CSPプログラムのデバッギングに対する実行時支援

4X-2

秋山 功, 山田 剛, 小原 啓義
早稲田大学理工学部 電子通信学科

1. はじめに

並行プログラミングに関する研究は以前から進められ，並行プログラム記述用言語として様々なものが提案され実現されてきた．しかし，並行プログラムは，プロセス間相互作用の時間依存性・非決定性という逐次プログラムとは本質的に異なる性質を有する．プログラムの動作は各プロセスの実行速度に左右され得るから，プログラマはこの性質に留意しつつプログラミング・デバッギングを行わなければならず，並行ソフトウェアの開発は逐次言語に比して困難である．

筆者らは，並行ソフトウェアの生産性を向上させるには充実した開発環境が必要であると考え，並行ソフトウェア開発支援環境に関する研究を行ってきた [1, 2]．本稿では，プロセス間相互作用に関する論理的な時間順序関係を定義し，これに基づいて我々が製作するデバッガがプログラマをいかに支援するか，その方式について報告する．

2. 並行プログラムのモデル

我々がデバッグの対象とするプログラムはCSP [3] に基づいたものであり，文献 [3] に提案されている機能に拡張・制限を行ってモデルを設定する．以下に，文献 [3] と我々のプログラムモデルの相違点を示す．

(a) 同一プロセス内での並行実行は認めない．
(b) 送受信双方の命令に対して非決定的待ちを許す．
(c) 送受信命令はチャネルと呼ばれる仮想的な通信路に対して発行され，通信はこのチャネルを介して行われる．

さらに次のことを仮定する．

(d) プロセス・チャネルの動的な生成・消滅は行われない．チャネルは2つのプロセス間に固定されて存在する．

3. デバッグ事項

我々が設定したモデルにおいてはプロセス間相互作用はメッセージ通信のみに限定されている．今後，メッセージの送受信をイベントと呼び，プロセスxのi番目のイベントをEx_iで表すことにする．

我々の製作するデバッガは，プログラマに対して以下の機能を提供する．

3．1 イベントトレース

デバッガは各プロセスで生起するイベントの履歴を収集する．履歴は，プロセス間相互で行われるメッセージ通信が正しい内容を持ち，かつ，正しい同期が実現されているか否かの検証に用いられる．また，履歴はプロセス単体の振舞のデバッグにも利用される．

3．2 チャネルの状態表示

各チャネルの現時点での状態，及び，前回の観測からの状態遷移を，プログラマの要求に応じて表示する．但し，メッセージ通信は論理的に不可分であると考え，チャネルは以下の状態を取り得るものとする．

(a) 空 (b) 送信待ち (c) 受信待ち (d) 送受信可

3．3 非決定的通信に対する候補集合の検出

プログラムは非決定的動作を行うため，その正しさを保証するには非決定的な選択枝の全ての枝の動作を保証する必要がある．これを支援するためにデバッガは，非決定的待ちに対して選択可能な送受信命令の集合，すなわち候補集合をリストアップする．

3．4 イベント間の実行順序の検証

任意の2つのイベント間の生起順序が偶然によるものか，同期機構により制御されたものかを判断する．

3．5 デッドロックの検出

プロセス間にデッドロックが発生したことを検出し，プログラマに通知する．

4. デバッグ方式

4．1 時間順序関係の設定

Ex_i とEy_jが1つの通信に対応する場合，

$$Ex_i \nRightarrow Ey_j \quad \text{かつ} \quad Ey_j \nRightarrow Ex_i$$

であるとする．また，任意の x,y,i,jに対して$Ex_i \nRightarrow Ey_j$ かつ $Ey_j \nRightarrow Ex_i$ であり，そしてこの時に限り同時関係＝が成立する，すなわち

$$Ex_i = Ey_j \quad (\text{または} \quad Ey_j = Ex_i)$$

であるとする．

$Ex_i \Rightarrow Ey_j$ は「$Ex_i \nRightarrow Ey_j$ かつ $Ex_i \neq Ey_j$」と同義であり，同一プロセス内のイベントEx_i, Ex_jに関し

$$Ex_i \nRightarrow Ex_i$$
$$Ex_i \Rightarrow Ex_j \quad (i < j)$$

On-the-fly Support for CSP Program Debugging
Isao AKIYAMA, Tsuyoshi YAMADA, Hiroyoshi OHARA
WASEDA UNIVERSITY

が成立するものとする．

以上のように定義を行うと，関係$\precsim$は半順序関係となる．なお，$Ex_i \precsim Ey_j$ の時，「Ex_i はEy_j より前である」，「Ey_j はEx_i より後である」と言う．また，$Ex_i \Rightarrow Ey$の時は，「Ex_i はEy_j より真に前である」，「Ey_j はEx_i より真に後である」と言う．

$Ey_j \precsim Ex_i$ なる関係が推移律によって成立するようなEy_j から成る集合を $P(Ex_i)$ と表す．また，$Ex_i \precsim Ey_j$ なる関係が推移律によって成立するようなEy_j から成る集合を $F(Ex_i)$ と表す．これらの意味は，それぞれ「Ex_i と同時あるいはEx_i より前に生起するイベントの集合」，「Ex_i と同時あるいはEx_i より後に生起するイベントの集合」となる．

Fig.1 にイベントEr_i から見たイベントの生起に対する順序関係の例を示す．■と▼で表したイベントから成る集合が $P(Er_i)$ であり，■と▲で表したイベントから成る集合が $F(Er_i)$ である．イベントを節，関係$\precsim$を有向枝とするグラフを描くと，Er_i から到達可能なイベントがEr_i より後のイベントである．また，そのグラフの有向枝の向きを全て反転させると，Er_i から到達可能なイベントがEr_i より前のイベントである．

4．2　順序関係の解釈

イベントEx_i，Ey_j に着目した場合，Ex_i が $P(Ey_j)$ あるいは $F(Ey_j)$ の何れかに属せば，Ex_i とEy_j の間には生起順序が存在する．逆に，Ex_i が $P(Ey_j)$ 及び $F(Ey_j)$ の何れにも属さなければ，これらのイベントの生起順序は不定である．

4．3　候補集合の検出に対しての順序関係の応用

ある非決定的送受信命令C_1の候補集合が送受信命令C_2を含むか否かを検証する場合を考える．C_1が非決定的な選択枝のうちの１つをすでに実行している場合はその選択を取り消し，C_2がC_1より真に前であるか真に後であるならば，順序関係の解釈から，C_1の候補集合が送受信命令C_2を含むことはない．従って，あるイベントに関する候補集合をリストアップする際に調査すべき送受信命令は，順序関係によって限定することができる．

4．4　順序関係に基づく支援

本来生起順序が不定であるはずの２つのイベントに関し順序関係が成立してしまった場合には，それらの間に不正なイベントによる同期が行われたことを意味している．逆に，本来順序関係が定まっているはずのイベント間でその関係が検証されなければ，プログラマは順序関係が成立するようにデバッグを行えばよい．

5．　おわりに

以上，我々が製作するデバッガのデバッグ方式について報告を行った．上に述べたように，プロセス間相互作用に対する論理的な時間順序関係は，デバッグに際し有用な情報を提供する．

今後は，プログラムモデルに検討を加え，本デバッガを実機上にインプリメントして，引き続き並行ソフトウェア開発支援環境に関する研究を進める予定である．

謝辞

本稿作成に協力していただいた小原研究室CCSG諸氏に感謝する．

参考文献

［1］山田　他,“オブジェクト　モジュール標準形式IEEE P-695の処理系と問題点”,情報処理学会マイクロ　コンピュータ研究会,MC34-1,Dec.,1984,pp.1-10.

［2］山田　他,“マルチプロセッサ　システムにおける並行言語デバッギング　ソフトウェア”,情報処理学会第29回全国大会論文予稿集,4R-13,Sep.,1984.

［3］C.A.R.Hoare,“Communicating Sequential Process”, Comm.ACM,Vol.21,No.8,Aug.,1978,pp.666-677.

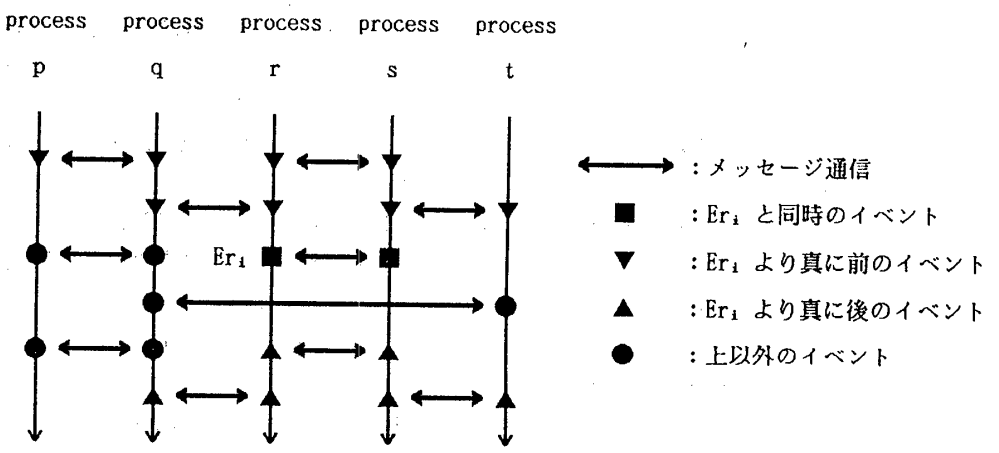

Fig.1　Er_i から見たイベントの生起に対する順序関係（例）