プログラミング 11−13
(1997. 1. 23)

SCCS 動作式に対する unfold 変換による LTS モデルの効率的な構成法

鈴木 晃　結縁 祥治　坂部 俊樹　稲垣 康善

名古屋大学 工学部 情報工学科

〒464-01 名古屋市千種区不老町

asuzuki@sakabe.nuie.nagoya-u.ac.jp
{yuen,sakabe,inagaki}@nuie.nagoya-u.ac.jp

あらまし

本稿では、SCCS 動作式に対する LTS モデル (ラベル付き遷移系モデル) を効率的に構成する方法を提案する。SCCS 動作式によって並行システムを仕様記述するとき、並列合成演算子と遅延演算子の組み合わせによって、LTS モデルの非決定性が大きくなり状態遷移数が増加する。しかし、実際的な仕様記述は部分仕様の組み合わせとして $(P_1|\cdots|P_n)\backslash L$ のような形をとり、このような仕様記述では部分仕様 P_i の並列合成による非決定性は制限演算 $\backslash L$ によって限定されることが多い。このためまず $P_1|\cdots|P_n$ の LTS モデルを構成してから全体の LTS モデルを構成すると、中間結果として求めた多くの遷移先の状態が $\backslash L$ により捨てられるため無駄になる。本稿では、この点に注目し、SCCS 動作式をあらかじめ構文的に変換して LTS モデルを構成する際の中間結果を小さくする方法を与え、この変換が SCCS の強等価性に対して健全であることを示す。さらに実験により、本方法が SCCS における検証に対して有効であることを示す。

キーワード： SCCS , LTS モデル , unfold 変換 , 並行プロセス

A Memory-efficient Contruction of Labeled Transition Models for SCCS Expressions by Unfold Transformation

Akira Suzuki　Shoji Yuen　Toshiki Sakabe　Yasuyoshi Inagaki

Department of Information Engineering,
Nagoya University
Furo-cho,Chikusa-ku,Nagoya,464-01,Japan

Abstract

In this paper, we present a memory-efficient construction of LTS models (labeled transition system models) for SCCS expressions. In a construction of an LTS model, the combination of parallel compositions (|) and delay operators ($) will increase the number of states and state transitions. For this reason, a large amount of memory may be required to check if an SCCS specification satisfies some property.

However, a typical SCCS specification is in the form of $(P_1|\cdots|P_n)\backslash L$ where $\backslash L$ internalizes the communications of L between P_i's. In this type of specification, it is often the case that the specification is deterministic externally although it is very nondeterministic internally because the restriction operation $\backslash L$ eliminates the internal nondeterminism that may cause the memory explosion by intermediate results. Based on this observation, we will give a transformation technique up to the strong equivalence of SCCS so that intermediate results may not be too large in constructing LTS models, where the SCCS expression is syntactically transformed beforehand. The transformation is given as the combination of process definition unfolding, expansion by the expansion theorem, and transformation by the strong equivalence. We show the impact of our transformation technique by presenting an example using Concurrency Workbench.

key words: SCCS , LTS model , unfold transformastion , parallel process

1 はじめに

SCCS(Synchronous CCS)[1] は、プロセスの並行動作において、すべてのイベントは同期するという原理を CCS[2] に導入した並行システム記述の体系である。SCCS によってモデル化される並行システムには大域的な同期を仮定できるのでハードウェアや低レベルの並行システムを記述できる [4],[5]。実際には動作をせず、単に同期するだけのアイドル動作を導入することにより、非同期的な並行システムも記述できる。SCCS に CCS と同様な代数的等価性が定義されている。SCCS 動作式の操作的意味は、LTS(ラベル付き遷移系) でモデル化され、LTS の等価性を定めることで代数的意味論が与えられている。

SCCS による典型的な仕様記述は、まず並列に動作する構成要素 P_i を SCCS で記述し、構成要素を並列合成演算子式 $|$ を用いて $P_1|\cdots|P_n$ のように合成することで与えられる。各構成要素は遅延演算子によって、非同期的に動作し、構成要素間の同期通信を内部化するため、制限演算子を適用する。このような仕様記述は、SCCS 動作式としては $(P_1|\cdots|P_n)\backslash L$ という形となる。仕様記述の検証は、動作式に対する LTS モデルを構成し、LTS モデルが性質を満たすかどうかを示すことで行なわれる。SCCS に対する検証ツール Concurrency Workbench[3] では、LTS モデルの構成が、まず $(P_1|\cdots|P_n)\backslash L$ の内側の式 $P_1|\cdots|P_n$ の状態遷移を求め、次に $(P_1|\cdots|P_n)\backslash L$ の状態遷移を内部の式の状態遷移を削除することにより行われる。

しかし、このような方法で LTS モデルを求めると、$P_1|\cdots|P_n$ の状態遷移が並列合成演算子の数により指数的に増大するため、多くのメモリを必要とし、コストが大きくなる。しかし、現実の仕様は、それほど非同期ではなく、非決定的な状態遷移のほとんどが制限演算子により削除されるため、最終的に必要な状態遷移は少なくなる。ここで $P_1|\cdots|P_n$ から制限演算子 $\backslash L$ によって制限されない動作をあらかじめ抽出しておけば、より少ないメモリで LTS モデルを構成することが可能である。

本稿では、与えられた SCCS 動作式を LTS モデル構成の際に中間結果があまり大きくならないような SCCS 動作式に変換する方法を与える。SCCS 動作式 $P\backslash L$ の内側の式 P より状態遷移数が少ない P' で $P\backslash L \sim P'\backslash L$ であるような P' を用いて計算を進めたほうが、$\backslash L$ で制限される遷移が少ないので時間、メモリ使用量の点で効率的である。このような P' を求めるために、SCCS 動作式におけるプロセス定数定義の unfold 操作, 展開定理, および SCCS の強等価関係に基づいた置き換えを行なう unfold 変換を提案する。この変換は、強等価性に関して健全であることが示される。この事実に基づいて、LTS モデルの構成におけるメモリ使用量、実行時間を Concurrency Workbench を用いて、評価する。その結果から、この unfold 変換の有効性を示す。さらに unfold 変換が適用できるクラスについても考察を行う。

本稿の構成は、2 章では SCCS を概観し、3 章では unfold 変換のアルゴリズムを与え、4 章では、unfold 変換が強等価性に関して健全であることを証明する。5 章では具体的な例に対して unfold 変換を適用し、6 章では変換方法に関して議論する。7 章でまとめと今後の課題について述べる。

2 SCCS

2.1 動作群

定義 2.1 (動作群) 動作名の集合 $\mathcal{A}$ が与えられるとき、相補動作名の集合を $\overline{\mathcal{A}} = \{\bar{\alpha} \mid \alpha \in \mathcal{A}\}$ とし、ここで $\mathcal{A} \cap \overline{\mathcal{A}} = \phi$ とする。$\mathcal{L} = \mathcal{A} \cup \overline{\mathcal{A}}$ とし、$\mathcal{L}$ 上の二項演算子 $\#$ は結合則、可換則を満たす。Act^+ を $\mathcal{A}$ から $\#$ と $^-$ によって自由に生成される集合とし、$Act = Act^+ \cup \{1\}$ とする。ここで、$1 \notin Act^+$ であり、任意の $a \in Act^+$ に対して $a = a\#1 = 1\#a$ および $1\#1 = 1$ とする。相補演算子は、$\#$ を保存し、$\bar{\bar{a}} = a, a\#\bar{a} = 1$, となるように Act 上に一意に拡張できる。Act 上で演算子 $\#, ^-$ と単位元 1 から構成される群構造 $(Act, \#, 1, ^-)$ を**動作群**とよぶ。 ■

ここでは、n 個の動作名 α の合成 $\alpha\#\cdots\#\alpha$ を α^n と、n 個の相補動作名 $\bar{\alpha}$ の合成 $\bar{\alpha}\#\cdots\#\bar{\alpha}$ は α^{-n} と略記する。任意の $a \in ACT^+$ は $\alpha_1^{z_1}\#\alpha_2^{z_2}\#\cdots\#\alpha_1^{z_n}(z_i \neq 0)$ と表現される。動作群において $\alpha_1\#\alpha_2$ は、動作名 α_1 と α_2 が同期して起こること、単位元 1 は、アイドリング動作、逆元のとの積 $\alpha\#\bar{\alpha} = 1$ は動作名 α と $\bar{\alpha}$ が通信することを表す。

2.2 構文と意味論

定義 2.2 (SCCS の構文と意味論) 動作群 $(Act, \#, 1, ^-)$ とプロセス定数集合 $\mathcal{K}$ が与えられたとき *SCCS* 動作式を次のように *BNF* で与える。

$$P ::= 0 \mid a:P \mid P+P \mid P \mid P \mid \$P \mid P[\phi] \mid K \mid P\backslash L$$

ここで $a \in Act, L \subseteq \mathcal{A}, K \in \mathcal{K}$(プロセス定数の集合) とする. 各 K には、プロセス定数定義 $K \stackrel{def}{=} P$ が存在するとする。

SCCS 動作式の動作的意味論 [1] は (表1) の *SOS* で表わされる。ϕ はラベル換えを表す写像 $\phi : Act \to Act$ であり、次の条件 $(i), (ii), (iii)$ を満たす。
$(i)\phi(1) = 1, (ii)$ 任意の $a, b \in \mathcal{A}$ に対して $\phi(a) = b$ ならば $\phi(\bar{a}) = \bar{b}$, (iii) $\phi(a\#b) = \phi(a)\#\phi(b)$.

動作プレフィックス	$\dfrac{}{a : P \xrightarrow{a} P}$	選択	$\dfrac{P_1 \xrightarrow{a} P'}{P_1 + P_2 \xrightarrow{a} P'}$ $\dfrac{P_2 \xrightarrow{a} P'}{P_1 + P_2 \xrightarrow{a} P'}$
並列合成	$\dfrac{P_1 \xrightarrow{a} P_1' \; P_2 \xrightarrow{b} P_2'}{P_1 \mid P_2 \xrightarrow{a\#b} P_1' \mid P_2'}$	遅延	$\dfrac{}{\$P \xrightarrow{1} \$P}$ $\dfrac{P \xrightarrow{a} P'}{\$P \xrightarrow{a} P'}$
ラベル換え	$\dfrac{P \xrightarrow{a} P'}{P[\phi] \xrightarrow{\phi(a)} P'[\phi]}$	定数	$\dfrac{P \xrightarrow{a} P'}{K \xrightarrow{a} P'}(K \stackrel{def}{=} P)$
制限	$\dfrac{P \xrightarrow{a} P'}{P\backslash L \xrightarrow{a} P'\backslash L} \quad (a = p_1\#p_2\#\cdots\#p_n, \forall p_i \in L \cup \bar{L} \cup \{1\})$		

表 1: SCCS の意味論

2.3 ソート

SCCS 動作式が行う可能性のある動作名の集合をソートと呼び以下のように定義する [1]。

定義 2.3 (Sort) 動作 $a = p_1^{n_1}\#\cdots\#p_k^{n_k}$ とする。このとき、$Part(a) = \{p_1, \ldots, p_k\}$ とする。また、動作の集合 A に対して、$Part(A) = \{Part(a) | a \in A\}$ とする。*SCCS* 動作式 P のソート $Sort(P)$ は以下のように帰納的に定義される。

- $Sort(a : Q) = Part(a) \cup Sort(Q)$
- $Sort(Q_1 + Q_2) = Sort(Q_1) \cup Sort(Q_2)$
- $Sort(Q_1|Q_2) = Sort(Q_1) \cup Sort(Q_2)$
- $Sort(Q\backslash L) = Sort(Q) \cap L$
- $Sort(\$Q) = Sort(Q)$
- $Sort(Q[\phi]) = Part(\phi(Sort(Q)))$
- プロセス定数 P に対して、$P \stackrel{def}{=} Q$ のとき、$Sort(Q) \subseteq Sort(P)$ をみたす。

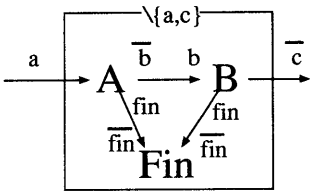
2.4 強等価関係と合同性

SCCS における強等価関係 $\sim$ は、合同関係であり、$\sim$ に関して以下の性質が成り立つことが知られている [1]。

$$P|0 \sim 0 \quad (1)$$
$$P + 0 \sim P \quad (2)$$
$$P + P \sim P \quad (3)$$
$$P|(Q + R) \sim P|Q + P|R \quad (4)$$
$$a : P|b : Q \sim a\#b : (P|Q) \quad (5)$$
$$(P + Q)\backslash L \sim P\backslash L + Q\backslash L \quad (6)$$
$$P\backslash L_1\backslash L_2 \sim P\backslash(L_1 \cap L_2) \quad (7)$$
$$(\$P)\backslash L \sim \$(P\backslash L) \quad (8)$$
$$\$P \sim P + 1 : \$P \quad (9)$$
$$\$P \sim P + \$P \quad (10)$$
$$\$P|\$Q \sim \$(P|\$Q + \$P|Q) \quad (11)$$
$$(a : P)\backslash L \sim \begin{cases} a : (P\backslash L) & if\ a = p_1\#\cdots\#p_n (\forall p_i \in L \cup \bar{L} \cup \{1\}) \\ 0 & otherwise \end{cases} \quad (12)$$
$$(P|Q)\backslash L \sim (P\backslash L)|(Q\backslash L)$$
$$if\ (Sort(P) \cap \overline{Sort(Q)}) \subseteq L \cup \bar{L}\ and\ (\overline{Sort(P)} \cap Sort(Q)) \subseteq L \cup \bar{L} \quad (13)$$

3 SCCS 動作式の unfold 変換

A は、ポート a からデータを受け取り、ポート $\bar{b}$ から結果を出力する。 B は、ポート b からデータを受け取り、ポート $\bar{c}$ から結果を出力しシステムは停止するようなシステムを表す。



$$AB \stackrel{def}{=} (A|B|Fin)\backslash\{a,c\} \quad (14)$$
$$A \stackrel{def}{=} \$a : \$\bar{b} : \$fin : 0 \quad (15)$$
$$B \stackrel{def}{=} \$b : \$\bar{c} : \$fin : 0 \quad (16)$$
$$Fin \stackrel{def}{=} \$\overline{fin}^2 : 0 \quad (17)$$

$$AB' \stackrel{def}{=} (A')\backslash\{a,c\} \quad (18)$$
$$A' \stackrel{def}{=} \$a : (\$\bar{b} : \$fin : 0|B') \quad (19)$$
$$B' \stackrel{def}{=} \$b : \$\bar{c} : (\$fin : 0|Fin) \quad (20)$$

図 1: A と B の構成

式 (14) の AB の中の B は、 A が動作 a を行なった時に、はじめて動作を起こすことができる。そのため、式 (18) の AB' では、 A' を式 (19) のように定義する。このとき、制限演算子で削除される B' の動作は計算されないが、AB' は AB と強等価になっている。図 2の LTS モデルにおいても、この変換により削除される状態遷移が少なくなっているのがわかる。 AB' は AB をプロセス定数定義を unfold して 2.4節による置き換えを行うことにより得られる。以下では、このような変換を **unfold 変換**として定式化する。

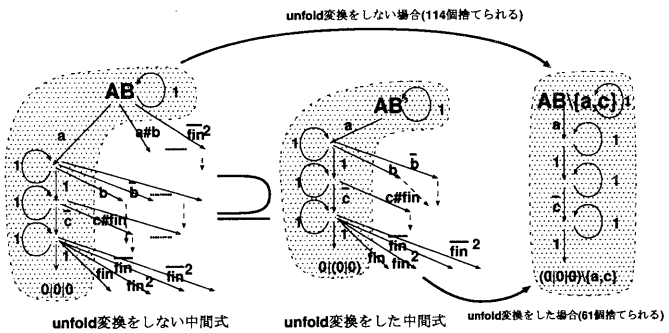


図 2: AB と AB' の LTS モデル

3.1 変換アルゴリズム

仕様として SCCS 動作式が

$$P \stackrel{def}{=} (P_1|P_2|\cdots|P_n)\backslash L \quad (21)$$

の形式であたえられたとする。これは、 SCCS による仕様記述の一般的な形式であり、妥当であると考えている。

- 入力:SCCS 動作式 P(式 21) を、次のように分解する。 Q は、 $P_1,\cdots,P_n$ の中で $P_i = \$a : P_i'$ の構造を持ち、動作 a を構成する動作名 p が制限演算子 $\backslash L$ により制限されており、唯一通信を行う相手 $P_j (i \neq j)$ をもつような P_i、 R は、 $P_1,\cdots,P_n$ から P_i を取り除いた $P_1|\cdots|P_{i-1}|P_{i+1}|\cdots|P_n$ とする。このとき、アルゴリズムの入力として与えるのは、 SCCS 動作式 Q,R, 動作名 p とする。関数の呼び出しは unfold(Q,R,p) の形式で行われる。

- 出力:SCCS 動作式を出力とする。 unfold(Q,R,p) の返り値 R' は、 $P \sim (Q|R)\backslash L \sim R'\backslash L$ の関係を満たす。

unfold 変換のアルゴリズムを (表 2) により与える。

```
SCCS unfold(SCCS Q, SCCS R, ActName p)
{
switch (outermost_operator(R))
        {
        case PARALLEL:
            R as R_1|R_2|...|R_n
            foreach i
                    S_i = Sort(R_i)
            if ∃ i((p̄ ∈ S_i)&&(∀j(j ≠ i) p̄ ∉ S_j))
                then
                  R'_i=unfold(Q, R_i, p)
                  return (R_1|...|R_{i-1}|R'_i|R_{i+1}|...|R_n)
                else
                  return (Q|R)
            break
        case CHOICE:
            R as R_1 + ... + R_n
            foreach i
                    R'_i=unfold(Q, R_i, p)
            return (R'_1 + ... + R'_n)
            break
        case PREFIX:
            R as b : R'
            b as q_1# ... #q_n
            if p̄ ∉ {q_1, ..., q_n}
              then
                    R''=unfold(Q, R', p)
                    return (b : R'')
              else return (Q|R)
            break
        case DEADLOCK:
            return (0)
            break
        case DELAY:
            Q as $a : Q'
            R as $R'
            R''=unfold(Q, R', p)
            return ($R'')
            break
        }
}
```

表 2: unfold 変換のアルゴリズム

4 強等価性に対する健全性

本節では、unfold 変換が強等価性に関して健全であることを証明する。

補題 **4.1** $P = (P_1|\cdots|P_n)\backslash L$, $Q = P_i, R = P_1|\cdots|P_{i-1}|P_{i+1}|\cdots|P_n$ であり、かつ $(Q|R)\backslash Rest \sim R'\backslash Rest$ $(L \subseteq Rest)$ であるならば $P \sim R'\backslash L$ である。

証明：$L \subseteq Rest$ であるので

$$P \sim (Q|R)\backslash L \sim (Q|R)\backslash Rest\backslash L$$

の等式が成り立つ。仮定より R' は $(Q|R)\backslash Rest \sim R'\backslash Rest$ を満たすので

$$\sim \quad (R'\backslash Rest)\backslash L$$
$$\sim \quad R'\backslash L$$

となり、$R'\backslash L$ と P は強等価である。 □

補題 **4.2** $Q = \$a : Q'$ の形をしており、動作 a の中の動作名 p は $\backslash L$ で許可されておらず、R と通信を行うとする。$Rest = ((Sort(Q) \cup Sort(R)) - \{p, \bar{p}\})$ かつ $R' = \mathrm{unfold}(Q, R, p)$ ならば $(Q|R)\backslash Rest \sim R'\backslash Rest$

証明：R の構造に関する帰納法で証明する。

- PARALLEL

  $R = R_1|\cdots|R_n$ とする。Q が動作名 p で通信を行う相手が $R_1, \cdots, R_n$ のうちの 1 つに定まらないときは、変換を行わないので強等価である。通信相手が 1 つに定まるときは、その通信相手を R_i とする。

  $p \in Sort(Q|R_i)$ かつ $p \notin Sort(R_1|\cdots|R_{i-1}|R_{i+1}|\cdots|R_n)$ であるので
  $(Sort(Q|R_i) \cap \overline{Sort(R_1|\cdots|R_{i-1}|R_{i+1}|\cdots|R_n)}) \subseteq Rest$ かつ
  $(\overline{Sort(Q|R_i)} \cap Sort(R_1|\cdots|R_{i-1}|R_{i+1}|\cdots|R_n)) \subseteq Rest$ である

  $$(Q|R_1|\cdots|R_n)\backslash Rest \quad \sim \quad ((Q|R_i)\backslash Rest|(R_1|\cdots|R_{i-1}|R_{i+1}|\cdots|R_n)\backslash Rest)$$

帰納法の仮定より R_i'=unfold(Q, R_i, p) は $(Q|R_i)\backslash Rest \sim R_i'\backslash Rest$ を満たす。そして、

$$\begin{aligned} &\sim ((R_i')\backslash Rest|(R_1|\cdots|R_{i-1}|R_{i+1}|\cdots|R_n)\backslash Rest) \\ &\sim (R_1|\cdots|R_{i-1}|R_i'|R_{i+1}|\cdots|R_n)\backslash Rest \end{aligned}$$

- CHOICE

$$\begin{aligned} (Q|(R_1+\cdots+R_n))\backslash Rest &\sim ((Q|R_1)+\cdots+(Q|R_n))\backslash Rest \\ &\sim (Q|R_1)\backslash Rest+\cdots+(Q|R_n)\backslash Rest \end{aligned}$$

帰納法の仮定より R_i'=unfold(Q, R_i, p) は $(Q|R_i)\backslash Rest \sim R_i'\backslash Rest$ を満たす。そして、

$$\begin{aligned} &\sim (R_1')\backslash Rest+\cdots+(R_n')\backslash Rest \\ &\sim (R_1'+\cdots+R_n')\backslash Rest \end{aligned}$$

- PREFIX

b as $q_1\#\cdots\#q_n$ であり、$Q \stackrel{def}{=} \$a:Q'$ という形式をとっているので $Q=1:Q+a:Q'$ と展開できる。

$$\begin{aligned} (Q|b:R')\backslash Rest &\sim ((1:Q|b:R')+(a:Q'|b:R'))\backslash Rest \\ &\sim (b:(Q|R')+a\#b:(Q'|R'))\backslash Rest \\ &\sim (b:(Q|R'))\backslash Rest+(a\#b:(Q'|R'))\backslash Rest \end{aligned}$$

動作 a は、禁止されている動作名 p を含むため $\backslash Rest$ によって実行できない。そして $\overline{p} \in \{q_1,\cdots,q_n\}$ ならば、変換を行わずに値を返すので強等価性は保存される。$\overline{p} \notin \{q_1,\cdots,q_n\}$ ならば、動作名 p と動作 b の中の動作名と通信が起きないので動作 $a\#b$ も実行できない。よって $(a\#b:(Q'|R'))\backslash Rest \sim 0$

$$\begin{aligned} &\sim (b:(Q|R'))\backslash Rest+0 \\ &\sim (b:(Q|R'))\backslash Rest \\ &\sim b:((Q|R')\backslash Rest) \end{aligned}$$

帰納法の仮定より R''=unfold(Q, R', p) は $(Q|R')\backslash Rest \sim R''\backslash Rest$ を満たすので

$$\begin{aligned} &\sim b:(R''\backslash Rest) \\ &\sim (b:R'')\backslash Rest \end{aligned}$$

- DEADLOCK

$$(Q|0)\backslash Rest \sim 0$$

- DELAY

R as $\$R'$ であるとして

$$\begin{aligned} (Q|R)\backslash Rest &\sim (\$a:Q'|\$R')\backslash Rest \\ &\sim (\$((\$a:Q')|R'+(a:Q')|\$R'))\backslash Rest \\ &\sim \$(((a:Q')|R'+(\$a:Q')|R'+(a:Q')|1:\$R'+(a:Q')|R')\backslash Rest) \\ &\sim \$(((a:Q')|R'+(\$a:Q')|R'+(a:Q')|1:\$R')\backslash Rest) \\ &\sim \$(((a:Q')|R')\backslash Rest+((\$a:Q')|R')\backslash Rest+((a:Q')|1:\$R')\backslash Rest) \end{aligned}$$

$a:Q' \stackrel{a}{\rightarrow} Q'$ であり $(a:Q'|1:\$R') \stackrel{a}{\rightarrow} Q'|\$R'$ となる。しかし動作 a を構成する動作名の p は $\backslash Rest$ により制限されるので $((a:Q')|1:\$R')\backslash Rest \sim 0$ となる。

$$\begin{aligned} &\sim \$((a:Q'|R')\backslash Rest+((\$a:Q')|R')\backslash Rest) \\ &\sim \$(((a:Q'+\$a:Q')|R')\backslash Rest) \\ &\sim \$((\$a:Q'|R')\backslash Rest) \\ &\sim \$((Q|R')\backslash Rest) \end{aligned}$$

帰納法の仮定より R''=unfold(Q,R',p) は $(Q|R')\backslash Rest \sim R''\backslash Rest$ を満たすので

$$\sim \$(R''\backslash Rest)$$
$$\sim (\$R'')\backslash Rest$$

□

定理 4.3 $P=(P_1|\cdots|P_n)\backslash L$, $Q=P_i$,$R=P_1|\cdots|P_{i-1}|P_{i+1}|\cdots|P_n$ であり、$Q=\$a:Q'$ の形をしており、動作 a の中の動作名 p は $\backslash L$ で許可されておらず、R と通信を行うとする。このとき $P\sim$unfold$(Q,R,p)\backslash L$ である

証明：仮定より $Q=\$a:Q'$ の形をしており、動作 a の中の動作名 p は $\backslash L$ で許可されておらず、R と通信を行うという条件を満たしている。$Rest=((Sort(Q)\cup Sort(R))-\{p,\overline{p}\})$ であるとすると、補題 4.2より $(Q|R)\backslash Rest$ $\sim$unfold$(Q,R,p)\backslash Rest$ が成り立つ。$Rest$ は、$L\subseteq Rest$ であるので、補題 4.1より $P\sim$ unfold$(Q,R,p)\backslash L$ が成り立つ。

□

5 変換例

本節では、Concurrency Workbench により、本稿の unfold 変換により実際に LTS モデルが効率良く構成されることを示す。本稿の unfold 変換を文献 [5] の SCCS 動作式によるコンパイラの仕様記述に対して適用し、unfold 変換をしない場合 $Compile$(表 3 (上) の式 30) と unfold 変換した場合 $Compile$(表 3 (下) の式 39) を求める。

$$Load\_t1\_va \stackrel{def}{=} \$('load\_t1\_va\#getr\#exec:1:T1\_both\_1) \quad (22)$$
$$Load\_t2\_vb \stackrel{def}{=} \$('load\_t2\_vb\#getr\#exec:1:T2\_both\_2) \quad (23)$$
$$Load\_t3\_vc \stackrel{def}{=} \$('load\_t3\_vc\#getr\#exec:1:T3\_both\_2) \quad (24)$$
$$T5\_Add\_t1\_t2 \stackrel{def}{=} \$(t1\#t2\#'add\_t5\_t1\_t2\#getr\#exec:T5\_reg\_1) \quad (25)$$
$$T6\_Sub\_t3\_t5 \stackrel{def}{=} \$(t3\#t5\#'sub\_t6\_t5\_t3\#getr\#exec:T6\_reg\_1) \quad (26)$$
$$T7\_Add\_t3\_t6 \stackrel{def}{=} \$(t3\#t6\#'add\_t7\_t3\_t6\#getr\#exec:T7\_reg\_1) \quad (27)$$
$$T8\_Sub\_t2\_t7 \stackrel{def}{=} \$(t2\#t7\#'sub\_t8\_t7\_t6\#getr\#exec:T8\_reg\_0) \quad (28)$$
$$ProgramSpec \stackrel{def}{=} Load\_t1\_va|Load\_t2\_vb|Load\_t3\_vc|T5\_Add\_t1\_t2 \quad (29)$$
$$|T6\_Sub\_t3\_t5|T7\_Add\_t3\_t6|T8\_Sub\_t2\_t7$$
$$Compile \stackrel{def}{=} (CPUSpec|ProgramSpec|Fin)\backslash Instr \quad (30)$$
$$(31)$$

$$Load\_t1\_va \stackrel{def}{=} \$('load\_t1\_va\#getr\#exec:1:(1\_both\_1|T5\_Add\_t1\_t2')) \quad (32)$$
$$Load\_t2\_vb \stackrel{def}{=} \$('load\_t2\_vb\#getr\#exec:1:T2\_both\_2) \quad (33)$$
$$Load\_t3\_vc \stackrel{def}{=} \$('load\_t3\_vc\#getr\#exec:1:T3\_both\_2) \quad (34)$$
$$T5\_Add\_t1\_t2' \stackrel{def}{=} \$(t1\#t2\#'add\_t5\_t1\_t2\#getr\#exec:(T5\_reg\_1|T6\_Sub\_t3\_t5')) \quad (35)$$
$$T6\_Sub\_t3\_t5' \stackrel{def}{=} \$(t3\#t5\#'sub\_t6\_t5\_t3\#getr\#exec:(T6\_reg\_1|T7\_Add\_t3\_t6')) \quad (36)$$
$$T7\_Add\_t3\_t6' \stackrel{def}{=} \$(t3\#t6\#'add\_t7\_t3\_t6\#getr\#exec:(T7\_reg\_1|T8\_Sub\_t2\_t7')) \quad (37)$$
$$T8\_Sub\_t2\_t7' \stackrel{def}{=} \$(t2\#t7\#'sub\_t8\_t7\_t6\#getr\#exec:T8\_reg\_0) \quad (38)$$
$$ProgramSpec' \stackrel{def}{=} Load\_t1\_va|Load\_t2\_vb|Load\_t3\_vc \quad (39)$$
$$Compile' \stackrel{def}{=} (CPUSpec|ProgramSpec'|Fin)\backslash Instr \quad (40)$$

表 3: unfold 変換前 (上)、unfold 変換後 (下)

	unfold 変換なし	unfold 変換あり
制限演算子の適用前	8.283e+08	6.271e+08
制限演算子の適用後	1.24e+05	1.25e+05
削除された状態遷移数	8.281e+08	6.269e+08

表 4: 制限演算子により無駄になった状態遷移の数

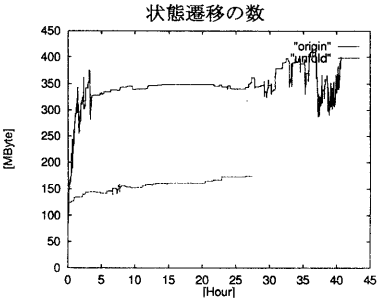


表 5: メモリ使用量と実行時間

次に Concurrency Workbench により LTS モデルを構成する。本実験では SGI Power Challenge(SPECint92 108.7, CPU R8000(75MHz) × 4 個, Memory 1GB) を使用した。Concurrency Workbench では、SCCS 動作式の定義が読みこまれても、LTS モデルが構成されない。このため、停止する遷移系列をすべて求める命令を実行させ LTS を構成し、命令の実行が完了するまでに無駄になった状態遷移を (表 4) に、メモリ使用量、実行時間を (表 5) に示す。

6 議論

この実験の結果より、unfold 変換により、削除される遷移が少なくなっており、それに伴い、メモリ使用量と実行にかかる時間が少なくなっていることがわかる。証明の中で等価関係 $P+0 \sim P$ に基づき SCCS 動作式 $P+0$ を P としていることが、LTS モデルの構成において、無駄な遷移を計算し、制限演算子 $\backslash L$ で削除することに対応している。unfold 変換により SCCS 動作式の段階で無駄な遷移を削除しているので、LTS モデルの構成の際に、メモリ使用量の効率が良くなっている。

制限演算子により、状態遷移が削除されるということは、並列合成された構成要素間で、通信の受信 (又は送信) 待ちが行われているということである。構成要素が依存関係を持つ SCCS 動作式は制限演算子により状態遷移を削除することにより得られる。この変換により、依存関係を持つ動作式では LTS モデル構成の効率を向上させることができる。構成要素間で通信がなく、独立している場合には、変換可能な条件をみたす Q、R、p が存在しない。

本稿の変換例では、変換アルゴリズムの入力 SCCS 動作式 Q,R と動作名 p を適切に選ばなければ、変換アルゴリズムが停止しない。これは、補題 4.2の CHOICE($R = R_1 + \cdots + R_n$) において、各 R_i の遷移の動作系列に動作名 $\bar{p}$ が含まれず、再び R となるような再帰的定義が 1 つでも存在する場合には、unfold(Q, R, p) が繰り返し呼び出されるからである。このため動作名 p, SCCS 動作式 Q,R を上手く選ぶ必要がある。

7 おわりに

SCCS 動作式に対する LTS モデルを効率的に構成するための SCCS 動作式の変換アルゴリズムを提案し、この変換は SCCS の強等価性に対して健全であることを示した。そして、Concurrency Workbench による具体的な評価を行った。5章の例は、各部分仕様が遅延演算子を含み、並列合成演算子が 8 個、部分仕様間に依存関係が存在する場合である。この例の場合では、unfold 変換によりメモリ使用量が約 50%、実行時間が約 70% になるという結果が得られた。これは並列合成演算子と制限演算子を用いた場合、無駄になる状態遷移の計算が、SCCS 動作式の段階で削除されるため、効率的に LTS モデルが構成できるからである。並列に合成される SCCS 動作式が強い依存関係をもつ場合には、本稿の unfold 変換により効率よく LTS モデルが構成できるようになる。結果として、状態遷移数が爆発してメモリ不足となってしまうような仕様記述 [5] の検証が可能になり、実用上かなりの改善がみられた。

今後の課題としては、本変換が部分仕様間に依存関係が存在する SCCS 動作式に対して有効であることが予想されているが、そのことを系統的に説明することが挙げられる。また、本変換への入力となる SCCS 動作式を適切に分解する方法は複数存在し、より効率的になるような分解を選択する方法を考察すること、また分解できる条件をより多くのクラスの SCCS 動作式に対して適用できるように緩和することも今後の課題である。本変換は Concurrency Workbench など検証ツールでの最適化の一方法であり、検証ツールへの直接的な組み込みについても検討する必要がある。

謝辞

本研究を進めるにあたり御討論いただいた名古屋大学情報工学科 稲垣・坂部研究室の皆様に感謝する。なお、本研究は一部、立松財団および文部省科学研究費補助金 (課題番号 08458066,0845867 および 08780260) からの補助をうけている。

参考文献

[1] Robin Milner,*Calculi for synchrony and asynchrony*, in Journal of Theoretical Computer Science,No.25:267-310,1983

[2] Robin Milner,*Communication and concurrency* ,Prentice-Hall,1989.

[3] R.Cleaveland,J.Parrow,and B.Steffen,*The Concurrency Workbench:A semantic based tool for verification of finite-state systems*, ACM Transaction on Programming Languages and Systems,15(1):36-72,January 1993

[4] Ed Harcout,Jon Mauney,Todd Cook,*High-Level Timing Specification of Instruction-Level Parallel Processors*,NCSU Tech Report TR-93-18,1993.

[5] 鈴木, 結縁, 坂部, 稲垣, 命令を並列に実行する *CPU* に対するコンパイラの仕様記述、電気情報通信学会 COMP 研 96-14(1996-05)