LOTOSの同期通信機構を持つ並列処理環境の構築法

5E-2

野村真吾　　長谷川亨　　瀧塚孝志
国際電信電話株式会社　上福岡研究所

1. はじめに

筆者らは通信システムの開発コスト削減のために、プロトコル仕様を形式的に記述する形式記述技法(FDT)であるEstelle, SDLで記述した仕様をAdaの通信プログラムに変換するトランスレータを試作してきた[1,2]。現在、ISOではLOTOS[3]によりプロトコル仕様を記述する作業も行われており、同様な処理系が必要となっている。これまでのトランスレータは、FDTの言語仕様を既成のプログラミング言語の持つ機能に対応させたが、並列に動作するLOTOSのプロセスの制御法、プロセス間の同期通信機構等は複雑であり、既成の言語の機能による実現は困難である。そこで、LOTOS実行環境を実現するために、プロセスの実行･同期通信機構を効率よく実現した並列プロセス処理環境の作成と、その環境上で動作するプログラムを生成するトランスレータの作成を検討している。本稿では、並列プロセス処理環境の実現法について述べる。

2. LOTOSの概要

LOTOSは、外部プロセスとの入出力応答であるイベントの列挙によりプロセスを記述し、並列オペレータ等の結合オペレータを用いてプロセスの関係を記述する。イベントは、ゲートを介して複数プロセス間で交換する。例えばg !x ?y:t の記述は、ゲートgに対して値xを宣言し、値の集合(ソート)がtである変数yを宣言するイベントの発行を示す。表1にイベントの同期条件を示すが、イベントの同期が成立するためには、少なくともソートが一致することが必要である。

プロセスの結合オペレータには以下のものがある。2つのプロセスをB1,B2とすると、独立実行B1 ||| B2は、外部プロセスのイベントをどちらかのプロセスが実行し、完全同期実行B1 || B2は、外部プロセスのイベントを両プロセスが実行可能であるとき同期して実行することを示す。また、順次実行B1 >> B2は、B1が正常に終了したならばB2が実行され、中断実行B1 [> B2 は、B1の実行中にB2の初期イベントが実行可能になったならばB1の実行を中断してB2が実行されることを示す。

3. LOTOS実行環境

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

表1　LOTOSにおける同期条件と後処理

3.1 言語選択

Ada等の並列プログラミング言語では、情報の送り側と受け側がはっきりした1対1の同期通信機構を提供している。一方LOTOSでは、複数のプロセスが一つのゲートを共有し情報の送受を同時に行うことができる。こうした機構をAdaのランデヴー等に対応させると、生成されるコードが複雑となるばかりでなく高いスループットが期待できない。そこでLOTOSの同期通信機構を持つ並列プロセス処理環境を、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。

3.2.LOTOS 実行環境の構成

LOTOS実行環境の構成を、図1に示す。

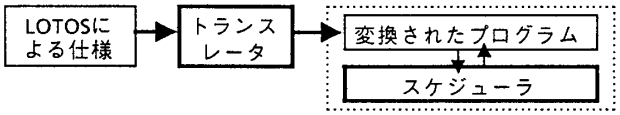

図1　LOTOS実行環境の構成

(1)LOTOSプロセスの実行･同期管理、およびイベントの通信管理機能をスケジューラとして実現する。スケジューラはC言語で記述するが、C言語の機能では実現できないプロセス生成時のスタックポインタの移し換えはアセンブラにより記述する。

(2)トランスレータを用いて、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関数呼び出しに変換する。

4. 並列プロセス処理環境

並列プロセスの管理および同期通信を実現するスケジューラに関して以下の検討を行った。

4.1. 並列プロセスの管理機構

(1)プロセス切り換え

実行可能なプロセスや生成されたプロセスは実行

Parallel Processing System for Specifications written in LOTOS
Shingo NOMURA, Toru HASEGAWA, Takashi TAKIZUKA
KDD Kamifukuoka R & D Laboratories

待ちキューにつながれ、イベントを発行したプロセスは同期待ちキューにつながれる。実行中のプロセスがイベントを発行してスケジューラを呼び出すことにより、プロセスの切り換えが行われる。スケジューラは、実行中のプロセスの状態を退避して同期待ちキューにつなぎ、4.2節の同期処理を実行した後、実行待ちキューの先頭のプロセスの状態を復帰する。

(2)スタックエリアの管理

LOTOSでは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参照されるので、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、プロセスの制御情報･デスクリプタへのポインタ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)プロセス終了の同期

完全同期実行オペレータで結合されたすべてのプロセスは同期して終了する必要があり、また順次実行オペレータで結合されたプロセスは先に実行しているプロセスの終了を待つ必要がある。これらの同期は、イベントによる同期待ちキューと同様な終了待ちキューを生成し、終了に対応するイベントを用いて実現する。

4.2. 同期通信機構

(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ため、データのソートと格納アドレスを示すデスクリプタを導入する。イベントを発行するプロセスはデスクリプタを生成し、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す。スケジューラはこのポインタをたどることにより、対応するゲートの同期待ちキューにつながれた他イベントと、ソートおよび値の一致をテストする。同期条件が満たされた時は、データを共有するようにポインタをつなぎ換え、そのイベントを発行したプロセスを実行待ちキューにつなぐ。同期条件が満たされない時は、デスクリプタへのポインタを同期待ちキューにつなぐ。(図2)

デスクリプタの導入により変数の場合は、データ領域を確保する必要がなくなる。また、データ領域に参照プロセス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/解放の管理を容易に実現することができる。

(2)複数並列プロセス間の同期通信の実現

“プロセス$_1$[g]‖プロセス$_2$[g]”の完全同期実行では、ゲートgを介した外部プロセスのイベントは、

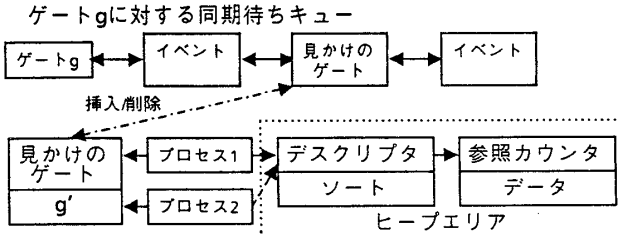

図2　同期管理

プロセス$_1$とプロセス$_2$が同時に実行可能である時に同期が成立する。この複数プロセスによる同期を管理するために、見かけのゲートを導入する。プロセス$_1$またはプロセス$_2$は、イベントを発行し同期待ちになると見かけのゲートg'につながれ、見かけのゲートg'は両プロセスが同期待ちになるのを待って、真のゲートgにつながれる。(図2)

(3)ガード付きイベント

[x>0]->g!x?y:int[y>1]の記述により、各イベント実行時の条件(ガード)を付けることができる。値宣言に対するガードは、イベントを発行する前に判断することができるため、トランスレータが条件文に変換する。変数宣言に対するガードは、実行時においてスケジューラが、ガード付きのイベントを発行したプロセスに一時制御を移してガードのテストを実行することにより実現する。

5. トランスレータ作成時の問題点

今後、この実行環境で動作するプログラムを生成するトランスレータを作成する予定である。効率の良いプログラムを生成するためには、以下の検討が必要である。

(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手法。

(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手法。

(3) 抽象データ型によるデータ記述の取扱い。

6. おわりに

LOTOSの同期通信機構を持つ並列プロセス処理環境について検討し、仕様の実行環境を構築する見通しを得た。本並列プロセス処理環境は、ほとんどの機能がC言語により記述することができ、アセンブラによる記述が必要なプロセス生成機能は20ステップ程度であり汎用性は高いと考えられる。最後に日頃御指導頂くKDD上福岡研究所小野所長、湯口次長、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鈴木主任研究員に感謝する。

参考文献

[1]: 加藤,長谷川,堀内,鈴木,“EstelleからAdaへのトランスレータの作成”,信学技報IN86-97,Nov. 1986.

[2]: 堀内,長谷川,加藤,“SDLからAdaへのトランスレータの設計”,信学総全大,1875, Mar. 1987.

[3]: 1987.ISO/DP 8807,Mar. 1987.