協調型言語による並列プロセスの時相論理に基づくモデル検証

大塚 寛†

愛媛大学理学部数理科学科†

1 概要

時相論理と有限状態機械(FSM)によるモデル検証は、仕様記述とその検証の基礎となる。ところで一般の並列プロセスをモデル化した複数個の FSM が通信しあいながら並列に動作する状況は、いわゆる state-explosion problem を引き起こす。そこで一般の並列プロセスではなく、制限された協調型言語を用いた並列プロセスのモデル検証、具体的には in,out,rd 操作を持つ各直列プロセスとタプル空間の間の通信を対象としたモデル検証について検討する。さらにこの手続きをなるべくタプル空間側の主導で行うような実験を試みたので、これについて報告する。

2 背景とタプル空間の概観

当該研究者はある実験的なシステムのソフトウェア共同開発者として、遠隔地からの実験、開発に参加している。しかし遠隔地であるゆえに、実験環境やソフトウェアの共同開発時に不便さが生じる。そこで実験の再現環境や共同開発者が担当するソフトウェアの仕様の API の再現環境としてタプル空間[1]の変形された一種を利用している。一方ソフトウェアの実装も特殊な並列言語ではなく、通常の手続き型言語にタプル空間へアクセスする基本的操作を加えた協調型言語[1]による開発を行っている。

協調型言語におけるタプル空間とは global buffer として働き、通信相手が特定される必要はないし、入出力時のデータの順序にも特に制限はない。しかしここでは、通常の相手がわかっておりデータの順序も保持されている通信を再現する環境として利用した。そこでまず簡単にタプル空間の側から見たシステムを概観する。

2.1 実験の再現環境として

実験で取り入れたデータ(をいくつか加工した結果)を取り入れ、保持し、タプル空間へのアクセスに従って、取り入れた時間間隔で送出する。タプル空間内部では、取り入れた時間間隔を超えてもアクセスがないデータは後述の可達性解析の II の処理時に利用している。

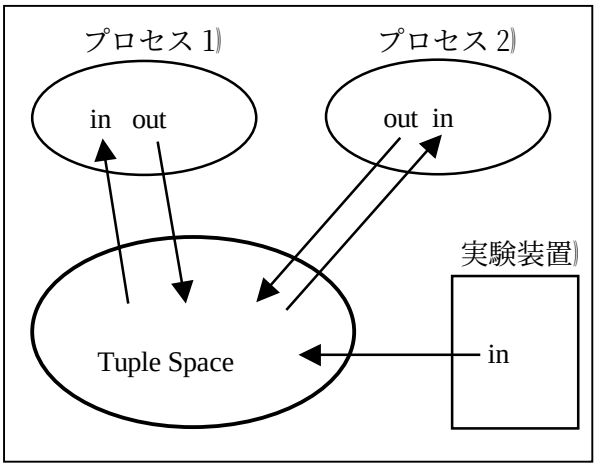

図 1

2.2 API の再現環境として

通信相手のプロセスが仕様どおりに実装されているものとして、API を期待通りに再現させる。我々の開発するシステムは機能ごとに分割され、それらを実装した複数プロセスを並列動作させる形を取っている。ここで各開発者(グループ)が担当するコンポーネントのソースを開発者全員で共有することはせず、各コンポーネント間の API にかかわる部分を正確に記述し、それに従ってプロセス間の通信を実装するようにしている。そこで他の共同開発者が担当するプロセスとの(通信データなども含めた)通信を仮想的にタプル空間との通信で代用している。この場合も 2.1 と同じく時間制約を設け、保持データを利用している。

なお以上の再現環境では、通信データなどの相手側のデータを取り寄せ、それらをタプル空間に設定するという手順を踏むことで、相手側プロセスとの仮想的な通信を再現している。

2.3 タプル空間のその他の機能

現在タプル空間内では呼出し手順の正しさなどをチェックしており、プロセスの単体テスト時に利用している。プロセスは既知の数値計算アルゴリズムを実装した直列プロセスで、タプル空間へアクセスする基本的操作は同期型を利用しており、タプル空間での上述のチェックをパスするものと仮定して、通常のタプル空間の

Temporal logic based model checking for parallel processes with a coordination language

†Hiroshi Ohtsuka, Department of Mathematical Science, Ehime University

動作の下で単体テストまたは従来の表明によるプログラム検証を行った。

3. 協調型言語の軌跡モデル

以前の研究[2]で通信相手を特定しない非同期型通信を行うプロセス代数を提案し、プロセスとタプル空間の組(様相)によるラベル付き遷移システム(有限状態機械, FSM)で意味を与え、更に軌跡によるプロセスの等価性を考察した。同時に簡単な協調型言語を実装し、この言語の通信に関する基本的操作に対する軌跡モデルを提案した。今回はモデル検証時のデータとして利用した。

4. 時相論理の導入

単体テストあるいはプログラム検証を終えた直列プロセスと再現環境としてのタプル空間に対して、線形時間命題型時相論理(詳しくは Hennessy-Milner Logic, HML)を利用したモデル検証[3]をおこなった。様相演算子は

- Always □ (HMLでは[K], [[K]])
- Sometime ◇ (HMLでは<K>,≪K≫)

を利用する。プロセス P が各々の様相論理式 Φ を満たす(P *sat* Φ)とは

P *sat* [K]Φ ≡ $\forall$Q(P $\xrightarrow{a}$ Q & a∈K). Q *sat* Φ

P *sat* <K>Φ ≡ $\exists$Q(P $\xrightarrow{a}$ Q & a∈K). Q *sat* Φ

等である。例えば軌跡 $a_1a_2\cdots a_n$, 拒絶 K に対応する式は≪a_1≫≪a_2≫…≪a_n≫*tt*, [[K]]*ff* であり、したがって失敗集合 ($a_1a_2\cdots a_n$, K) に対応する式は

≪a_1≫≪a_2≫…≪a_n≫([τ]*ff*∧[[K]]*ff*)

となる。また上の軌跡が完全軌跡であれば、対応する式は≪a_1≫≪a_2≫…≪a_n≫[-]*ff* となる。

5. 可達性解析

時相論理を利用して記述できる表明として重要なものに安全性と生存性がある。ここでは以下の安全性の検査を仕様の策定時およびタプル空間の利用時から行った。

I. デッドロックが起きないこと
これはシステムの機能分割時から検証し、タプル空間との組み合わせでは以下と共にバッファが空の状態に絞り込んだ。

II. バッファあふれがないこと
再現環境としてのタプル空間にはバッファを持ち込んでおり、仕様では同期型のプロセス間通信とすることで、検証時にはバッファにデータが残る状態への遷移を検査から除くなどの工夫を行った。

一般の並列プロセスを対象に可達性解析に基づくモデル検証を行う際、状態空間の中でデッドロック状態、ここでは待たされている通信データが無い(すべてのバッファが空)状態に近づくように受信処理(in)を優先して探索を進める。この探索を再現環境としてのタプル空間と開発中のプロセスの軌跡で行うようにした。この方法は FSM の能力を上げるわけではなく、実行効率もこの場合だけだが数%でしかなかった。しかし実装段階での通信時のエラーを通常の FSM の並列合成という手段で解析するには、実装を手作業で抽象化しなければならず非常に面倒である。この点ではプロセスの軌跡の利用は都合が良かったとも言える。

6. まとめ

実際に開発段階で利用すると、大半は時間制約(イベントの順序という意味ではない)が満たされない II の場合に該当した。しかしいくつか I の場合も確認できた。また II の場合もすべてが計算アルゴリズムの実装の間違いと言うわけではなく、少数だが I の場合と同じく仕様通りの実装でないことに原因があることが確認できた。更にこれらをプロセスの実装に近いレベルで検証(適合性試験)することが出来た。

今回は検証用に利用したツール/プロトタイプ言語[4]が開発言語とは異なり、またタプル空間の改良も平行して行ったので、手作業の部分が多くミスが混入したこともあった。出来れば自動化ツールに任せたい。また時間制約を扱える方がより実用的であったが、現在のタプル空間内での扱いに理論的な根拠があるわけではない。時間制約を扱う時相論理を採用すべきであろう。

7. 参考文献

[1] D. Gelernter, Generative Communication in Linda, ACM TOPLAS, Jan. 1985 (pp. 80-112)

[2] 本多他, 非同期型通信を用いた並列プロセスの意味論と並列言語への応用, 情報処理学会九州支部研究会報告, Mar. 1999 (pp. 26-33)

[3] G. Holzmann, Design and Validation of Computer Protocols. Prentice-Hall, 1991

[4] G. Holzmann, The model checker SPIN, IEEE TSE, 23, 5, 1997 (pp.279-295)