ソフトウェア基礎論 29−4
(1989. 5. 18)

LOTOS実行系の並列処理環境

Parallel Execution Environment for LOTOS Specifications

野村真吾　長谷川亨　瀧塚孝志
Shingo NOMURA　Toru HASEGAWA　Takashi TAKIZUKA
国際電信電話株式会社 上福岡研究所
KDD Kamifukuoka R & D Labs.

あらまし　LOTOSは、ISOで標準化された通信プロトコルの仕様記述言語の一つである。筆者らは、LOTOSにより記述された仕様を、プログラムに変換して実行するLOTOS実行系を検討した。LOTOSの実行系の構築において、そのプロセスの並列動作の実現が重要であり、特に相手が動的に変化し複数のプロセス間で実行されるプロセス間通信に問題があることを明らかにした。LOTOSのプロセス間通信は、既存のOSや並列プログラミング言語の機能により実現することが困難であるため、独自のプロセス管理を実行するスケジューラを実行系に組込むことで実現した。スケジューラが、プロセスの動作を管理する同期待ちキューで同期の関係を保持することにより、プロセス間通信を実現する。本稿では、LOTOS実行系を構成するスケジューラおよびトランスレータの実現法について述べる。

Abstract　LOTOS is one of the specification languages for communication protocols, and is standardized by ISO. Described in this paper are software tools executing specifications in LOTOS. They consist of a translator from LOTOS specifications to C programs, and a scheduler executing LOTOS processes in parallel. In developing these tools, it is highly important to implement parallel processes of LOTOS, especially the interprocess communications where the synchronized processes are determined dynamically. The mechanism to implements interprocess communications is also described, in which the data structures representing relations among synchronized processes are introduced.

1. はじめに

通信ネットワークの高度化に伴い、通信機能を実現する通信プロトコルが大規模化している。自然言語等により通信プロトコル仕様を記述すると、自然言語の曖昧性から解釈間違い等の問題が生じる。ISOやCCITTの標準化機関では、通信プロトコルを曖昧なく記述するための形式記述技法(FDT: Formal Description Technique)の重要性を考慮して、LOTOS, Estelle, SDLのFDTを標準化している。

この内LOTOS[1,2](Language of Temporal Ordering Specification)は、時間的順序モデルに基づいて、通信プロトコルを規定するFDTである。LOTOSは、分散並列システムの仕様記述に適しており、任意の抽象レベルでの記述が可能であるためシステムの仕様をその実装に依存しないで記述ができるという特徴を持つ。また数学的モデルに基づく言語であるため仕様の検証、適合性試験に向いている。現在ISO等でLOTOSによりOSIの通信プロトコル仕様を記述する作業が行われており[3,4]、また開放型分散処理(ODP)仕様記述への適用も検討されているため、今後広く通信プロトコル仕様の記述に用いられる可能性が高い。またLOTOSを、一般的な分散システムの仕様の記述に適用する試みも行われている[5]。

LOTOSによる仕様の検証、実行等を行うツールが必要であると考えられ、現在LOTOSによる仕様をインタプリタ形式により実行するツールが検討されている[2,6]。これらのインタプリタでは、人間が対話的に仕様を実行する方式で実現されている。筆者らは、検証済みのLOTOSによる仕様を、実機用の言語に人手で翻訳することなく、通信システムを実装できるようにすることを目的として、LOTOSによる仕様をプログラムに変換するトランスレータ(コンパイラ)を

用いたLOTOS仕様の実行系を検討している。

LOTOSによる仕様は並列に動作するプロセスの集合により記述するため、トランスレータにより変換したプログラムを効率的に実行させるためには、LOTOSプロセスの並列動作の実現法が重要な課題である。特に、複数のプロセスが同時に同期可能であり複雑な同期要求が記述できるLOTOSのプロセス間通信の実現が問題となる。LOTOSのプロセス間通信をOSや既存の並列プログラミング言語の機能により実現することが困難であったため、筆者らはプロセスの実行・同期通信機構を効率よく実現する並列プロセス処理環境(スケジューラ)を組込んだLOTOS実行系の実現を検討している[7,8]。

本論文では、LOTOSによる仕様からプログラムを生成して実行する実行系の概要を示し、スケジューラによるLOTOSプロセスの実現を中心に実行系の実現法について述べる。2章でLOTOSの概要を説明し、3章で実行系の構成について述べる。4章ではスケジューラに関して、並列プロセスおよびプロセス間通信の実現法を中心に述べる。5章では仕様からプログラムを生成するトランスレータについてその概要を述べる。

2. LOTOS

LOTOSは、CCS[9]やCSP[10]に見られるプロセス代数法(process algebraic methods)を基礎としており、プロセスの動作およびインタラクションの記述法の多くは、CCSやCSPのものを取り入れている。ここで、インタラクションとはプロセス間での情報の送受のことであり、特にLOTOSではイベントと呼ばれる。一方、データ構造や値の記述は、抽象データ型(ADT: Abstract Data Type)を基礎としており、その概念の多くはACT ONE[11]のものを取り入れている。以下では、プロセスおよびイベントの記述法を中心にLOTOSの概要を説明し、実行系を作成する際に特に問題となったプロセス間通信の特徴を述べる。

2.1 LOTOSの概要

LOTOSによる仕様では、ひとまとまりの動作をプロセスとして記述し、結合オペレータを用いてプロセスの関係を記述する。LOTOSのプロセスは、他のプロセス及び仕様の外部(環境)と交換する情報の出入口であるゲートを持つブラックボックスと考えられる。プロセスの動作は、ゲートにおいて観測される、他のプロセス及び環境との入出力応答であるイベントの時間的順序を列挙することにより記述される。

```
specification two_slot_buffer[ inp, outp] :=
behaviour
 hide middle in
  buffer[ inp, middle] |[middle]| buffer[ middle, outp]
where
 process buffer[ inp, outp] :=
  inp ?value:int; outp ! value; buffer[ inp, outp]
 endproc
endspec
```

図1　LOTOSによる仕様記述例

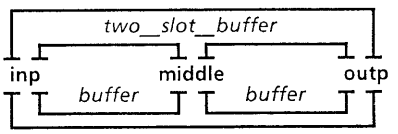

図2　図1の仕様のイメージ

2個の整数を記憶できるバッファの記述例及びイメージを図1,2に示す。プロセス“two_slot_buffer”は、2つのゲート“inp”,“outp”を持ち、2個のプロセス“buffer”が並列に結合されている。プロセス“buffer”は、2つのゲート“inp”,“outp”を持ち、ゲート“inp”から入力の整数を受け取り、ゲート“outp”にその整数を出力して、自身と同じ仕様を持つプロセス“buffer”を生成する。従って、“two_slot_buffer”では、左側のプロセス“buffer”がゲート“inp”から入力されてゲート“middle”を介して出力した整数を、右側のプロセス“buffer”がゲート“middle”からその整数値を受け取りゲート“outp”に出力することにより、2つの整数を蓄えるバッファを実現している。ゲート“middle”は“hide”と指定されることにより、環境とは独立なゲートとして宣言され、2つのプロセス“buffer”間で共有される。ゲート“middle”で実行されるイベントは、環境とは独立にこの2つのプロセス間で実行される。

(1) イベント

イベントは、ゲートを介した他のプロセスとの入出力を記述し、“!E”と記述される値宣言(Eは特定のデータ値)と、“?x:t”と記述される変数宣言(xは変数名、tはデータ型(ソート)識別子)から構成される。イベントは、一個のゲート識別子と、一個以上の値宣言および変数宣言の組合せで記述される。ゲートを共有する複数のプロセスで発行されたイベントは、そのゲートに対応する値宣言および変数宣言が表1に示す同期条件を満たす時、同期して実行される。同期条件とその後処理は、論理型言語に見られるユニフィケーションと同じである。図1のプロセス“buffer”における“inp ?value:int”および“outp !value”はイベントの

process A	process B	sync. condition	interaction sort	effect
g!E1	g!E2	value(E1) = value(E2)	value matching	synchronization
g!E	g?x:t	value(E) ∈ domain(t)	value passing	after synchronization x = value(E)
g?x:t	g?y:u	t = u	value generation	after synchronization x = y = v for v∈domain(t)

表1　LOTOSにおける同期条件と後処理

記述例である。

また、ガードを用いて値宣言および変数宣言に制約を加えることにより、複雑な同期要求が記述できる。

例えば"[x>0]−>g !x ?y:int [y>5]"の記述では、"[x>0]−>"および"[y>5]"がガードであり、値xが正数である時ゲートgに対してイベントが発行され、変数yが5より大きな整数とユニファイ可能な時に限りイベントが実行されることを示している。この時、値xの送信/変数yへの値の受信が同時に行われる。

(2) プロセスの記述

プロセス内のイベントの実行順序は、順次実行を規定するaction prefix';'、および選択実行を規定するchoiceオペレータ'[]'により記述される。a, b, cをイベント、Bをプロセスの生成とした時、"a; b; c; B [] b; a; c; B"という記述は、a, b, cの順にイベントを実行してプロセスBを生成するか、またはb, a, cの順にイベントを実行してプロセスBを生成するかのどちらかであることを意味している。

(3) 結合オペレータ

独立に定義されたプロセスは、結合オペレータにより関係を記述され、このプロセスの関係の記述もまた、プロセスの動作として記述される。結合オペレータとしては、並列実行、順次実行、中断実行がある。以下において、B_nはプロセスまたはイベントを、g_nはゲート識別子を意味する。

① 並列実行

並列実行オペレータにより複数の並列に動作するプロセスを生成することができる。生成された子プロセスは、親プロセスのゲートを引数として渡され、親プロセスが同期するプロセスおよび環境(外部プロセス)とこのゲートを介して同期する必要がある。

並列実行オペレータには、独立実行と依存実行の2つがあり、独立実行"$B_1 ||| B_2$"は、外部プロセスのイベントをB_1またはB_2のどちらかが受理可能である時実行し、互いには同期しないことを示す。依存実行"$B_1 || B_2$"は、外部プロセスとのイベントをB_1およびB_2の両プロセスが受理可能である時に限り、互いに同期して実行することを示す。

プロセスが複数のゲートを持ち、独立実行と依存実行が混在する時、依存実行をすべきゲート(g_1)だけを指定して"B_1[g_1, g_2] |[g_1]| B_2[g_1, g_2]"と記述する。

② 順次実行

順次実行"$B_1 >> B_2$"は、2つのプロセスが順に実行されることを示す。B_1が正常に終了したならば、B_2が実行される。

③ 中断実行

中断実行オペレータは、異常発生時等の通常の実行の中断を記述する。中断実行"$B_1 [> B_2$"は、B_1の実行中にB_2の中で最初に実行すべきイベントが受理可能になったならばB_1の実行を中断してB_2が実行されることを示す。B_1の最初のイベントが実行される前に、B_2の最初のイベントが実行可能になった時は、B_2のみが実行される。B_1が正常に終了したならば、B_2が実行されることはない。

2.2 プロセス間通信の特徴

プロセス間の同期通信では、生成する子プロセスに依存実行のためのゲートを渡すことにより、そのゲートで同期しなければならないプロセスの数が動的に増加する。また同期時に、送信/受信が同時に行われる。このような同期通信は、多くのOSやプログラミング言語の提供する送信/受信が明確な1対1のプロセス間の同期通信とは異なる。さらに、仕様の実行過程における動的な子プロセスの生成/消滅により、イベントが実行される時に同期するプロセス数および相手は刻々と変化していく。

```
process A[g] := B[g] || C[g]
where
   process B[g] := B1[g] ||| B2[g] endproc
   process C[g] := C1[g] || C2[g] endproc
endproc
```

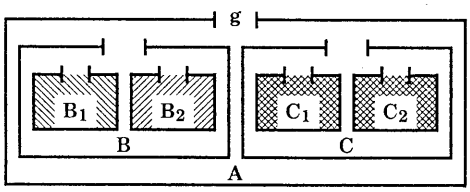

図3　複数の同期相手

例えば、図3のLOTOS仕様の実行過程において、プロセスC_1およびC_2は、外部のプロセスとともにプロセスB_1またはB_2のいずれかと同期する必要がある。それぞれのプロセスがさらに子プロセスの生成を繰

り返すことにより、イベントが実行されるために同期すべきプロセスの数および相手が変化していく。

3. LOTOS実行系

本実行系は、LOTOS仕様をC言語のプログラムに変換するトランスレータと、変換したプログラム内のLOTOSプロセスを並列に実行するスケジューラから構成される。

3.1 基本検討

本実行系は、LOTOSによる仕様をインタプリタ形式でなく、コンパイル形式で高速に実行することを目的としており、以下の検討を行った。

(1) 並列プロセスの実現法

LOTOSによる仕様は並列に動作するプロセスの集合として記述されるため、並列プロセスをいかに実現するかが重要な問題となる。並列プロセスを実現する方法として以下の3つが考えられる。

① LOTOSプロセスを、OSの管理するプロセスに対応させて実現する。

② LOTOSプロセスを、並列言語の提供する並列動作の単位(Adaのタスク等)に対応させて実現する。

③ OSや並列プログラミング言語にたよることなく、独自の並列プロセスの実行環境を構築する。

①および②の実現法では、プロセス管理/メモリ管理等をOSまたは言語の機能にまかせるため並列プロセスの実現は容易であるが、以下の問題がある。

LOTOSでは同期時にプロセス間でイベントの値をユニファイするため、計算機上で実現された全てのプロセスが同一のアドレス空間を持つことが望ましい。①では、このユニフィケーションの実現が困難になる。さらに、LOTOSでは、一般的に通信システムの仕様中に数十個のプロセスが定義され、同一プロセスが複数個生成されることも考えると、実行中に生成されるプロセス数はかなり多くなる。またLOTOSプロセスの動作が、他のプロセスとの同期通信であるイベントの授受からなることを考えると、イベントの実行によるプロセス切り替えが頻繁に発生することになる。従って、通信の処理時間、プロセス切り替えの時間が全体の処理速度に大きく影響すると考えられ、高速な処理が期待できない。

②の実現法では、LOTOSのプロセス間通信を言語の提供するプロセス間通信に対応させてどのように実現するかが問題となる。提供されるプロセス間通信は基本的に、1対1で情報の送受信を特定した通信機構であり、2個以上複数のプロセス間で送受信が同時に行われるLOTOSのプロセス間通信を実現するためには、同期相手およびイベントによる同期条件を集中して管理する機構が必要となり、その実現法が複雑になる。

これに対して、③の方法では、計算機上の1プロセス内でLOTOSプロセス制御機構を構築できるため、高速なプロセス切り替え、プロセス間通信をLOTOSの動作に合わせて実現することが可能である。そこで、③の方法により処理系を作成することにし、プロセス管理/メモリ管理/LOTOSのプロセス間通信を実現するスケジューラを作成した。

(2) ADTによるデータ型の定義

LOTOSによる仕様では、ADTを用いてデータ型(ソート)およびその操作を規定する。ADTを用いた記述は容易ではなく、理解性も乏しい。また操作の定義を変換したとしても生成された操作関数の処理速度は期待できない。データ型およびその操作関数は、通信システムを実装するプログラミング言語により直接コーディングしたほうが、高い処理速度が期待できるため、本処理系では、データ型はプログラミング言語のものを使用し、その操作関数はその言語により直接コーディングすることとする。

3.2 構成

本実行系は、LOTOSによる仕様をプログラムに変換するトランスレータと、LOTOSプロセスを実現するスケジューラからなる(図4)。

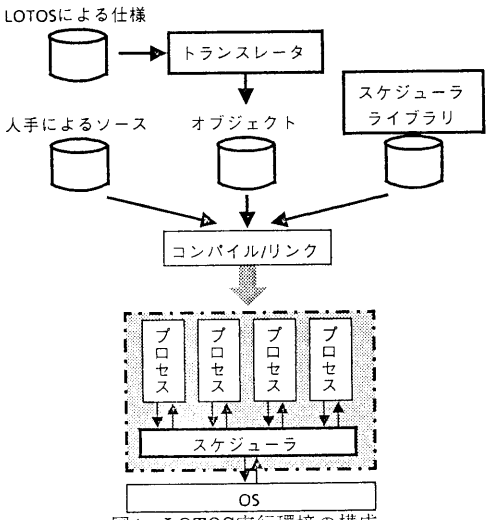

図4　LOTOS実行環境の構成

スケジューラは、ライブラリ(スケジューラ関数群)

として実現され、プロセスの生成/消滅、メモリ管理等のプロセス実行･同期管理、およびイベントを用いた通信管理機能を提供する。また、本実行系では、以下の手順によりLOTOSによる仕様を実行する。

① トランスレータは、LOTOSによる仕様をCのプログラムに変換する。プロセスの同期･通信はスケジューラの提供する関数の呼び出しに変換する。

② ADTで記述されるべきデータ型/操作関数の定義、LOTOSの仕様の範囲として扱われない実装に依存した動作は、プログラミング言語により直接記述する。

③ トランスレータのオブジェクト、人手により記述されたプログラムにスケジューラライブラリをリンクすることにより、スケジューラが組み込まれる。

LOTOS仕様から生成されたプログラムはスケジューラライブラリとともに、OS上では一つのユーザプロセスとして実現される。スケジューラは個々のLOTOSプロセスをライトウエイトプロセスとして扱い、そのスタックエリア等を管理する必要があるため、アドレス操作が容易なC言語を用いて実現することにした。

4. スケジューラ

4.1 機能

スケジューラは、LOTOSプロセスをライトウエイトプロセスとして実行するライブラリとして実現され、以下の機能を提供する。

(1) 並列に実行されるプロセスを実行する機能。具体的には、プロセスの生成/消滅にともなう資源の管理、および実行の管理を行う。

(2) 同期通信を実現する機能。相手プロセスを指定することなく複数プロセス間で行われるLOTOSの同期通信をプロセス間の通信機能として提供する。

スケジューラは、上記の機能を実現するため、図5に示すアルゴリズムに従って処理を行う。スケジューラは、同期待ちキューによりイベントを発行して同期を待つプロセスを管理し、実行待ちキューにより同期が成立して実行可能となったプロセスを管理する。スケジューラは、プロセスがイベントを発行することにより起動され、同期処理を実行して次のプロセスを選択する。

同期処理は、同期テストと待ちキュー処理からなる。同期テストは、同期可能性の判定、同期対象の探索、マッチングテスト、ガードテストの4つの処理からなる。同期待ちキューは、イベントを発行して同期を待っているプロセス間の関係を保持しており、スケジューラはこのキューを用いて以下の手順に従って、同期処理を行う。

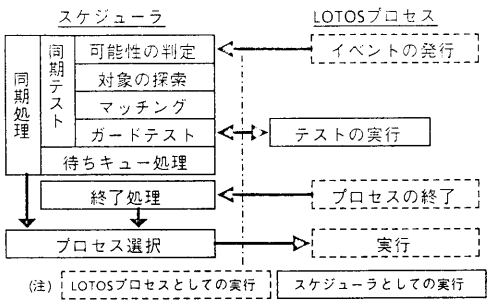

図5　スケジューラのアルゴリズム

まず最初に発行されたイベントの同期可能性を判定し、可能性がある場合は、同期待ちキューを探索して同期する相手(同期対象)を見つける。次に、探索された同期対象とイベントの値のユニフィケーションを実行するマッチングテストを行う。この同期対象の探索、マッチングテストを繰り返して、同期すべきすべての同期対象との間でユニフィケーションが成功したならば、イベントに付加されたガードのテストを行う。ガードテストまでの全ての同期テストが成功した時同期が成立し、同期が成立したすべてのプロセスを実行待ちキューにつなぎかえる待ちキュー処理を実行する。

以下、4.2節で並列プロセスの実現法を述べ、4.3節ではプロセス間通信の実現法を述べる。プロセス間通信を実現するには、同期の関係を保持する同期待ちキューおよびユニフィケーションの実現が重要である。4.3.1節で、同期対象を管理するための同期待ちキューの構成を示し、同期可能性の判定と同期対象の探索について述べる。4.3.2節では探索された相手とのユニフィケーションの実現法を示し、マッチングテスト、ガードテストについて述べる。また、4.4節では、LOTOSの機能の中で他プロセスとの相互関係があるために、スケジューラにより実現する必要のあるものついて述べる。

4.2 並列プロセスの実現

スケジューラは、並列に動作するLOTOSプロセスを、以下のように実現する。

(1) ライトウエイトプロセスによる実現

LOTOS仕様に現れるプロセスは、トランスレータによりCの関数に変換される。この関数をライトウエイトプロセスとして動作させるために、スケジュー

ラはスタックポインタ/プログラムカウンタ等の動作情報と動作するために使用する固有のスタックエリアを管理する必要がある。

これを実現するためにOSが本実行系に対して割当てるスタックエリアを、スケジューラが一括して管理し、スタックエリアを分割してそれぞれのプロセスに割当てる。プロセスは割当てられたスタックエリア内で動作情報を用いて実行する。プロセスがイベントを発行することにより起動されたスケジューラは、プロセスの動作情報とイベントの同期要求を同期待ちキューにつないで管理する。同期が成立して実行可能となったプロセスの動作情報は実行待ちキューにつなぎ替えられる。実行待ちキューにつながれている動作情報を用いてコルーチンコールすることにより、次のプロセスが実行される。(図6)

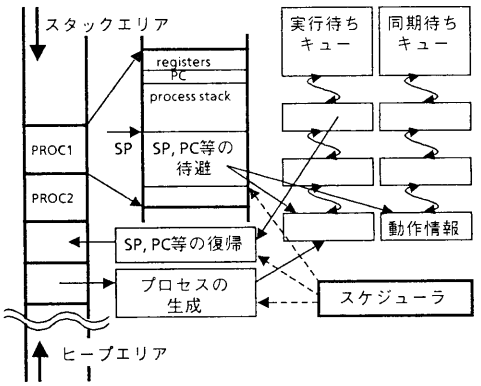

図6　並列プロセス管理

(2)スタックエリアの管理

プロセスに割当てられたスタックエリアは、プロセスが終了すると解放されてしまう。これに対して、プロセス間で同期が成立すると変数と変数、または変数と値がバインドされ複数のプロセスにより一つのデータが共有されるので、これらの値や変数をデスクリプタにより表現し、共有のヒープエリアにデータを格納する。このため各プロセスのスタックエリアは、データの値を管理するデスクリプタへのポインタとプロセスの制御情報を保持するだけで良く、小さくすることができる。そこで、スタックエリアを固定長で割り当てることにより、生成･消滅が頻繁に発生するLOTOSプロセスのスタックエリアの確保/解放を、簡単に効率良く実現することができる。

(3) プロセスの生成

プロセスが生成されると、スタックエリア上に生成されたプロセスが動作するための情報(関数構造)を設定する必要がある。

スケジューラは、レジスタ/プログラムカウンタ/関数の引数を設定した関数構造を、使用されていないスタックエリア上に生成し、実行待ちキューに生成されたプロセスの動作情報をつなぐ。プロセスが選択されると、この動作情報を用いてコルーチンコールすることにより設定されたスタックエリア上でプロセスが実行される。

(4) プロセスの終了

プロセスの終了が実行されると、プロセスの発行したイベントが残っているならばこれを同期待ちキューから取り除き、プロセスの保持するデスクリプタおよびスタックエリアを解放する。

4.3 プロセス間通信の実現

プロセス間通信が実行されるためには、それぞれのプロセスが発行したイベントの対応する値宣言または変数宣言の間で同期条件が成立する必要がある。本節ではプロセス間通信を提供するために、同期対象の管理の実現、同期条件のテストおよび対応する後処理を行うユニフィケーションの実現を述べる。

4.3.1 同期対象の管理

(1) 同期待ちキューの構成

2種類の並列実行オペレータを用いたプロセスの生成により、同期すべき対象プロセスおよび対象プロセスの数が動的に変化する。これに対してイベントの発行では、同期する相手を指定しないため、同期対象を探索する必要がある。同期すべきプロセスの関係を管理するために、同期待ちキューは、宣言されたゲート毎に図7に示すようなプロセスの親子関係を表現する木構造を持つ。

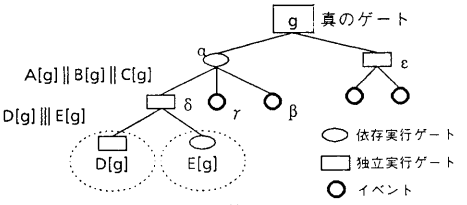

図7　同期待ちキュー

並列実行オペレータを用いて新しいプロセス(子プロセス)を生成する時、オペレータに対応する見かけのゲートを生成する。子プロセスの発行するイベントは、この見かけのゲートにつながれる。見かけのゲートでは依存実行と独立実行を区別して同期関係を表現する。図7では3つのプロセスA,B,Cが依存実行により生成され、さらにプロセスAがプロセスD,Eを独

立実行により生成したことを示している。

(2) 同期可能性の判定

イベントを発行するプロセスは、ゲート識別子とデスクリプタへのポインタを引数としてスケジューラを呼び出して同期待ちの状態になる。イベント発行により起動されたスケジューラは、同期の成立を確認する同期テストを実行する。同期テストでは、発行されたイベントにより同期が成立する可能性を最初に調べる。同期可能性は、同期待ちキューを用いて次のように判定する。

同期待ちキューにおいて、依存実行の見かけのゲートを生成した時点では親のゲートとつながず、そのゲートにつながるイベントが全て発行されてから親ゲートにつなぐことにする。これにより、あるイベントを発行した時、発行されたイベントから真のゲートまでたどれる場合のみ、同期が成立する可能性があると判定できる。

(3) 同期対象の探索

同期が成立する可能性がある場合には、同期対象の探索を行う。

同期対象の探索では、発行されたイベントのつながれるゲート(見かけのゲート)を出発点として、同期待ちキューを探索する。依存実行の見かけのゲートにつながるすべてのイベントは同期対象であり、独立実行の見かけのゲートにつながるイベントの中の一つが同期対象である。図7中のイベントβにより同期が成立するためには、依存実行ゲートαにつながるイベントβ,γ、独立実行ゲートδを介したイベント、およびゲートgを介したゲートεにつながるイベントが全て発行されており、さらに全てのイベントがユニファイ可能である必要がある。

4.3.2 ユニフィケーションの実現

同期すべき相手が決定すると、それぞれのプロセスが発行したイベントの対応する値宣言または変数宣言の間で、ソートの一致および値の一致を確認し、値未定の変数に値をバインドする。このユニフィケーションを実現するために、値のソートを表現するデスクリプタを導入した。

(1) デスクリプタの導入

同期条件をテストするには、ソートの一致を調べる必要がある。C言語のデータはソートを持たないので、データのソートを示すためのデスクリプタを導入する。ユニフィケーションを効率良く実行するために、データの値を保持する値デスクリプタと、値デスクリプタの格納アドレスとソートを保持する参照デスクリプタを用意した(図8)。本実行系で扱われる値はすべて値デスクリプタを用いて表され、値の確定した変数または定数は参照デスクリプタを介してその値を保持する。値が未定の変数は、参照デスクリプタが値未定であることを表現する。これにより値未定の変数のために、値により大きさの異なる値デスクリプタの領域を確保しておく必要がなくなる。

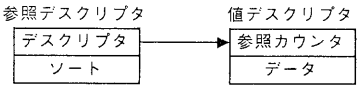

図8 デスクリプタ

また、データ領域に参照数を示すカウンタを設定し、値が0となった時に解放することにより記憶領域の確保/解放の管理を実現している。

(2) ユニフィケーションの実行(マッチングテスト)

スケジューラは、4.3.1節で述べたように、同期待ちキューをたどり同期対象を探索しながら、発行されたイベントと同期対象として選択されたイベントが同期条件を満たすかどうかをテストする。データのソートの一致を確認した後で、デスクリプタをたどって値のマッチングを確認し値をバインドする。

イベントの値宣言および変数宣言のユニフィケーションは、デスクリプタを用いて次のように実行される(図9)。以下の処理は、2つのプロセス間のユニフィケーションを示すが、これらを複数回適用することにより複数プロセス間のユニフィケーションを実現する。

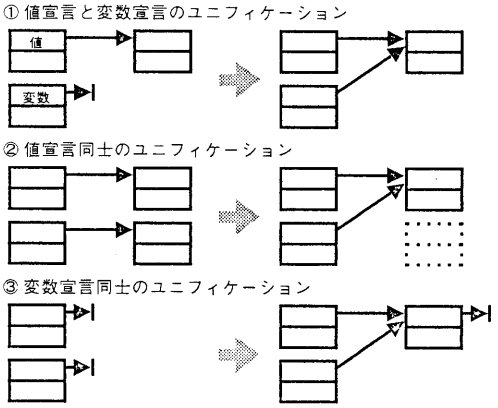

図9 デスクリプタとユニフィケーション

① 値宣言と変数宣言のユニフィケーション

ソートの一致が確認されたならば、変数宣言の参照デスクリプタが値デスクリプタを参照する。

② 値宣言同士のユニフィケーション

ソートの一致を確認し、さらに参照デスクリプタをたどって値の一致を確認する。双方の一致が確認されたならば、一方の値デスクリプタは解放され他方の値デスクリプタを参照する。

③ 変数宣言同士のユニフィケーション

ソートの一致が確認されたならば、新しい参照デスクリプタを生成し、両者がこれを参照する。

(3) ガードテスト

ガードを用いることにより、イベント実行時における値宣言または変数宣言に制約を付けることができる。このためユニフィケーションの実行において、イベントに付けられた制約であるガードを判定する必要がある。値宣言に対するガードは、イベントを発行する時点で判断することができるため、トランスレータはガードを条件文に変換し、条件が成立した時にのみイベントを発行する。変数宣言に対するガードは、その判定時に値が決定している必要があるため、マッチングテストの終了後にスケジューラが実行しなければならない。

変数宣言に対するガードの実現では、個々のプロセスにより異なるガード条件の判定を実現する必要がある。本処理系はC言語により実装しているため、スケジューラにガード情報を渡すことが困難であった。このためプロセスの一部にガードの条件を展開しておき、ガードテスト時にコルーチンとして一時的にプロセスに制御を戻して、テスト結果をスケジューラに返している。

プロセスは、ガード付きのイベントであることをイベントの情報に付け加えて発行する。スケジューラは、ユニフィケーションが成立した場合、マッチングテスト中に抽出しておいたガード付きプロセスに、順に制御を渡してガードテストを行う。

(4) バックトラック

マッチングの実行中にテストが失敗した時、またはガードテストにおいて同期が成立しなかった時、バックトラックを行い、別の同期対象を探索して再度ユニフィケーションを行う。

4.4 LOTOSの機能の実現

ここでは、スケジューラの機能として実現する必要のあるものについて述べる。

(1) プロセスの中断実行管理

LOTOSの機能である中断実行は、他プロセスの実行を中断(消滅)するためスケジューラが実現する必要がある。この中断実行も並列実行オペレータと同様に、プロセスの生成/消滅を繰り返すことによりその関係が複雑となるためこれを管理する必要がある。

図10に示す仕様を考える。プロセスB_1が実行可能となった時はプロセスA(A_1, A_2)を中断させる必要があり、プロセスB_2が実行可能となった時はプロセスB_1およびプロセスA(A_1, A_2)を中断させる必要がある。またプロセスA_1およびA_2が正常に終了した時は、プロセスB(B_1, B_2, B_3)およびプロセスCを中断させる必要がある。

```
process X := A [>B [>C
where
    process A := A1 ||| A2 endproc
    process B := B1 [>(B2 ||| B3) endproc
    process C := ..... endproc
endproc
```

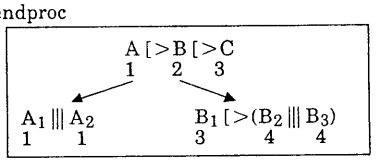

図10 中断実行

各プロセスに中断の関係を表現する整数値(中断レベル値)を持たせることで、このような中断実行の管理を行う。図10でプロセスBは中断レベル値2を持っており、プロセスB_1とB_2, B_3の並列動作を中断実行により生成する時、B_1には自分より1大きい中断レベル値3を、B_2, B_3には自分より2大きい中断レベル値4を付けていく。A_1, A_2のように中断実行を用いない場合は、自分と同じ中断レベル値1を付ける。

これによりプロセスが実行可能となった時、中断レベル値が親プロセスよりも大きいならば、中断関係があることが判定でき、中断レベル値の小さいプロセスの実行を中断(消滅)させる。プロセスが正常に終了した時、中断レベル値が1以上であるならば中断関係があり、中断レベル値の大きいプロセスの実行を中断させる。

(2) 選択実行

選択実行により、複数個のプロセスまたはイベントの実行を選択することができる。個々の動作をプロセスとして実現することもできるが、本処理系では処理効率を考慮し、その動作がイベントである場合には、一つのプロセスから複数個のイベントが発行できるようにして実現した。

例えば、"g_1 !x [] g_2 ?y:int [] B"の記述では、ゲートg_1およびg_2にイベントを発行し、プロセスBを生成

し、これらの中で一番最初に実行可能となった動作が選択される。この時、ゲートg_1に対するイベントが実行されたならば、ゲートg_2に対するイベント情報およびプロセスBの動作情報を削除する必要がある。プロセスBが動作可能になったならば、ゲートg_1およびg_2に対するイベント情報を削除する必要がある。このため選択実行を行ったプロセスの同期が成立した時、スケジューラは待ちキュー処理の一つとして成立しなかった選択動作の情報を削除する。

(3) プロセス終了時の同期処理

並列実行オペレータにより生成されたプロセスは同期して終了する必要があるため、プロセスの終了を実行するプロセスは、親プロセスに通知して終了の同期を待つ。終了の同期が成立するとプロセスの終了が実行され、プロセスの終了により発生するプロセスの中断実行管理を実行しプロセスを終了する。

5. トランスレータ

本章では、スケジューラの機能を使用するプログラムを生成するトランスレータについて述べる。

5.1 機能

トランスレータは、LOTOSによる仕様の内、プロセスを用いて記述されるシステムの動作仕様を入力としてC言語のプログラムを生成する。ADTによるデータ型の記述に対しては、仕様の中でデータ型のソート名およびデータ型に対する操作名のみを指定することとし、トランスレータはデータ型の一覧、および操作に対応するCの関数を呼び出すために必要なextern宣言をヘッダファイルとして生成する。このシステムの実装者は、ヘッダファイルに宣言されている関数をコーディングする必要がある。

以下では、トランスレータの実現に際し特に問題となったプロセスの変換を中心に述べる。

5.2 プロセスの変換

(1) 仕様に定義されるプロセス

トランスレータは、原則として仕様に記述されたプロセスをスケジューラが提供するプロセスに対応させて、Cの関数(プロセス関数)に変換する。またプロセス関数が動作するための情報としてプロセス構造体と、プロセス記述子を生成する。プロセス構造体は、関数アドレス/プロセス名/ローカル変数等の静的な情報を保持しており、プロセスの記述に対応して生成される。プロセス記述子は、発行したイベントおよび中断レベル値等の動作情報/親および子プロセスの関係等の動的な情報を格納するもので、仕様実行中のプロセス生成時に、スケジューラが親プロセス等の情報を設定したプロセス記述子を生成する。このため同一の仕様を持つプロセスを生成する時、プロセス記述子は生成されたプロセス毎に設定されるが、プロセス構造体は一つだけ存在し共通して参照される。

プロセスの内部だけで使用されるローカル変数は、メモリ管理の関係から一ヶ所にまとめて管理する必要がある。このためトランスレータは、プロセスの仕様で使われているローカル変数をプロセス構造体に登録する。スケジューラは、プロセスの生成時にローカル変数をまとめて確保し、プロセスの終了時にこれらのローカル変数を解放する。

(2) 仕様に定義されないプロセス

トランスレータは、仕様に記述されたプロセスをプロセス関数に変換するだけでなく、定義された動作からプロセス関数を生成する場合がある。

例えば、"g_1 ! x ||| g_2 ? y:int"の記述が可能であるが、これは2つの異なる並列動作を示しており一つのプロセスでは実行できない。このためトランスレータは、並列オペレータの左辺および右辺の動作をプロセス(ダミープロセス)として生成する。

この他にも、オペレータの組合せにより同様なダミープロセスを生成する場合がある。ただし、選択実行によりイベントを選択する動作は、仕様中で多く使用されるため処理の速度を考慮し(3)に示すような特別な扱いをする。

(3) イベントの変換

スケジューラでは4.4(2)に示したように、一つのプロセスから複数個のイベントが発行できる選択実行を実現している。トランスレータは、イベントの記述"g ?x:int [x>0]"を以下のように変換する。

先ずイベントの発行のためにスケジューラを起動する関数を生成する。この関数は、発行されたイベントにより同期が成立したか(match)、同期が成立しないため次のイベントの発行を要求するか(next)を戻り値として返す。もし発行されたイベントにガードが付けられているならば、ガードテストを要求し、テスト結果を返すスケジューラの番地を戻り値として返す(図11)。トランスレータは、この戻り値に対応した構造の条件分岐を生成する。matchによる分岐には、イベントが成立することにより実行される処理を展開し、nextによる分岐には、次のイベントに対する同様な構造を展開する。ガードテストによる分岐に

は、対応するテストを展開し、戻り値をもとにスケジューラをコルーチンコールしてテスト結果を返す処理を生成する。選択実行を伴わないイベントの記述は、nextによる分岐を持たない構造に変換される。

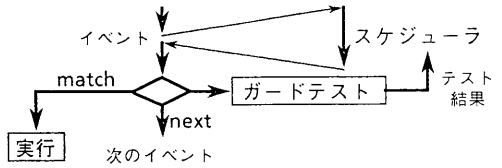

図11　イベントの選択実行の処理の流れ

5.3 その他の変換

並列実行オペレータで結合されたプロセスの生成は、オペレータに対応する見かけのゲートを生成し、このゲートを各プロセスに渡して生成するように変換する。

中断実行オペレータで結合されたプロセスの生成は、4.4で述べた中断レベル値をプロセス記述子に設定して生成するように変換する。

順次実行オペレータで結合されたプロセスの生成は、実行すべき順にプロセスを生成していくが、各プロセスの実行の終了を待ち合わせて次のプロセスを生成していく構造に変換する。

5.4 トランスレータによる最適化

効率の良いプログラムを生成するために、トランスレータは以下の機能を持つことが考えられる。

(1) 選言的に記述された複数のイベントの内、ガード以外の同期条件が等しいものを一つのイベントにまとめる。

(2) 再帰的または逐次的に実行されるプロセス群を一つのプロセスにまとめる。

これらの具体的な実現法については、現在検討中である。

6. 考察

トランスレータを用いたLOTOS実行系の作成において、LOTOS特有のプロセス動作の実現法が重要であり、特に相手が動的に変化し複数のプロセス間で実行されるプロセス間通信に問題があることを指摘した。これをスケジューラがプロセスの動作を管理する同期待ちキューに同期の関係を表現する構造を持たせることで実現する方法を示した。またデスクリプタを用いて値を管理することにより、メモリエリアの効率的な利用を実現するとともに、同期の実行としてのユニフィケーションを実現する方法を示した。

LOTOSによる仕様に依存しない機能をスケジューラの機能として実現することにより、トランスレータを比較的に簡単な構造にすることができた。ADTの難解さおよびその変換処理の複雑さからADTを扱わなかったが、その代わりとして本処理系に合わせた専用のデータ型記述の方法を検討したほうが有効であると考えられる。

本実行系は、ほとんどの機能がC言語により記述され、汎用性は高いと考えられる。また並列プロセスを実現するスケジューラとCのプログラムを生成するトランスレータに分割して作成したことにより、異なるOSに対してスケジューラの変更のみで対応できるため高い移植性があると考えられる。

7. おわりに

現在、並列プロセスを実現するスケジューラ、およびCのプログラムを生成するトランスレータを本稿で述べた検討に基づいて作成している。またトランスレータについて、さらに効率の高いプログラムの生成法を検討中である。最後に日頃御指導頂くKDD上福岡研究所小野所長、安藤次長、通信ソフトウェア研究室小西室長、若原主任研究員、コンピュータ通信研究室加藤主査に感謝する。

参考文献

[1]: ISO 8807, "LOTOS - A formal description technique based on the temporal ordering of observational behaviour", Feb. 1989.

[2]: T.Bolognesi, E.Brinksma, "Introduction to the ISO Specification Language LOTOS", Computer Networks and ISDN Systems, 14(1987), pp.25-59

[3]: ISO/DTR 9571, "LOTOS Description of the Session Service", Feb. 1988.

[4]: ISO/DTR 9572, "LOTOS Description of the Session Protocol", Feb. 1988.

[5]: 大薗, 二木, "Specifications of Library and Lift Problems in LOTOS", 情報処理学会ソフトウェア工学研究会, 64-12, Feb. 1989.

[6]: J.P.Briand, M.C.Fehri, L.Logrippo, A.Obaid, "Executing LOTOS Specifications", Proceedings of IFIP Workshop 'Protocol specification, testing and verification VI', pp.73-84, North-Holland, Amsterdam, 1987.

[7]: 野村, 長谷川, 瀧塚, "LOTOSの同期通信機構を持つ並列処理環境の構築法", 情処全大, 5E-2, Sep. 1988.

[8]: 野村, 長谷川, 瀧塚, "LOTOSプロセスの並列処理環境における同期処理機構", 情処全大, 4P-7, Mar. 1989.

[9]: R.Milner, "A Calculus of Communicating Systems", Lecture Notes in Computer Science, Vol.92, Springer-Verlag, 1980.

[10]:C.A.R.Hoare, "Communicating Sequential Processes", Prentice-Hall Intl.,1985.

[11]:H.Ehrig, B.Mahr, "Fundamentals of Algebraic Specification 1", Springer-Verlag, Berlin, 1985.