP; qi!idt; In[b,qi] Out[b,qo,m](address-B) := qo?odt:IP[dst(odt)=address-B]; b!odt; Out[b,qo,m](address-B) [] m?odt:IP[src-address(odt)=address-B]; Out[b,qo,m](address-B) [] m?odt:IP[not(src-address(odt)=address-B)] b!odt; Out[b,qo,m](address-B) P3[c,qi,qo,m](address-C) := ... Crd[qi,qo,m] := ... </pre>	・システム全体は3つのプロセスP1, P2, P3とプロセスCrdからなる。 ・ブロードキャスト用のゲートmでの動作はP1, P2, P3が同時に行う。 ・ゲートqi, qo, mを介してお互いにデータ交換を行う。 ・プロセスP1はプロセスP2とほぼ同じ。 ・P2は2つのプロセスIn, Outの並行プロセスとして記述。 ・ゲートbからIPパケットidtを読み、それをFIFOキュー(ゲートqi)に出力、という動作を繰り返す。 ・FIFOキューの出力ゲートqoからIPパケットを読み、それをゲートbに出力する。 ・ブロードキャストパケットodtが送られてきたら、自ノードからのパケットでない限り、ゲートbに出力する。 ・P3では、Apple TalkパケットをIPパケットに変換する部分の記述などが必要。 ・Crdの仕様はキューの仕様と類似(省略)。

プロセスOutにおいて条件式dst(odt)=address-Bが真のときパケットの送り先がP2であることを表す。また、ゲートmから送られたパケットが自分のゲートbから受信したパケットであるときに限り条件式src-address(odt)=address-Bが真になる。

というスタイルをとっている。

また、上記のネットワークスイッチの例などでは、複数のプロセス間の通信を記述する必要が出てくる。もちろん、C言語などでもソケットやパイプを用いた通信機能を用いることができるが、それらの機能だけでは、上記のゲートmでのブロードキャストやゲートqi,qoでの排他的なプロセス間通信などを簡潔に記述できない。これは、C言語などのプログラム言語では、あまり高度な通信機能が利用できないことに起因する (図-4参照)。

大雑把な捉え方をすると、次のようなイメージとなる。

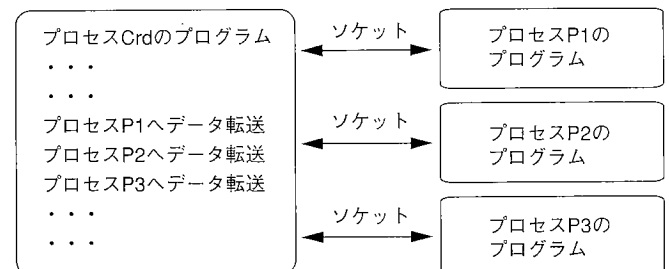
C言語+抽象データ型+高度な通信機構
＝通信システム向けFDT

●FDTはソフトウェアの設計開発に役立つのか？●

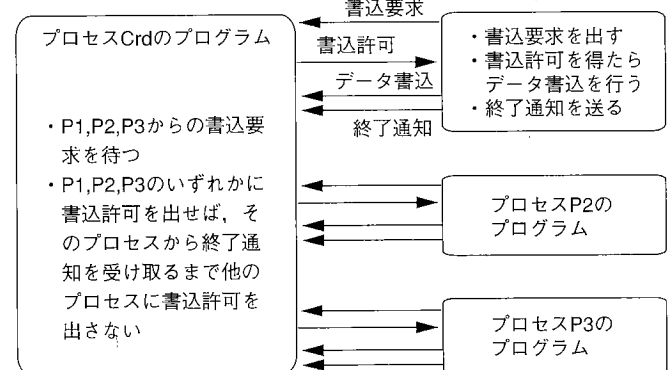
FDTでシステムの仕様を形式的に記述するとどのようなメリットがあるのだろうか。仕様を形式的に記述することにより、仕様の無矛盾性や曖昧性、完全性などの証明や試験が容易になる。

このうち、システムの論理的な正しさの証明については、必ずしも完全な機械化が進んでいるわけではないが、HOLやPVS, CafeOBJなどいくつかの言語やその証明支援システムが開発されており、実用化も進んでいる。通常、証明はインタラクティブな操作が必要なため、プログラムの証明に関する簡単な数学的知識を要する場合もある。しかし、近年の検証ツールでは、証明によく利用されるいくつかの推論規則が計算機に

CやPascalでのブロードキャスト



CやPascalでの排他制御



ネットワークスイッチの本質的な動きに加えて、上記のようなブロードキャストや排他制御のための手続きが加わり、仕様が分かりにくくなるとともに、仕様のサイズもかなり大きくなる。

図-4 C言語によるブロードキャスト・排他制御

組み込まれ、かなりのサイズのシステムの正当性の証明が機械的、あるいは、半機械的に行えるようになってきた。たとえば、NASAの土星探査機の故障対処プログラムの仕様をPVSと呼ばれる言語で記述し、安全性の検証を行うことにより数十個の不具合を発見した、などという報告もある。

また、携帯電話システムの実行手順を断片的に記述し、それらの記述からシステム全体のLOTOS仕様を構成したり、逆に、システム全体の仕様からいくつかの典型的な実行手順を構成する方法なども考案されている⁴⁾。さらに、ネットワークで接続されたシステム全体の仕様とI/Oポートやデータベースがどの計算機に配置されているかの配置指定から、各計算機がどのようにデータ交換を行いながら動作すれば全体の仕様通りにイベントを実行できるかを記述した各計算機ごとの動作仕様を自動生成するための研究もかなり実用化されている²⁾。

通信システムの分野では、作成したシステムが要求仕様を満足するかどうかの適合性試験が重要視されている。このような適合性試験では、システムの仕様をいくつかの状態変数を含む拡張有限状態機械、あるいは、それらの組で記述し、その記述からシステム全体として実行可能なすべての全状態を機械的に構成し、それらの全状態から実行可能な遷移を少なくとも1回は実行するような試験系列を構成し、その試験系列を実際に実装したシステムに与えてシステムの誤りを発見する、などといった手法がよく用いられる。一般に、全状態を機械的に網羅しようとする、すぐにその数がたとえば数十億通りとかそれ以上になるが、最近では安価で高速なパソコンを用いて、実際の試験を行うことが可能になってきている。たとえば、ベル研やフランステレコム、オタワ大学などでSSCOP-ATMプロトコルや移動体通信システムの試験系列を自動生成し、開発したシステムのバグを発見するのにかなり役立った、などといった事例報告がある^{5), 6)}。

●これからのFDT●

近年数多くのパソコンソフトが安価に手にはいるようになってきている。しかし、パソコンソフトは、「バグがあるのが当たり前」という風潮で販売され、正規ユーザには最新のバグフィックス版が配布される、などといったサービスが行われている。しかし、これからはホームオートメーションなどパソコンソフトより信頼性を求められる装置組み込みのソフトウェアが数多くでてくることが考えられ、近い将来、電話やネットワークから自宅のお風呂やポットを沸かせるとか、帰宅前に自宅のエアコンで室内を適温にしておく、などといったことがあたり前になるかもしれない。しかし、頼みもしないにお風呂が空焚きになったり、温度コントロールのミスで小さな子供の寝ている部屋の

室温が異常に高くなったり、などといったことが起これば、笑い話では済まなくなる。

もちろん、こういったホームオートメーションの装置にはフェイルセーフの機構が働くので、簡単には異常事態は生じないかもしれない。しかし、「このソフトの利用によって不利益が生じて責任は負えません」などといって売っているパソコンソフトと違って、家電製品や多くのホームオートメーション装置などハード主体の製品では「製造物責任」が厳しく問われる時代である。これは、電話の交換機やオンラインシステム、高度道路交通システムなどの社会システムの場合も同様であり、バグの発生は社会的に大きな問題となる。したがって、信頼性の高いソフトウェアを開発することは今後ますます重要になってくる。

信頼性の高いソフトウェアを開発するための手法としては、オブジェクト指向で設計を行うとか、ソフトウェアの開発プロセスを明確化するとか、十分な時間をかけて試験を行うとか、いろいろなことが考えられるが、FDTを用いて仕様を形式的に記述し、その記述からさまざまなツールを用いて機械的に検証や試験を行う、という手法も大いに役立つと考えられる。

ただし、現状では、簡単に利用できるFDTやその処理系、ユーザインタフェースの充実した検証ツール、などが十分そろっているとはいえない。また、SDLやEstelle、LOTOS、MSCといった複数の言語が並行してISOなどで標準化される背景には、モデル化したい問題に多様性があり、対象とする問題の記述に適したモデルや言語を複数考えざるを得ないという点がある。ソフトウェア開発者がどのモデルや言語を選べばよいのかや、どのようなモデル化が検証や試験に適しているのか、などに関する解説書やチュートリアル的な書物、実際の応用例にFDTを適用した結果に関する事例報告、などの充実が望まれる。また、ISOなどで標準化された言語の機能拡張や、さらなるツールの充実が望まれる。

安価なネットワークやパソコンの普及とFDT技術の発展に伴い、ようやく実用的な問題にFDTを適用できる環境が整ってきたように思われる。今後のこの分野の発展を期待したい。

参考文献

- 1) 水野忠則監修: プロトコル言語, カットシステム社 (1994).
- 2) 佐藤, 長谷川, 東野, 大崎: 形式的手法による分散システムへの取り組み, 日本ソフトウェア科学会, 講習会資料シリーズ, No.15 (1998).
- 3) ISO: Information Processing System, Open Systems Interconnection, LOTOS - A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour, IS 8807 (1989).
- 4) Tuok, R. and Logrippo, L.: Formal Specification and Use Case Generation for a Mobile Telephony System, Computer Networks and ISDN Systems, Vol. 30, pp.1045-1063 (1998).
- 5) Lee, D. and Yannakakis, M.: Principles and Methods of Testing Finite State Machines - A survey, Proc. of the IEEE, Vol.84, No.8, pp.1090-1123 (1996).
- 6) Cavalli, A., Lee, B.-H. and Macavei, T.: Test Generation for the SSCOP-ATM Networks Protocol, SDL Forum'97 (1997).

(平成10年7月23日受付)