数理モデル化と問題解決 22−14
（1998. 11. 27）

リリース・コンシステンシ・モデルとその実現の形式的仕様記述について

高田 司郎 *,◇ 田口 研治 †
城 和貴 ‡ 福田 晃 *

* 奈良先端科学技術大学院大学 ◇ けいはんな
† 九州大学大学院 システム情報科学研究科
‡ 和歌山大学システム工学部

概要

我々は，分散共有メモリシステムの振る舞いを定義したメモリ・コンシステンシ・モデルとその実現の形式的な仕様記述とその検証の研究を行っている．我々は，仕様記述言語 Z と value-passing CCS を統合した形式的技法をコーザル・メモリ・コンシステンシ・モデルに適用した結果，そのモデルと実現の形式的仕様記述，および，その実現の検証がこの技法で一貫して行えることを確認した．そこで，本稿では，リリース・コンシステンシ・モデルに同様の形式的技法を適用して，そのモデルとその同期に関する実現の形式的仕様記述を行った．

1 はじめに

近年，分散共有メモリ（Distributed Shared Memory : DSM）システムは，並列計算機アーキテクチャおよびシステムソフトウェアの分野において注目されている．メモリ・コンシステンシ・モデルは，DSM システムをより効率的に実現するために多重メモリアクセスの振る舞いを定義したものであり，多くのモデルが提案されている [1, 2]．一般にメモリ・コンシステンシ・モデルの条件が弱ければ弱いほど，その DSM システムにおけるメモリ操作の振る舞いはより複雑になる．

そこで，我々は，メモリ・コンシステンシ・モデルとその実現の形式的な仕様記述と検証に関する研究を行ない，従来のモデルの正確な理解や比較を行うと共に，新しいモデルを提案することを目的とする．

我々は，田口と荒木が提案した Z 記述技法 [3] と value-passing CCS [4]（Calculus of Communicating Systems）を統合した形式的技法 [5] をコーザル・メモリ・コンシステンシ・モデルに適用した結果，そのモデルと実現の記述，および，その実現の検証がこの技法で一貫して行えることを確認した [6]．そこで，本稿では，リリース・コンシステンシ・モデル [7] に同様の形式的技法を適用して，そのモデルとその同期に関する実現の形式的仕様記述を行った．

以下，2 節では，本仕様記述方式の操作的意味論である「状態遷移に基づく CCS 意味論」について説明する．3 節では，状態遷移に基づく CCS 意味論の式を使ってリリース・コンシステンシ・モデルを記述する．4 節では，Z の表記法と value-passing CCS を組み合わせてリリース・コンシステンシ・モデルの同期に関する実現の形式的仕様記述とその展開例を示す．最後に，まとめと今後の課題を述べる．

2 状態遷移に基づく CCS 意味論

本節では，本稿において用いられる形式的手法の操作的意味論「状態遷移に基づく CCS 意味論（State-Based CCS Semantics : S-CCS）」とその遷移規則を説明する [5]．

2.1 ラベル付き遷移システム

Milner は CCS の操作的意味論として，以下のラベル付き遷移システムを与えた [4]．

$$\langle \mathcal{E}, Act, \{\xrightarrow{\alpha} \mid \alpha \in Act\}\rangle$$

このシステムは，CCS のエージェント表現の集合 $\mathcal{E}$，操作の集合 Act，および全ての操作 $\alpha \in Act$ のための遷移関係 $\xrightarrow{\alpha} \subset \mathcal{E} \times \mathcal{E}$ から構成される．例えば，操作 α によるプロセス E から別のプロセス E' への展開は以下の遷移関係で示される．

$$E \xrightarrow{\alpha} E'$$

田口と荒木は，Z の操作スキーマを古い状態から新しい状態へのラベル付き遷移システムとして，以下の操作的意味論を提案した．

$$\langle St, Op, \{\xrightarrow{\alpha} \mid \alpha \in Op\}\rangle$$

このシステムは，Z の状態集合 St，操作スキーマの集合 Op，および全ての操作スキーマ $\alpha \in Op$ のための遷移関係 $\xrightarrow{\alpha} \subseteq St \times St$ から構成される．例えば，操作スキーマ α による状態 s から別の状態 s' への展開は以下の遷移関係で示される．

$$s \xrightarrow{\alpha} s'$$

2.2 状態遷移に基づく CCS 意味論（S-CCS）

次に，彼らは，Z の表記法と value-passing CCS を組み合わせた操作的意味論を，以下のラベル付き遷移システムとして与えた．

$$\langle \mathcal{E} \times St, Act \cup Op, \{\xrightarrow{\alpha} \mid \alpha \in Act \cup Op\}\rangle$$

CCS の操作と Z の操作スキーマを区別するため $Act \cap Op = \emptyset$ とする．例えば，Z の操作スキーマ α によって，状態 s を持つプロセス E から状態 s' を持つ別のプロセス E' への展開は以下の遷移関係で示される．

$$\langle \alpha.E, s\rangle \xrightarrow{\alpha} \langle E, s'\rangle \Leftrightarrow \alpha.E \xrightarrow{\alpha} E \wedge s \xrightarrow{\alpha} s'$$

但し，Θ を操作スキーマ α の 1 階述語表現とするとき，$s, s' \models [\![\Theta]\!]$ を満足する．

2.3 遷移規則

操作スキーマと操作に関する展開は以下の遷移規則に従う．

Prefix operator (1)

$$\frac{}{\langle \alpha.E, s\rangle \xrightarrow{\alpha} \langle E, s'\rangle}\ (\alpha \in Op, s \xrightarrow{\alpha} s')$$

Prefix operator (2)

$$\frac{}{\langle \alpha.E, s\rangle \xrightarrow{\alpha} \langle E, s\rangle} \ (\alpha \in Act)$$

S-CCS では，Z の入出力変数として定義された変数を value-passing CCS の入出力ポートの変数として使用することで，環境と Z の操作スキーマ間の入出力を行なう．このため，value-passing CCS の表現では，この変数の項書き換えは認めていない．また，Z の表記法では，? を持つ変数 $x?$ は入力変数，! を持つ変数 $x!$ は出力変数である．そこで，出力ポート $\overline{\alpha}$ を通して環境に出力変数 $x!$ の値を送り出す $\overline{\alpha}(x!)$，入力ポート α を通して環境から Z で定義された入力変数 $x?$ の値を受けとる $\alpha(x?)$ は，それぞれ以下の遷移規則に従う．

Prefix operator (3)

$$\frac{}{\langle \overline{\alpha}(x!).E, s\rangle \xrightarrow{\overline{\alpha}(c)} \langle E, s\rangle} \ (s[\![x!]\!] = c)$$

Prefix operator (4)

$$\frac{}{\langle \alpha(x?).E, s\rangle \xrightarrow{\alpha(c)} \langle E, s'\rangle} \ (s' = s\{c/x?\})$$

以下，prefix operator が prefix operator(2) または (3) ならば，s' は s である．

Recursion

$$\frac{\langle E, s\rangle \xrightarrow{\alpha} \langle F, s'\rangle}{\langle P, s\rangle \xrightarrow{\alpha} \langle F, s'\rangle} \ (P \stackrel{\text{def}}{=} E)$$

Sum(Non-deterministic Choice)

$$\frac{\langle E_1, s\rangle \xrightarrow{\alpha} \langle F, s'\rangle}{\langle E_1 + E_2, s\rangle \xrightarrow{\alpha} \langle F, s'\rangle} \qquad \frac{\langle E_2, s\rangle \xrightarrow{\alpha} \langle F, s'\rangle}{\langle E_1 + E_2, s\rangle \xrightarrow{\alpha} \langle F, s'\rangle}$$

Concurrent Composition(1)

$$\frac{\langle E, s\rangle \xrightarrow{\alpha} \langle E', s'\rangle}{\langle E \mid F, s\rangle \xrightarrow{\alpha} \langle E' \mid F, s'\rangle} \qquad \frac{\langle F, s\rangle \xrightarrow{\alpha} \langle F', s'\rangle}{\langle E \mid F, s\rangle \xrightarrow{\alpha} \langle E \mid F', s'\rangle}$$

出力値（たとえば，c）が出力ポート $\overline{\alpha}$ から入力ポート α に通信する場合，以下の規則が適用される．

Concurrent Composition (2)

$$\frac{\langle E, s\rangle \xrightarrow{\overline{\alpha}(c)} \langle E', s\rangle \quad \langle F, s\rangle \xrightarrow{\alpha(c)} \langle F', s'\rangle}{\langle E \mid F, s\rangle \xrightarrow{\tau} \langle E' \mid F', s'\rangle}$$

操作 α と $\overline{\alpha}$ が値を伴わない通信（通常，同期を取る通信）の場合は，以下の規則が適用される．

Concurrent Composition (3)

$$\frac{\langle E, s\rangle \xrightarrow{\overline{\alpha}} \langle E', s\rangle \quad \langle F, s\rangle \xrightarrow{\alpha} \langle F', s\rangle}{\langle E \mid F, s\rangle \xrightarrow{\tau} \langle E' \mid F', s\rangle}$$

Restriction

$$\frac{\langle E, s\rangle \xrightarrow{\alpha} \langle F, s'\rangle}{\langle E \setminus L, s\rangle \xrightarrow{\alpha} \langle F \setminus L, s'\rangle} \ (\alpha \notin L, \alpha \in Act)$$

Renaming

$$\frac{\langle E, s\rangle \xrightarrow{\beta} \langle F, s'\rangle}{\langle E[f], s\rangle \xrightarrow{\alpha} \langle F[f], s'\rangle} \ (\alpha \in Act, \alpha = f(\beta))$$

Stirling は一つの操作の遷移関係 $\xrightarrow{\alpha}$ を有限長の操作 **トレース** $\alpha_1 \ldots \alpha_n$ に対して以下のような自然な拡張を提案した [8]．ω は，空のトレース ε を持った列とする．表記法 $E \xrightarrow{\omega} F$ は「E はトレース ω を逐次展開して F になる」という関係である．

$$\frac{}{E \xrightarrow{\varepsilon} E} \qquad \frac{E \xrightarrow{\alpha} E' \quad E' \xrightarrow{\omega} F}{E \xrightarrow{\alpha\omega} F}$$

我々は，遷移規則を有限長のトレースへ自然に拡張した [6]．例えば，トレース ω が操作スキーマ $\alpha_1 \ldots \alpha_n$ を含む場合，その展開の遷移関係を，以下に示す．

$$\langle E, s_1\rangle \xrightarrow{\omega} \langle F, s_n'\rangle \Leftrightarrow E \xrightarrow{\omega} F \wedge s_1 \xrightarrow{\omega} s_n'$$

但し，n を ω の中の操作スキーマの個数，Θ_i を α_i の 1 階述語による表現とするとき，$\forall i : 1 \ldots n \bullet s_i, s_i' \models [\![\Theta_i]\!]$ を満たす．

Trace

$$\frac{}{\langle E, s\rangle \xrightarrow{\varepsilon} \langle E, s\rangle} \qquad \frac{\langle E, s\rangle \xrightarrow{\alpha} \langle E', s'\rangle \quad \langle E', s'\rangle \xrightarrow{\omega} \langle F, s''\rangle}{\langle E, s\rangle \xrightarrow{\alpha\omega} \langle F, s''\rangle}$$

また，本稿では，4.3 節にて value-passing CCS の *Conditional* 構文を使用している．そこで，以下の遷移規則を追加する．

Conditional

$$\frac{\langle E, s\rangle \xrightarrow{\alpha} \langle E', s'\rangle}{\langle \textit{if } b \textit{ then } E, s\rangle \xrightarrow{\alpha} \langle E', s'\rangle} \ (b \textit{ is true})$$

3 リリース・コンシステンシ・モデルの記述

本節は，Gharacholoo によって提案されたリリース・コンシステンシ・モデル [7] について説明し，S-CCS の式を使ってこのモデルを定義する．

3.1 共有メモリ並列計算機モデル

まず，リリース・コンシステンシ・モデルを適用する共有メモリ並列計算機モデルを定義する．

- プロセス $\{P_1, \ldots, P_n\}$ を含む有限集合 $\mathcal{P}$ から構成され，有限なアドレス空間からなる共有メモリを通して，ロード命令，ストア命令，および同期命令の列によって相互作用する．
- プロセス P_i が発行するロード命令 $r_i(x?, v!)$ は，不可分な命令であり，値 v がアドレス x に格納されていることを P_i に通知する．
- プロセス P_i が発行するストア命令 $w_i(x?, v?)$ は，不可分な命令であり，値 v をアドレス x に格納する．
- プロセス P_i が発行する同期命令は，同期変数を持たない $sync_i$ と，同期変数 z をもつ獲得命令 $acq_i(z)$ および開放命令 $rel_i(z)$ からなる．また，獲得命令と開放命令は，それぞれ，ロード命令とストア命令の一種である．
- プロセス P_i が発行する観測命令 $per_i(o)$ は，プロセス P_k が発行した命令 o を共有メモリを通して観測して，プロセス P_i のローカルメモリに反映させる．但し，P_i が自ら発行した命令 o の場合は，その命令 o そのものとする．

例えば $per_i(r_i(x?, v!))$, $per_i(w_j(x?, v?))$ は，それぞれ，P_i が発行したロード命令 $r_i(x?, v!)$，P_j が発行した $w_j(x?, v?)$ を P_i のローカルメモリに格納する $Apply_i$ 命令である（4.2 節を参照）．以下，上記で定義した命令の実行履歴と順序関係を定義する．

プロセス P_i が逐次発行したロード命令，ストア命令，および同期命令の実行履歴を L_i とする．例えば $L_1 = \{w_1(x, 1), r_1(y, 2)\}$ のようなリストで表す．

また，全てのプロセッサの実行履歴の集合を $H = \langle L_1, L_2, \ldots, L_n \rangle$ とする．これら実行履歴に含まれる命令の実行順序関係は以下の通り．

1. $o_1 \overset{}{\underset{i}{\rightarrow}} o_2$: 実行履歴 L_i に含まれる命令 o_1, o_2 について，o_1 は o_2 より先行（先に実行）している．
2. $o_1 \rightarrow o_2$: 実行履歴 H に含まれる命令 o_1, o_2 について，o_1 は o_2 より先行している．

次に，実行履歴 H に含まれる全ての命令と，全てのプロセスにおいて別のプロセスから観測した命令に関する全ての観測命令を，実行順序で列にしたものを逐次実行列 S と言い，プロセス間の共有メモリを通した相互作用を表わしている．例えば $S = \{w_1(x,1), w_2(y,2), per_1(w_2(y,2)), r_1(y,2)\}$ のような列で，どのアドレスにどのプロセスが書き込みを行いそれをどのプロセスが観察して読み込んだかが分かる．同様に，逐次実行列 S に含まれる命令の中からプロセス P_i が発行した命令列を逐次実行列 S_i と言い，プロセス P_i と他のプロセスとの共有メモリを通した相互作用が分かる．例えば，$S_1 = \{w_1(x,1), per_1(w_2(y,2)), r_1(y,2)\}$ では，プロセス P_1 が発行した $w_1(x,1), r_1(y,2)$ とプロセス P_2 の発行した $w_2(y,2)$ とを共有メモリを通して観測した結果，$w_1(x,1), w_2(y,2), r_1(y,2)$ のプログラム順序でプロセス P_2 と相互作用したことが分かる．

A を 実行履歴 H の全ての命令の集合，A^H_{i+w} を P_i が発行した全ての命令と，その他の全てのプロセスが発行したストア命令と開放命令の和集合とする．また，$A^H_{i+w} \mid z$ を命令の集合 A^H_{i+w} の中から変数 z を含む命令のみを取り出した命令の集合，$S_i \mid z$ を逐次実行列 S_i の中から変数 z を含む命令のみを取り出した逐次実行列とする．逐次実行列 S_i に含まれる命令の実行順序関係は以下の通り．

3. $o_1 \overset{}{\underset{S_i}{\rightarrow}} o_2$: 逐次実行列 S_i に含まれる命令 o_1, o_2 に関して，o_1 は o_2 より先行している．
4. 同期変数 z に関する $A^H_{i+w} \mid z$ の逐次実行列 $S_i \mid z$ が有効 : $S_i \mid z$ は命令集合 $A^H_{i+w} \mid z$ の全ての命令を含み，かつ，$S_i \mid z$ 内の全ての獲得命令には，それぞれで異なる開放命令が先行している．但し，先行する開放命令がない場合は，$S_i \mid z$ で最初の獲得命令である．

次に，上記で定義された順序関係 1 ～ 4 を S-CCS の式で記述する．上記で定義した共有メモリ並列計算機モデルを 4 節で定義される $\langle \mathcal{E} \times St, Act \cup Op, \{\overset{\alpha}{\rightarrow} \mid \alpha \in Act \cup Op\} \rangle$ のラベル付き遷移システムとする．ω を逐次実行列 S の部分列 A^* からなる有限長な命令トレース，$E_0, E_1, E'_1, E_2, E'_2, E_i, E_j$ を $\mathcal{E}$ の要素，$s_0, s_1, s'_1, s_2, s'_2, s_i, s_j$ を St の要素とするとき，上記の 1 ～ 4 で定義された順序関係は，逐次実行列 S の有限長な命令トレース ω を使って，それぞれ以下のように記述できる．

- $o_1, o_2 \in L_i, \exists \omega \in A^* \bullet$
  $\langle o_1.E_1, s_1 \rangle \overset{\omega}{\rightarrow} \langle o_2.E_2, s_2 \rangle$
- $o_1, o_2 \in H, \exists \omega \in A^* \bullet$
  $\langle o_1.E_1, s_1 \rangle \overset{\omega}{\rightarrow} \langle o_2.E_2, s_2 \rangle$
- $o_1, o_2 \in S_i, \exists \omega \in A^* \bullet$
  $\langle o_1.E_1, s_1 \rangle \overset{\omega}{\rightarrow} \langle o_2.E_2, s_2 \rangle$
- $(\forall o \in A^H_{i+w} \mid z \bullet \exists per_i(o) \in S_i)$
  $\wedge$
  $j : 1 \ldots n, \forall acq_i(z) \in S_i \bullet$
  $((\exists per_i(rel_j(z)) \in S_i,$
  $rel_j(z), per_i(rel_j(z)), acq_i(z) \notin \omega \bullet$
  $\langle per_i(rel_j(z)).E_j, s_j \rangle$
  $\overset{per_i(rel_j(z))\omega}{\rightarrow} \langle acq_i(z).E_i, s_i \rangle)$
  $\vee$
  $(per_i(rel_j(z)) \notin \omega \bullet$
  $\langle start.E_0, s_0 \rangle \overset{\omega}{\rightarrow} \langle acq_i(z).E_i, s_i \rangle))$

3.2 リリース・コンシステンシ・モデル

リリース・コンシステンシ・モデルの基本的な考え方は，プログラマが同期命令を使ってプログラムの同期を取ることで，DSM のオーバヘッドを軽減することである．プログラム中の同期命令には，獲得（ロック）と開放（アンロック）があり，これら同期命令によってストア命令の遅延が隠蔽される．このモデルは以下のような制約条件で定義されている [7, 1, 2].

- 他の全てのプロセスに関して通常のロード命令やストア命令の実行が許される前に，それ以前のすべての獲得命令が完了していなければならない．
- 他の全てのプロセスに関して開放命令の実行が許される前に，それ以前の全てのロード命令やストア命令が完了していなければならない．
- 獲得命令と開放命令はプロセッサ・コンシステンシ・モデルに従う．

以下，形式的に定義する．但し，最後の条件は，獲得命令と開放命令が異なる同期変数を持つ場合，獲得命令が開放命令を飛び越すことがあるという条件である．この条件については，本稿の目的とする同期に関する仕様記述の本質的な問題ではないため，紙面の関係上省略する．

5. 各同期変数 z に関する，各プロセス P_i の命令の集合 $A^H_{i+w} \mid z$ に有効な逐次実行列 $S_i \mid z$ が存在して，$A^H_{i+w} \mid z$ の任意の獲得命令 $acq_i(z)$ と，$rel_j(z) \overset{}{\underset{i}{\rightarrow}} acq_i(z)$ であるような開放命令 $rel_j(z)$ は，$rel_j(z) \overset{}{\underset{S_i \mid z}{\rightarrow}} acq_i(z)$ が成り立ち，この $acq_i(z)$ に引き続くプロセス P_i のロード命令やストア命令 o は，$acq_i(z) \overset{}{\underset{S_i}{\rightarrow}} o$ を満たす．
6. 各プロセス P_i の任意の獲得命令 o_{rel} と，$o_1 \overset{}{\underset{i}{\rightarrow}} o_{rel}$ を満たす P_i の任意のロード命令やストア命令 o_i は，その o_{rel} と o_i 含むどのような逐次実行列 S_i においても，$o_j \overset{}{\underset{S_i}{\rightarrow}} o_{rel}$ を満たす．

以下，5,6 の条件を，S-CCS の式を使って記述する．Syn を同期変数の集合とする．

$i : 1 \ldots n \bullet$
$((\forall z \in Syn \bullet$
$((\forall o \in A^H_{i+w} \mid z \bullet \exists per_i(o) \in S_i)$
$\wedge$
$j : 1 \ldots n, \forall acq_i(z) \in S_i \bullet$
$(((\exists per_i(rel_j(z)) \in S_i,$
$rel_j(z), per_i(rel_j(z)), acq_i(z) \notin \omega \bullet$
$\langle per_i(rel_j(z)).E_j, s_j \rangle$
$\overset{per_i(rel_j(z))\omega}{\rightarrow} \langle acq_i(z).E_i, s_i \rangle)$
$\vee$
$(per_i(rel_j(z)) \notin \omega \bullet$
$\langle start.E_0, s_0 \rangle \overset{\omega}{\rightarrow} \langle acq_i(z).E_i, s_i \rangle)))$
$\wedge$
$(\forall o \in L_i \; of \; A^H_{i+w} \bullet$
$\langle acq_i(z).E_1, s_1 \rangle \overset{\omega_1}{\rightarrow} \langle o.E_2, s_2 \rangle \Rightarrow$
$\langle acq_i(z).E_1, s_1 \rangle \overset{\omega_2}{\rightarrow} \langle per_i(o).E'_2, s'_2 \rangle))))$
$\wedge$
$j : 1 \ldots n, \forall rel_i(y) \in S_i \bullet$
$(\forall o \in L_i \; of \; A^H_{i+w} \bullet$
$\langle o.E_1, s_1 \rangle \overset{\omega_1}{\rightarrow} \langle rel_i(y).E_2, s_2 \rangle \Rightarrow$
$\langle per_i(o).E'_1, s'_1 \rangle \overset{\omega_2}{\rightarrow} \langle rel_i(y).E_2, s_2 \rangle))$

4 リリース・コンシステンシ・モデルの実現

この節では，リリース・コンシステンシ・モデルの実現 $\mathcal{RM}$ の同期に関する仕様を Z の表記法と

value-passing CCS を統合して記述する．まず，本稿で採用する $\mathcal{RM}$ の実現方式について説明する．次に，Z の表記法と value-passing CCS を用いて $\mathcal{RM}$ の機能と並行性について，それぞれ分離して記述する．最後に，S-CCS の遷移規則を適用した $\mathcal{RM}$ の展開例を挙げる．

4.1 実現方式 [2]

$\mathcal{RM}$ で採用するリリース・コンシステンシ・モデルの実現方式は，以下のものとする．

- 同期モデル：セマフォを使った分散同期サービス
- 更新オプション：書き込み時更新
- 粒度：変数単位 [1]

書き込み時更新とは，共有メモリの更新はあるプロセスによってローカルに作られ，データ項目のコピーを他の全てのプロセスにマルチキャストして，それぞれのローカルメモリが更新されるまで待つ方式である．リリース・コンシステンシ・モデルでは，クリティカル・セクションで発行された更新については，開放命令が終了するまでにコピーが他の全てのプロセスに反映されていればよい．

4.2 機能の記述

各プロセス P_i は，以下の状態スキーマ s_i を持つ．各 s_i は，プロセス番号 pn_i，ローカルメモリ M_i，同期変数 z_i，ストア命令番号 t_i，他プロセスとの通信キュー $OutQueue_i$ と $InQueue_i$，受け取り確認関数 $Confirm_i$，実行履歴 L_i，および命令集合 A^H_{i+w} の逐次実行列 S_i を持つ．

$$[\mathcal{M}, A, Val, Syn]$$

$$write\_tuple == Syn \times \mathbb{N}_1 \times (\mathbb{N}_1 \cup \{\top\}) \times \mathbb{N}_1 \times \mathcal{M} \times Val$$
$$confirmed\_process == (Syn, \mathbb{N}_1) \pfun \mathrm{seq}\,\mathbb{N}_1$$
$$Bool == \{true, false\}$$

$$MaxOutQueue, MaxInQueue : \mathbb{N}_1$$
$$MaxSerial, MaxLocalHis : \mathbb{N}_1$$
$$update\_confirmation : Syn \times confirmed\_process \pfun confirmed\_process$$

s_i

$$pn_i : \mathbb{N}_1$$
$$M_i : \mathcal{M} \pfun (Val \cup \{\bot\})$$
$$z_i : Syn \cup \{\bot\}$$
$$t_i : \mathbb{N}$$
$$OutQueue_i : \mathrm{seq}\ write\_tuple$$
$$InQueue_i : \mathrm{seq}\ write\_tuple$$
$$Confirm_i : confirmed\_process$$
$$L_i : \mathrm{seq}\ A$$
$$S_i : \mathrm{seq}\ A$$

$$\#OutQueue_i \leq MaxOutQueue$$
$$\#InQueue_i \leq MaxInQueue$$
$$\#L_i \leq MaxLocalHis$$
$$\#S_i \leq MaxSerial$$

各プロセス P_i は，操作スキーマ $InitP_i$ で初期化される．また，後述する入力ポート loc_i を通して環境からアドレス $x?$ を受け取ると，操作スキーマ $Read_i$ を発行してローカルメモリ $M_i(x?)$ の値 $v!$ を受理する．また，実行履歴 L_i と逐次実行列 S_i にロード命令のラベル $r_i(x?, v!)$ を追加する．これら操作スキーマは紙面の関係で省略する．

[1] 通常，物理的なページ単位を粒度とするが，仮想メモリの問題を含む複雑な問題となるため本稿では変数単位とした．

各プロセスは P_i は，後述する入力ポート $heap_i$ を通して環境からアドレス $x?$ と値 $v?$ を受け取ると，以下の操作スキーマ $Write_i$ を発行してローカルメモリ $M_i(x?)$ の値を $v?$ に更新する．また，出力キュー $OutQueu_i$ に更新メッセージ (同期変数 z_i, 起点プロセス番号 pn_i, 終点プロセス番号, ストア命令番号 $t'_i, x?, v?$) を追加する．尚，到着プロセス番号が $\top$ とは全てのプロセスを示す．また，同期変数 z_i とストア命令番号 t'_i で区別できるこのストア命令に関する書き込み更新メッセージを送ってきたプロセスのプロセス番号列を値とする受け取り確認関数 $Confirm_i$ の初期値を <> にする．同様に，実行履歴 L_i と逐次実行列 S_i にストア命令のラベル $w_i(x?, v?)$ を追加する．

$Write_i$

$$\Delta s_i$$
$$x? : \mathcal{M}$$
$$v? : Value$$

$$pn'_i = pn_i$$
$$M'_i = M_i \oplus \{x? \mapsto v?\}$$
$$z'_i = z_i$$
$$t'_i = t_i + 1$$
$$OutQueue'_i = OutQueue_i \frown < (z_i, pn_i, \top, t'_i, x?, v?) >$$
$$InQueue'_i = InQueue_i$$
$$Confirm'_i = Confirm_i \oplus \{(z_i, t'_i) \mapsto <>\}$$
$$L'_i = L_i \frown < w_i(x?, v?) >$$
$$S'_i = S_i \frown < w_i(x?, v?) >$$

各プロセス P_i は，操作スキーマ $Send_i$ を適時発行して，出力キュー $OutQueu_i$ 内の更新情報を，後述する $\overline{pipe_j}$ を通して他の全てのプロセスに同報通信する．また，後述する $pipe_j$ を通して通信されたメモリ更新の情報を受け取ると，操作スキーマ $Receive_i$ を発行して入力キュー $InQueue_i$ に追加する．これらも紙面の関係上省略する．

各プロセス P_i は，以下の操作スキーマ $Apply_i$ を適時発行して，$InQueue_i$ に格納された更新メッセージの中から終点プロセス番号が $\top$ の更新メッセージを選択して，ローカルメモリに逐次反映させる．また，この起点プロセス番号 src_j のプロセスにストア命令書き込み確認メッセージを送るために，書き込み更新メッセージを $OutQueue_i$ に追加する．

$Apply_i$

$$\Delta s_i$$
$$(z_j, src_j, tar_j, t_j, x_j, v_j) : write\_tuple$$

$$InQueue_i \neq <>$$
$$pn'_i = pn_i$$
$$(z_j, src_j, tar_j, t_j, x_j, v_j) = head\ InQueue_i$$
$$tar_j = \top$$
$$M'_i = M_i \oplus \{x_j \mapsto v_j\}$$
$$z'_i = z_i$$
$$t'_i = t_i$$
$$OutQueue'_i = OutQueue_i \frown < (z_j, pn_i, src_j, t_j, x_j, v_j) >$$
$$InQueue'_i = tail\ InQueue_i$$
$$Confirm'_i = Confirm_i$$
$$L'_i = L_i \frown < Apply_i(w_j(x_j, v_j)) >$$
$$S'_i = S_i \frown < per_i(w_j(x_j, v_j)) >$$

各プロセス P_i は，後述する入力ポート syn_a_i を通してセマフォから同期変数 $s?$ を受け取ると，操作スキーマ Acq_i を発行して，$s?$ に対応したクリティカル・セクションに入る．同様に，実行履歴 L_i と逐次実行列 S_i に獲得命令のラベル $acq_i(s?)$ を追加する．

Acq_i

Δs_i
$s? : Syn$

$pn_i' = pn_i$
$M_i' = M_i$
$z_i' = s?$
$t_i' = t_i$
$OutQueue_i' = OutQueue_i$
$InQueue_i' = InQueue_i$
$Confirm_i' = Confirm_i$
$L_i' = L_i^\frown < acq_i(s?) >$
$S_i' = S_i^\frown < acq_i(s?) >$

各プロセス P_i は，後述する入力ポート syn_r_i を通してセマフォから同期変数 $s?$ を受け取ると，操作スキーマ Acq_i を発行して，$s?$ に対応した書き込み更新メッセージを全てのプロセスから受けたか確認する．関数 *update_confirmation* は，$s?$ と受け取り確認関数 $Confirm_i$ を入力として，確認済みであれば，$update!$ に *true* を，そうでなければ，*false* を返す関数である．また，*true* の場合は，$s?$ に関係する定義域と値の対は全て $Confirm_i$ から取り除く．この関数の記述は，紙面の関係上，省略する．同様に，実行履歴 L_i と逐次実行列 S_i に開放命令のラベル $rel_i(s?)$ を追加する．

Rel_i

Δs_i
$(z_j, src_j, tar_j, t_j, x_j, v_j) : write\_tuple$
$s? : Syn$
$update! : Bool$

$InQueue_i \neq <>$
$(z_j, src_j, tar_j, t_j, x_j, v_j) = head\ InQueue_i$
$tar_j = pn_i$
$InQueue_i' = tail\ InQueue_i$
$pn_i' = pn_i$
$M_i' = M_i$
$t_i' = t_i$
$OutQueue_i' = OutQueue_i$
$Confirm_i' = update\_confirmation(z_j, Confirm_i \oplus \{(z_j, t_j) \mapsto (Confirm_i(z_j, t_j)^\frown < src_j >)\}, update!)$
$update! = true$
$z_i' = \perp$
$L_i' = L_i^\frown < rel_i(s?) >$
$S_i' = S_i^\frown < rel_i(s?) >$

4.3 並行性の記述

この節では，value-passing CCS を用いて，$\mathcal{RM}$ の並行性および同期に関する形式的仕様記述を行う．各プロセス P_i は，入出力ポートとして，

$$P_i : \{id_i, loc_i, \overline{val_i}, heap_i, pipe_i, \overline{pipe_i}\}$$

を持つ．入力ポート id_i は，プロセス番号を環境から入力する．入出力ポート $pipe_i$ は，プロセス間の通信ポートである．その他のポートは，以下のロード命令，ストア命令内で使用される．

ラベル $r_i(x?, v!)$ は，不可分なロード命令であり，入力ポート loc_i を通して環境からアドレス $x?$ を受け取り，操作スキーマ $Read_i$ を実行した後，ローカルメモリ $M_i(x?)$ の値 $v!$ を出力ポート $\overline{val_i}$ を通して環境に出力する．但し，このロード命令が不可分であることの制約を記述していない．これはこの操作に関するセマフォを追加することで記述できるが，仕様記述が煩雑になるため省略する．

$$r_i(x?, v!) \equiv loc_i(x?).Read_i.\overline{val_i}(v!)$$

ラベル $w_i(x?, v?)$ は，不可分なストア命令であり，入力ポート $heap_i$ を通して環境からアドレス $x?$ と値 $v?$ を受け取り，操作スキーマ $Write_i$ を実行する．この操作も同様に不可分な命令の制約記述は省略する．

$$w_i(x?, v?) \equiv heap_i(x?, v?).Write_i$$

ラベル $broadcast_i(message!)$ は，不可分な同報通信命令であり，操作スキーマ $Send_i$ を実行後，$\overline{pipe_i}$ 以外の出力ポート $\overline{pipe_j}$ を通して他の全てのプロセスと通信する．この操作も同様に不可分な命令の制約記述は省略する．

$$\begin{aligned} broadcast_i(message!) \quad &\equiv \quad Send_i. \\ &\overline{pipe_1}(message!). \quad \cdots \quad .\overline{pipe_{i-1}}(message!). \\ &\overline{pipe_{i+1}}(message!). \quad \cdots \quad .\overline{pipe_k}(message!) \end{aligned}$$

ラベル $acq_i(s?)$ は，同期変数 $s?$ を持つ不可分な獲得命令であり，入力ポート syn_a_i を通して環境から $s?$ を受け取り，操作スキーマ Acq_i を実行する．次に，後述するセマフォ Sem から入力ポート acq を通して $s?$ を受け取り，$s?$ に対応したクリティカル・セクションに入る．この操作も同様に不可分な命令の制約記述は省略する．

$$acq_i(s?) \equiv syn\_a_i(s?).Acq_i.acq(s?).$$

ラベル $rel_i(s?)$ は，同期変数 $s?$ を持つ不可分な開放命令であり，入力ポート syn_r_i を通して環境から $s?$ を受け取り，エージェント $check_i(s?)$ を起動する．そして，入力ポート $confirm_i$ を通して操作スキーマ Rel_i から $update!$ を受け取り，$s?$ に対応したクリティカル・セクション内の書き込み時更新が終了するまで確認を繰り返す．終了確認後，出力ポート $\overline{rel}$ を通してセマフォ Sem に $s?$ を送り，$s?$ に対応したクリティカル・セクションから出る．この操作も同様に不可分な命令の制約記述は省略する．

$$\begin{aligned} rel_i(s?) \quad &\equiv \quad syn\_r_i(s?).check_i(s?).\overline{rel}(s?) \\ check_i(s?) \quad &\stackrel{\text{def}}{=} \quad Rel_i.confirm_i(update!). \\ &((if \quad update! \quad = true\ then\ \mathbf{0}) \\ &+(if \quad update! \quad = false\ then\ check_i(s?))) \end{aligned}$$

以上の定義により，以下のプロセス P_i が定義される．

$$\begin{aligned} P_i \stackrel{\text{def}}{=} \; & acq_i(x?).P_i + \\ & r_i(x?, v!).P_i + \\ & w_i(x?, v?).P_i + \\ & broadcast_i(message!).P_i + \\ & receive_i(message?).P_i + \\ & Apply_i.P_i + \\ & rel_i(x?).P_i \end{aligned}$$

各プロセス P_i は以下のセマフォで同期を取る．但し，同期変数の集合 $Syn = \{a, \cdots z\}$ に対するセマフォは，それぞれ，以下の $Sem_a \cdots Sem_z$ とする．

$$\begin{aligned} Sem_a \quad &\stackrel{\text{def}}{=} \quad \overline{acq}(a).rel(a).Sem_a \\ &\cdots \\ Sem_z \quad &\stackrel{\text{def}}{=} \quad \overline{acq}(z).rel(z).Sem_z \\ \mathcal{RM} \quad &\stackrel{\text{def}}{=} \quad id(pn_1?).InitP_1.\cdots.id(pn_n?).InitP_n. \\ &\quad (Sem_a \mid \cdots \mid Sem_z \mid P_1 \mid \cdots \mid P_n) \setminus L \\ where\ L \quad &= \quad \{acq(a), \cdots, acq(z), rel(a), \cdots, rel(z), \\ &\quad pipe_1, \cdots, pipe_n\} \end{aligned}$$

4.4 $\mathcal{RM}$ の展開例

上記で定義した $\mathcal{RM}$ の同期に関係する部分の展開例を以下に示す．但し，簡略化のため，プロセスは，P_1, P_2 の二個，同期変数は，y, z の二個とする．

$$
\begin{aligned}
& \langle \mathcal{CM}, s_0 \rangle \\
\xrightarrow{\omega\, syn\_a_1(z)} & \langle (Sem_y \mid Sem_z \mid Acq_1.acq(z).P_1 \mid P_2) \setminus L, s'_0 \rangle \\
\xrightarrow{Acq_1} & \langle (Sem_y \mid \overline{acq}(z).rel(z).Sem_z \mid acq(z).P_1 \mid P_2) \setminus L, s'_1 \rangle \\
\xrightarrow{\tau} & \langle (Sem_y \mid rel(z).Sem_z \mid w_1(x_1,1).P_1 \mid P_2) \setminus L, s'_2 \rangle \\
\xrightarrow{w_1(x_1,1)} & \langle (Sem_y \mid rel(z).Sem_z \mid broadcast_1(message!).P_1 \mid P_2) \setminus L, s'_3 \rangle \\
\xrightarrow{broadcast_1(z,1,\top,1,x_1,1)} & \langle (Sem_y \mid rel(z).Sem_z \mid P_1 \mid receive_2(message?).P_2) \setminus L, s'_4 \rangle \\
\xrightarrow{receive_2(z,1,\top,1,x_1,1)} & \langle (Sem_y \mid rel(z).Sem_z \mid P_1 \mid Apply_2.P_2) \setminus L, s'_5 \rangle \\
\xrightarrow{Apply_2} & \langle (Sem_y \mid rel(z).Sem_z \mid receive_1(message?).P_1 \mid P_2) \setminus L, s'_6 \rangle \\
\xrightarrow{receive_1(z,2,1,1,x_1,1)} & \langle (Sem_y \mid rel(z).Sem_z \mid syn\_r_1(s?).P_1 \mid P_2) \setminus L, s'_7 \rangle \\
\xrightarrow{syn\_r_1(z)} & \langle (Sem_y \mid rel(z).Sem_z \mid check_1(z).\overline{rel}(z).P_1 \mid P_2) \setminus L, s'_8 \rangle \\
\xrightarrow{check_1(z)} & \langle (Sem_y \mid rel(z).Sem_z \mid \overline{rel}(z).P_1 \mid P_2) \setminus L, s'_9 \rangle \\
\xrightarrow{\tau} & \langle (Sem_y \mid Sem_z \mid P_1 \mid syn\_a_2(s?).P_2) \setminus L, s'_{10} \rangle \\
\xrightarrow{syn\_a_2(z)} & \langle (Sem_y \mid Sem_z \mid P_1 \mid Acq_2.acq(z).P_2) \setminus L, s'_{11} \rangle \\
\xrightarrow{Acq_2} & \langle (Sem_y \mid \overline{acq}(z).rel(z).Sem_z \mid P_1 \mid acq(z).P_2) \setminus L, s'_{12} \rangle \\
\xrightarrow{\tau} & \langle (Sem_y \mid rel(z).Sem_z \mid P_1 \mid P_2) \setminus L, s'_{13} \rangle \\
& \cdots
\end{aligned}
$$

5 終わりに

本稿では，コーザル・メモリ・コンシステンシ・モデルの形式的仕様記述と検証 [6] に続いて，リリース・コンシステンシ・モデル [7] に，田口と荒木が提案した Z 記述技法 [3] と value-passing CCS [4] (Calculus of Communicating Systems) を統合した同様の形式的技法 [5] を適用して，そのモデルと同期に関する実現の形式的仕様を記述した．そして，記述された共有メモリシステムの同期に関する実行トレース例を示した．状態を Z のキューと関数で記述して，それらの状態を持つ操作スキーマの実行順序を，value-passing CCS で記述した結果，複雑な同期処理においても，この形式技法を用いて柔軟に記述できることを確認した．

今後の課題としては，次の 2 点が挙げられる．まず，このモデルと実現の検証およびその他のコンシステンシ・モデル，特に，本稿で省略したプロセッサ・コンシステンシ・モデルなどを，この形式的技法で記述・検証すること．次に，これらの経験を踏まえて，この形式的技法を拡張することなどが挙げられる．

参考文献

[1] 城 和貴. メモリ・コンシステンシ・モデルの諸定義と解釈例. In **並列処理シンポジウム** *JSPP'97* **(チュートリアル講演)**, pages 149–163. 情報処理学会, Sep 1997.

[2] 福田 晃. **並列オペレーティングシステム**. 並列処理シリーズ 7. コロナ社, 1997.

[3] J.Spivey. *The Z notation*. Prentice Hall, second edition, 1992.

[4] R.Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[5] Kenji Taguchi and Keijiro Araki. The State-based CCS Semantics for Concurrent Z Specification. In *Proc. of the 1st Int'l Conf. of Formal Engineering Methods*, pages 283–292. IEEE, November 1997.

[6] S. Takata, K. Taguchi, K. Joe, and A. Fukuda. Specification and Verification of Memory Consistency Models for Shared-Memory Multiprocessor Systems. In *Proc. of Int'l Conf. on Parallel and Distributed Processing Techniques and Applications*, volume 2, pages 923–930, July 1998.

[7] K. Gharachorloo, D. Lenoski, J.Laudon, P. Gibbson, A. Gupta, and J.Hennessy. Memory Consistency and Event Ordering in Scalable Shared-Memory Multiprocessors. In *Proc. 17th Ann. Int'l Symp. Computer Architecture*, pages 15–26. IEEE, 1990.

[8] Colin Stirling. Modal and temporal logic for processes. In *Logic for Concurrency*, number 1043 in LNCS, pages 149–237. Springer-Verlag, 1996.