数理解析研究所講究録 1325 巻 2003 年 98-103

協調型言語に基づく並列プロセスのモデル検証

愛媛大学・理学部・数理科学科　大塚 寛 (Hiroshi Ohtsuka)
Department of Mathematical Science,
Faculty of Science, Ehime University

1　概要と背景

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。ここでは一般の並列プロセスではなく、制限された協調型言語 [1] を用いた並列プロセスの検証、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 [1] の間の通信を対象としたモデル検証 [3] あるいは適合性試験 [3] 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。

当該研究者はある実験的なシステムに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が、遠隔地であるゆえに実験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、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はグローバルバッファとして働き、通信相手が特定される必要はなく、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、通信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を中心にシステムを概観する。

実験の再現環境として　実験データを取得/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。同時に取り入れた時間間隔を超えてもアクセスがないデータの処理なども行う。実際には実験データを加工しタプル空間に送るプロセスを介して、実験装置からの直接的なデータだけでなく、入力パラメータに従った実験データの生成も行っている。

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コンポーネントに分割され、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。ここで各開発グループが担当するコンポーネントのソースを全員で共有することはせず、各コンポーネントの外部仕様 (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。この場合も実験の再現環境と同じく時間制約を設けている。

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。

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、単一プロセスの単体テスト時に利用している。我々が担当した単一プロセスは既知の数値計算アルゴリズムを通常の手続き型言語で実装しており、タプル空間へアクセスする基本的操作は同期型のみを利用している。この単一プロセスに対しては、仕様策定時には従来の表明による検証を、実装時には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た通常の動作の下で単

体テストを行った。

以上によりイベント駆動型プロセスの複数の並列動作を逐次型の複数の単一プロセスとタプル空間 (イベント駆動型) に分け、単一プロセスに対して動作検証を行なったことになる。

2 協調型言語の軌跡モデル

今回の実験では上で述べたように相手側の通信データを取得し、モデル検証および適合性試験時に利用しているが、このデータはプロセスの軌跡とみなせる。一方、以前の研究 [5] で通信相手を特定しない非同期型通信 [4] 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。これは今回の検証等の基礎となるので、以下にその概略を述べる。

2.1 タプル空間の仕様

ここでの並列プロセス間の通信とは、以下の仕様をもつ特殊なバッファプロセスと見なせるタプル空間を媒介とした非同期型通信 [4] である。通信されるメッセージはラベルとデータ (を格納する領域) の組であるが、今回の意味論を構築する際にはデータ部分は考慮しない。

out(v) タプル空間が満杯でない限り、タプル空間にメッセージ v を追加できる。

in(v) タプル空間に v のラベルと一致するメッセージが存在すれば、タプル空間からそれを取り出せる。

rd(v) タプル空間に v のラベルと一致するメッセージが存在すれば、タプル空間からそれを読み込む。タプル空間に変化はない。

タプル空間への操作は全て排他的に行われ、先着順に処理されるものとする [4]。操作的意味論はタプル空間の変化に着目して構成する。以下ではタプル空間をマルチセットと解釈する。

2.2 プロセス代数

ラベルの集合 Σ に対し**アクション**の集合 $\mathcal{A}$ を以下で定義する。ただし $\tau \notin \Sigma$ である。

$$\mathcal{A} := \{out(v) \mid v \in \Sigma\} \cup \{in(v) \mid v \in \Sigma\} \cup \{rd(v) \mid v \in \Sigma\} \cup \{\tau\}$$

τ 以外のアクション a に対し $label(a)$ で a のラベルを表す。

構文は CCS のサブセットで、並列プロセスがアクションプレフィックス、選択、再帰プロセスの内部に現れないもののみを扱う (cf. Simple CCS)。

定義 2.1 (プロセス表現) プロセス表現 P を以下のように再帰的に定義する。ただし $a \in \mathcal{A}$。

$$\begin{array}{lrcl} \textbf{並列プロセス} & P & ::= & S \mid P \parallel P \\ \textbf{単一プロセス} & S & ::= & x \mid \mathbf{0} \mid a.S \mid S + S \mid \mu x.S \end{array}$$

プロセス表現 P の**自由変数**、自由変数への**代入**、P が**ガード的**であること、P が**閉じている**こと等は CCS と同じ意味である。

定義 2.2 (プロセス) P に出現する再帰プロセス表現はすべてガード的であり、なおかつ閉じたプロセス表現 P を**プロセス**と呼び、プロセス全体の集合を $\mathcal{P}$ で表す。

定義 2.3 (様相) プロセス $P \in \mathcal{P}$ と Σ の元から成るマルチセット M の組 $<P, M>$ を**様相**と呼び、様相全体の集合を $\mathcal{C}$ で表す。

マルチセット M に元 v を追加する操作を $M+v$ で、M から元 v を削除する操作を $M-v$ で表す。

定義 2.4 ラベル付き遷移システムあるいは単に**遷移システム**とは、4項組 $<S, s_0, T, \longrightarrow>$ である。ここで S は状態の集合、$s_0 \in S$ は初期状態、T はラベルの集合、$\longrightarrow \subseteq S \times T \times S$ は遷移関係として定義される。表記法として $(s, a, s') \in \longrightarrow$ を $s \xrightarrow{a} s'$ と書く。

遷移関係 $\longrightarrow$ はその合成に拡張できる。ラベル $a_1, a_2, \cdots, a_n \in T$, $(n \geq 1)$ に対しその系列を $t = a_1 a_2 \cdots a_n \in T^*$ とするとき、$s, s' \in S$ に対し $s \xrightarrow{t} s'$ とは、$s \xrightarrow{a_1} s_1 \xrightarrow{a_2} s_2 \cdots \xrightarrow{a_n} s'$ といった一連の遷移関係があることである。このとき t を s の**軌跡**あるいは**トレース**、s' を s の t-**導出**と呼ぶ。特に $n = 0$ の場合 $s \xrightarrow{\varepsilon} s$ とする。また $D(s) = \{s' \mid \exists t \in T^* \text{ s.t. } s \xrightarrow{t} s'\}$ とおく。

様相に対する操作的意味を次の遷移関係から導出されるラベル付遷移システムで与える。

定義 2.5 様相 $<P, M>$ の遷移関係 $\longrightarrow$ を次の遷移規則で生成される最小の関係で定義する。

Tau
$$\frac{}{<\tau.P, M> \xrightarrow{\tau} <P, M>}$$

In
$$\frac{}{<in(v).P, M> \xrightarrow{in(v)} <P, M-v>} \quad (v \in M)$$

Out
$$\frac{}{<out(v).P, M> \xrightarrow{out(v)} <P, M+v>}$$

Rd
$$\frac{}{<rd(v).P, M> \xrightarrow{rd(v)} <P, M>} \quad (v \in M)$$

Sum
$$\frac{<P, M> \xrightarrow{a} <P', M'>}{<P+Q, M> \xrightarrow{a} <P', M'>} \qquad \frac{<Q, M> \xrightarrow{a} <Q', M'>}{<P+Q, M> \xrightarrow{a} <Q', M'>}$$

Rec
$$\frac{<P\{\mu x.P/x\}, M> \xrightarrow{a} <P', M'>}{<\mu x.P, M> \xrightarrow{a} <P', M'>}$$

Comp
$$\frac{<P, M> \xrightarrow{a} <P', M'>}{<P \| Q, M> \xrightarrow{a} <P' \| Q, M'>} \qquad \frac{<Q, M> \xrightarrow{a} <Q', M'>}{<P \| Q, M> \xrightarrow{a} <P \| Q', M'>}$$

プロセス P の意味とは、様相 $<P, \emptyset>$ を初期状態とする遷移システム $< D(<P, \emptyset>), <P, \emptyset>, \mathcal{A}, \longrightarrow>$ であり、プロセス P の軌跡の集合 $trace(P)$ とは、$<P, \emptyset>$ の軌跡の集合 $trace(<P, \emptyset>) = \{t \mid \exists s' \in \mathcal{C} \text{ s.t. } <P, \emptyset> \xrightarrow{t} s'\}$ のことである。

定義 2.6 (トレース等価) プロセス $P, Q \in \mathcal{P}$ が**トレース等価**であるとは、$trace(P) = trace(Q)$ すなわち $trace(<P, \emptyset>) = trace(<Q, \emptyset>)$ であるときに言い $P \sim_{tr} Q$ と書く。

定理 2.7 トレース等価性に関して、次の性質が成り立つ。ただし $a \in A$。

1. $\sim_{tr}$ は合同関係である。
2. $a.(P+Q) \sim_{tr} a.P + a.Q$

次に今回の検証に関係した並列プロセスの軌跡に関する性質を挙げる。

定理 2.8 $t \in trace(P\|Q)$ で $<P\|Q, \emptyset> \xrightarrow{t} <R, M>$ とすると

1. $\forall t' \leq t$ (プレフィックス), $\forall v \in \Sigma$ に対して $\quad |t' \uparrow in(v)| \geq |t' \uparrow out(v)|$
   すなわち t' の中に現れる $in(v)$ の個数は、t' の中に現れる $out(v)$ の個数を越えない。
2. $\forall v \in \Sigma$ に対して $\quad \#(M, v) \leq |t \uparrow out(v)| - |t \uparrow in(v)|$

特に M の元の個数については

$$\#M \leq \sum_{v \in \Sigma} |t \uparrow out(v)| - \sum_{v \in \Sigma} |t \uparrow in(v)|$$

第3節ではデッドロックが起きないこと (deadlock-free) を対象とした安全性の検証の例を挙げている。今回の実験では「バッファが空の状態でのバッファへの読み込み操作」すなわち「タプルがないタプル空間へ in 操作」が起きないことであり、括弧部分を軌跡を用いて表すと

$$\sum_{v \in \Sigma} |t \uparrow out(v)| < \sum_{v \in \Sigma} |t \uparrow in(v)|$$

すなわち $\#M = 0$ の場合に上式が成り立たない場合を対象とすることになる。

3 可達性解析

今回、モデル検証を利用した安全性の検証として deadlock-free であることをシステムの設計時から検証し、単一プロセスとタプル空間との組み合わせでは定理 2.8 直後に述べた状態に絞り込み、可達性解析時に対象とする FSM を生成するかわりに以下の方針で行なった。

1. 単一プロセス側では非決定的な動作を極力排してた上で軌跡を生成する。
2. タプル空間の動作をモデル化した状態空間の中で潜在的なデッドロック状態に近づけることを優先して探索を進める。
3. 2. の方針により、タプル空間に待たされているデータがない状態に進むために、単一プロセス側の可能な軌跡から受信処理 ($in(v)$) を送信処理 ($out(v)$) より優先して選ぶ。

なおもう一つの安全性である「バッファあふれがないこと」は時間制約に関わる事として今回の検証の対象とはしなかった。

以下では上の方法を、デッドロックが起きる場合と起きない場合の例を用いて説明する。いずれの場合も食事をする哲学者問題の 2 名 (phil1, phil2) をモデル化したコード断片を挙げる。

デッドロックが起きる場合　右図の phil1 では C1, C2 の順序で *in*, *out* 操作を行うのに対し、phil2 では C2, C1 の順序で *in*, *out* 操作を行う。それぞれの軌跡は以下の通り。

C1　in(C1)in(C2)out(C1)out(C2)in(C1)in(C2)···

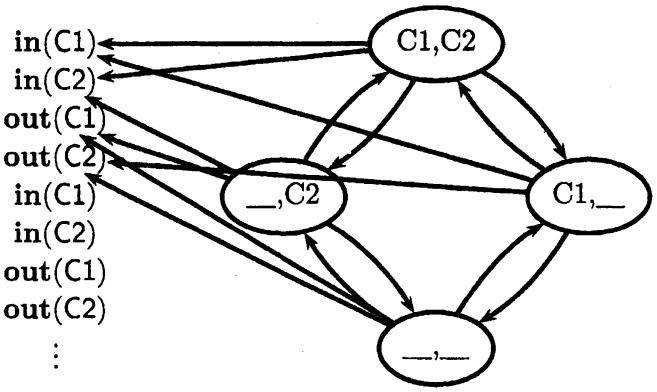
C2　in(C2)in(C1)out(C2)out(C1)in(C2)in(C1)···

またタプル空間内ではラベル C1, C1 に対応する長さ 1 のバッファが存在すると仮定する。

```
while(true) {
 think();
 in(C1);
 in(C2);
 eat();
 out(C1);
 out(C2);
}
```
phil1

```
while(true) {
 think();
 in(C2);
 in(C1);
 eat();
 out(C2);
 out(C1);
}
```
phil2

例 3.1 タプル空間をモデル化した FSM は C1, C2 各々の存在非存在に対応して 4 状態 (2 状態同士の直積) の状態空間となる。いずれも存在しない状態で *in* 操作しか行えない状況でデッドロックに陥る。このとき我々の方法では FSM の状態に対し遷移可能なアクションに対応する phil1(相手側コード) の軌跡へのポインタを持たせ (右図参照)、phil2(我々のコード) とタプル空間に対する可達性解析に利用する。



trace of phil1　　FSM of Tuple Space

この場合通常の可達性解析時の FSM と同様であり、状態数に変化はないので、ポインタを持つ分だけ複雑になる。

デッドロックが起きない場合　右図の phil1, phil2 では上の例の *in* 操作の前に *in*(token) を、*out* 操作の後に *out*(token) 操作を行う。それぞれの軌跡は以下の通りである。

C1　in(token)in(C1)in(C2)out(C1)out(C2)out(token)···

C2　in(token)in(C2)in(C1)out(C2)out(C1)out(token)···

またタプル空間内ではラベル token, C1, C1 に対応する長さ 1 のバッファが存在すると仮定する。

```
while(true) {
 think();
 in(token);
 in(C1);
 in(C2);
 eat();
 out(C1);
 out(C2);
 out(token);
}
```
phil1

```
while(true) {
 think();
 in(token);
 in(C2);
 in(C1);
 eat();
 out(C2);
 out(C1);
 out(token);
}
```
phil2

例 3.2 タプル空間をモデル化した FSM は token, C1, C2 各々の存在非存在に対応して 8 状態 (2 状態同士の直積) の状態空間となる。このとき我々の方法ではタプル空間の FSM の状態に対し遷移可能なアクションに対応する phil1 の軌跡へのポインタを持たせるのは上の例と同じだが、その際状態空間中の遷移の起こり得ない状態を除くことにより、5 状態に減らすことができる (次ページ図参照)。この例での要点は、相手側コードが仕様を守って実装していると仮定した上で、token の *in*, *out* 操作が一番上とその直下の状態の間でしかなされないことである。

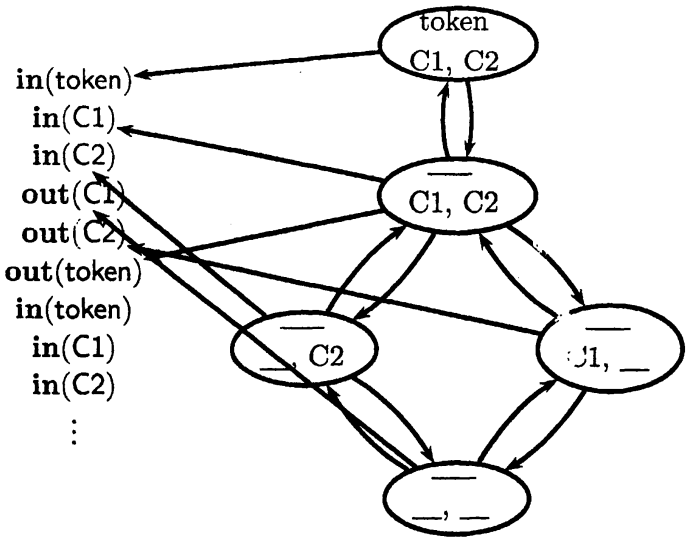
我々の方法を従来の FSM の並列合成による手法と比べると、

- プロセス間の直接の通信ではなく、タプル空間 (FSM) との通信に限った段階的なモデル検証/適合性試験

- FSM の状態空間の直積/最小化を経るのではなく、軌跡の interleaving に当たる部分を前もって FSM に適用した特殊な FSM と軌跡によるモデル検証/適合性試験

と捉えることができる。FSM の遷移可能なアクションを軌跡に関係付ける状況は、失敗集合 [2] を構成する軌跡と拒絶集合を FSM 中に保持する状況と似ていると思われる。また試験等価性 [4] に基づく手法と似ているが、軌跡を探索のためのガイドとして利用する部分が異なると思われる。しかしこれらの理論的根拠を確認するまでには至っていない。

なおこの手法により FSM の能力が上がるわけではなく、実行効率もこの場合では数%でしかなかった。しかし実装段階での通信時のエラーを解析するのに、プロセスの軌跡が利用出来たのは都合が良かったとも言える。



trace of phil1　　FSM of Tuple Space

4 課題

今回の実験/開発では設計時に failure–divergence モデルを採用したが、実装時のモデルにも採用したい。また「バッファあふれ」についてはタイムアウト処理で対処したが、実装時にはこれに関わる問題が頻繁に生じた。これに対処するには、時間オートマトンなどモデル中に時間制約を導入した上で検証/試験する必要がある。

なお協調型言語には非同期通信用の *inp* 操作があるが、そもそもプロセス間の非同期通信をタプル空間との通信に置き換えている事、*inp* 操作を使ったプログラミングは一般のプログラマにとって複雑である事などから、*inp* 操作を導入することには視野にいれていない。

参考文献

[1] D. Gelernter, Generative Communication in Linda, ACM Transactions on Programming Languages and Systems, pp. 80–112, January 1985

[2] C. A. R. Hoare, Communicating Sequential Process, Prentice Hall, 1985

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

[4] R. De Nicola and R. Pugliese, Testing semantics of asynchronous distributed programs, Analysis and Verification of Multiple-Agent Languages. 5th LOMAPS Workshop. Selected Papers, pp. 320-344, 1997

[5] 本多和正, 河原康雄, 森雅生, 大塚寛, 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, pp. 26-33, Mar. 1999