モード制約の漸近的一様補強による並行論理プログラムの occurs-check 解析

Occurs-Check Analysis of Concurrent Logic Programs through Asymptotic Uniform Augmentation of Mode Constraints

加藤 紀夫 † 上田 和紀 †

Norio KATO Kazunori UEDA

† 早稲田大学大学院理工学研究科

Graduate School of Information and Computer Science, Waseda University

n-kato@ueda.info.waseda.ac.jp

出現検査（occurs-check）は、単一化（unification）で無限構造を生成しないことを検査する処理であるが、効率が悪いため多くの論理型言語処理系で意図的に省略されてきた。しかしこの省略により、一階述語論理の証明器としての健全性が失われる、またはプログラム停止性の保証が困難になるなどの問題が生じており、省略の安全性を保証する解析技法が必要とされている。本研究では、入出力モード付きの並行論理プログラムに対して、特定の引数パス集合のモードが漸近的に一様となるようにモード制約が補強できることを検査することにより、プログラムが無限構造を生成しないことを静的に保証するアルゴリズムを開発した。これにより、従来は行えなかった循環依存的な双方向通信を行う並行論理プログラムに対する検査を可能した。

1 はじめに

1.1 背景

出現検査は単一化時に無限構造を作るかどうか調べる処理であり、論理型言語において処理系の一階述語論理証明器としての健全性を保証したり、プログラムの停止性を保証したりするために必要な検査である。たとえば単一化 `X=f(X)` は出現検査に合格しない。しかし実行時の出現検査は効率が悪いため、多くの処理系では明示的に指定されない限りこの検査を意図的に省略し、検査に合格しなくても単一化が成功するように決めている。そのため、上記の健全性および停止性の保証ができないばかりでなく、無限構造を扱うためのオーバーヘッドが処理系にかかる。そこで、静的な出現検査の研究が必要となる。

1.2 従来の研究

静的な出現検査には、occurs-check reduction と NSTO（Not Subject To Occurs-check）の 2 つのアプローチがある。前者はプログラム中の単一化処理のうちで出現検査が省略できるものを見つけ出すもので、無限構造を許しながら処理系の実行効率を高めることを目的とする。後者はプログラムの実行中に一切の無限構造が生成されないことを保証するもので、健全性および停止性の保証を目的とし、副次的に処理系の実行効率を高めることができる。[2]

関連研究の finite-tree analysis [1] は前者に属する手法で、抽象実行によりプログラム中のデータのうちで無限構造を持つ可能性が無いものを見つけ出し、出現検査が省略できることを保証する。しかしその抽象実行をゴールの中断および再開あるいは並行実行のある一般の論理型言語に適用することは難しい。

そのような言語では、ゴール選択順序に依存しない解析が必要となるため、抽象実行ではなく入出力モード情報というプログラムの静的な情報を使う後者のアプローチが取られる。しかし従来の研究では、引数トップレベルのモードしか活用しなかったため、図 2 のような双方向通信するプログラムは無限構造を作らないこと（NSTO）を保証できなかった。[3, 5]

1.3 本研究の内容

本研究は、入出力モード付きの並行論理プログラムに対して、モード制約が漸近的に一様補強できるか検査するという方法によって双方向通信も扱えるようにした、NSTO 性の保証アルゴリズムを提案する。

以下、2 節で NSTO 性を定式化し、3 節で NSTO 性を開始グラフに帰着させる。4 節でモード情報を使って開始グラフが無限構造を作らないことを検査する方法を示す。5 節で解析例を示し、6 節でまとめる。

2 問題の定式化

2.1 項

定義 2.1 一階述語論理の**変数**の集合 Var および**項**の可算集合 $Term$ および**関数記号**の集合 $Func$ が与えられているとする。項 t に出現する変数の全体集合を $\mathsf{Vars}(t)$ と表記する。$\equiv$ は構文的等価性を表す。

定義 2.2 $Path \stackrel{\text{def}}{=} \{\langle f/n, i\rangle \mid f/n \in Func, i \in 1..n\}^*$ とし、その要素を**パス**と呼ぶ。ただし $m..n \stackrel{\text{def}}{=} \{m, m+1, \ldots, n\}$ とする。$p \bowtie q \stackrel{\text{def}}{\Leftrightarrow} p = \epsilon \vee q = \epsilon$ と定義する。ただし ϵ は長さ 0 のパスを表す。

定義 2.3 $\bot \notin Term$ とし、任意のパス p に対して関数 $\overrightarrow{p}\colon Term \to Term \cup \{\bot\}$ を次のように定義する。

$$\begin{array}{l} \overrightarrow{\epsilon}(t) \stackrel{\text{def}}{=} t \\ \overrightarrow{\langle f/n, i\rangle p}(f(t_1, \ldots, t_n)) \stackrel{\text{def}}{=} \overrightarrow{p}(t_i) \\ \overrightarrow{\langle f/n, i\rangle p}(t) \stackrel{\text{def}}{=} \bot \qquad \text{otherwise} \end{array}$$

2.2 制約

定義 2.4 変数 X およびパス p に対して、Xp を**パス変数**と呼ぶ。パス変数 Xp と Yq および関数記号 f/n に対して、$Xp = Yq$ を**等式制約**、$\mathrm{func}(Xp, f/n)$ を**具体化制約**、まとめて**制約**と呼ぶ。制約の集合 C と変数 X に対して、X を構文的に含む制約を全て C から除去して得られる集合を $\exists X(C)$ と表記する。□

例 2.1 $\exists \mathrm{Y}(\{\mathrm{X}{=}\mathrm{X}, \mathrm{X}{=}\mathrm{Y}, \mathrm{Y}{=}\mathrm{Z}\}) = \{\mathrm{X}{=}\mathrm{X}\}$

定義 2.5 次のように定義する。

$Sat(C) \stackrel{\text{def}}{=} \bigcup_{k=0}^{\infty} \mathcal{S}^k(ID \cup C\star)$,
$\mathcal{S}^0(C) \stackrel{\text{def}}{=} C, \quad \forall k \geq 0(\mathcal{S}^{k+1}(C) \stackrel{\text{def}}{=} \mathcal{S}(\mathcal{S}^k(C)))$,
$ID \stackrel{\text{def}}{=} \{Xp = Xp \mid X \in Var, p \in Path\}$,
$C\star \stackrel{\text{def}}{=} \{Yq = Xp \mid (Xp = Yq) \in C\} \cup C$,
$\mathcal{S}(C) \stackrel{\text{def}}{=} C$
$\cup \{Xps = Zrt \mid (Xp{=}Yqt), (Yqs{=}Zr) \in C, s \bowtie t\}$
$\cup \{\mathrm{func}(Yq, f/n) \mid \mathrm{func}(Xp, f/n), (Xp{=}Yq) \in C\}$,
$\mathrm{fin}(C) \stackrel{\text{def}}{\Leftrightarrow} \neg\exists X \exists p \exists q \neq \epsilon((Xp = Xpq) \in Sat(C))$

2.3 プログラム言語

定義 2.6 $n \geq 0$ のとき、項 $h, b_1, \ldots, b_n$ に対して、$(h \leftarrow b_1, \ldots, b_n)$ を**節**と呼ぶ。節の有限集合[1]を**プログラム**と呼ぶ。また $\mathrm{main} \in Term \setminus Var$ とする。

定義 2.7 $n \geq 1$ のとき、正整数の列 $i_1, \ldots, i_n$ に対して、$i_1.i_2.\ldots.i_n$ を**ゴール識別子**、$\overline{i_1.i_2.\ldots.i_n}$ を**ヘッド識別子**、まとめて**ノード識別子**と呼ぶ。ゴール識別子 w に対して $\overline{\overline{w}} \stackrel{\text{def}}{=} w$ と定義する。また、ノード識別子から変数への 1 対 1 写像 $u \mapsto \langle u\rangle$ が与えられているとする。□

[1]図 1 のアルゴリズムを修正すれば、必ずしも有限集合である必要は無くなり、無限個の節へのガード展開にも対応できる。

定義 2.8 プログラム $\boldsymbol{P}$ に対して、次の規則を満たす最小集合によって遷移系 $\longrightarrow$ を定義し、本研究が対象とする並行論理型言語の操作的意味とする。

(R1) $\langle \{w : a\} \cup Q, G\rangle$
$\longrightarrow \langle \{w.i : b_i \mid i \in 1..n\} \cup Q, G'\rangle$
where $c = (h \leftarrow b_1, \ldots, b_n) \in \boldsymbol{P}$,
$G' = G \cup \mathcal{G}(w, a, c)$,
$Sat(\exists\langle\overline{w}\rangle\exists\langle w.1\rangle \ldots \exists\langle w.n\rangle(Sat(G'))) = Sat(G)$

(R2) $\langle \{w : (s =_k t)\} \cup Q, G\rangle$
$\longrightarrow \langle Q, G \cup \mathcal{G}(w, (s =_k t), (U_k \leftarrow))\rangle$

ただし

$\mathcal{G}(w, a, (h \leftarrow b_1, \ldots, b_n))$
$\stackrel{\text{def}}{=} \{\langle w\rangle pqr = \langle\overline{w}\rangle pqr \mid \overrightarrow{pq}(a), \overrightarrow{pr}(h) \in Var, q \bowtie r\}\star$
$\cup \{\langle\overline{w}\rangle\langle{=_k}/2, 1\rangle = \langle\overline{w}\rangle\langle{=_k}/2, 2\rangle \mid h \equiv U_k\}\star$
$\cup \{\langle w.i\rangle p = \langle\overline{w}\rangle q \mid \overrightarrow{p}(b_i) \equiv \overrightarrow{q}(h) \in Var\}\star$
$\cup (\{\langle w.i\rangle p = \langle w.j\rangle q \mid \overrightarrow{p}(b_i) \equiv \overrightarrow{q}(b_j) \in Var\} \setminus ID)$

とし、任意の $k \geq 0$ に対して、$U_k \stackrel{\text{def}}{=} (\mathrm{X} =_k \mathrm{X})$ とする。ただし ${=_k}/2 \in Func$, $\mathrm{X} \in Var$ を仮定する。□

$\mathcal{G}(w, a, c)$ は、ゴール $w : a$ の節 c による書き換えで生成される等式制約集合を定義している。第 1 項はリダクション時のマッチングを表す。第 2–4 項は節内での変数の複数出現を表す。なお (R1) の最後の等式は、マッチングが読み取り専用であることを表す。

定義 2.9 (NSTO) プログラム $\boldsymbol{P}$ は、次の式を満たすときそのときに限り **NSTO** であると定義する。
$\forall Q, G(\langle\{1 : \mathrm{main}\}, \{\}\rangle \longrightarrow^* \langle Q, G\rangle \;\Rightarrow\; \mathrm{fin}(G))$

3 開始グラフへの帰着

3.1 サイクル

定義 3.1 プログラム $\boldsymbol{P}$ が与えられたとする。$n \geq 0$, 任意の項 $a_1, \ldots, a_n$, 節 $c_1, \ldots, c_n \in \boldsymbol{P}$ および互いに異なるゴール識別子 $w_1, \ldots, w_n$ に対して、等式制約集合 $\bigcup_{i=1}^{n} \mathcal{G}(w_i, a_i, c_i)$ を**接続グラフ**と呼ぶ。□

定義 3.2 $u_1, \ldots, u_n$ はノード識別子とする。等式制約とパスの組の列 $H = (\langle\langle u_{i-1}\rangle q_{i-1} = \langle\overline{u_i}\rangle p_i; s_i r_i\rangle \langle\langle\overline{u_i}\rangle p_i s_i = \langle u_i\rangle q_i t_i; r_i\rangle)_{i=1}^{n}$ は、以下の (1)–(4) を全て満たすときそのときに限り**サイクル**と定義す

る。(1) $n \geq 1$ かつ $\langle u_0\rangle q_0 \equiv \langle u_n\rangle q_n$ (2) $\forall i \in 2..n(t_{i-1}r_{i-1} = s_i r_i)$ (2′) $\exists q(t_n r_n = s_1 r_1 q)$ (3) $\forall i \in 1..n(p_i s_i = q_i t_i)$ (3′) $\forall i \in 1..n(\langle u_{i-1}\rangle q_{i-1} \not\equiv \langle u_i\rangle p_i)$ (4) $H' = \{\langle u_{i-1}\rangle q_{i-1} = \langle \overline{u_i}\rangle p_i \,|\, i \in 1..n\} \cup \{\langle \overline{u_i}\rangle p_i s_i = \langle u_i\rangle q_i t_i \,|\, i \in 1..n\}$ とおくと、H' はある接続グラフの部分集合。

今後、H 自身により上記の集合 H' を表してよいことにする。$\langle u_0\rangle q_0 s_1 r_1 = \langle u_n\rangle q_n t_n r_n$ を H の**目的等式制約**と呼ぶ。$L(H) \stackrel{\text{def}}{=} \{\langle u_{i-1}\rangle q_{i-1} \,|\, i \in 1..n\}$ および $R(H) \stackrel{\text{def}}{=} \{\langle \overline{u_i}\rangle p_i \,|\, i \in 1..n\}$ と定義する。 □

H の各 i の前半は節内での変数の複数出現に対応する単一化を表し、後半はリダクション時のマッチングに対応する単一化を表す。サイクルは、プログラム $\boldsymbol{P}$ が与えられたときに定義されることに注意する。

定理 3.1 (サイクルと無限構造の関係) サイクルが存在しないならば、プログラム $\boldsymbol{P}$ は NSTO である。

証明 接続グラフ G に対して、$\neg\mathrm{fin}(G)$ ならばある $q \neq \epsilon$ なる $(Xp = Xpq) \in Sat(G)$ が存在して $Xp = Xpq$ を目的等式制約とするサイクルが存在する。 □

3.2 開始グラフ

定義 3.3 ゴール識別子 w および節 $c \in \boldsymbol{P}$ に対して、$\mathcal{T}(w,c) \stackrel{\text{def}}{=} \exists\langle\overline{w}\rangle(\mathcal{G}(w,\mathrm{main},c))$ を**開始グラフ**と呼ぶ。($\mathcal{G}$ の第 4 項に相当する。main 以外の項も可) □

補題 3.1 任意のサイクル H に対してある w,cが存在して $\forall a \in Term(\{\} \neq H \cap \mathcal{G}(w,a,c) = H \cap \mathcal{T}(w,c))$ が成り立つ。$H \cap \mathcal{T}(w,c)$ を H の**開始部分**と呼ぶ。 □

定義 3.4 開始グラフの部分集合 T に対して、T の部分集合を開始部分とするようなサイクルが存在しないときそのときに限り、$\mathrm{fingen}(T)$ と表記する。 □

定理 3.2 $\forall w,c(\mathrm{fingen}(\mathcal{T}(w,c)))$ ならば、プログラム $\boldsymbol{P}$ は NSTO である。 □

4 モード付け関数の利用

4.1 モード付け関数

定義 4.1 記号 in および out を**モード**と呼ぶ。$Path$ から $\{in,out\}$ への関数を**モード付け関数**と呼ぶ。モード付け関数 m およびパス p に対して、$(m/p)(q) \stackrel{\text{def}}{=} m(pq)$ によりモード付け関数 m/p を定義する。$\forall p \in Path(\{m(p),\overline{m}(p)\} = \{in,out\})$ を満たすようにモード付け関数 $\overline{m}$ を定義する。 □

定義 4.2 モード付け関数 m の値を制約する論理式を**モード制約**[2]と呼ぶ。節 $(h \leftarrow b_1,\ldots,b_n)$ は、以下のモード制約を課すると考えることができる。

(HF) $\forall p(\overrightarrow{p}(h) \in Term \setminus Var \Rightarrow m(p) = in)$
(BF) $\forall p(\overrightarrow{p}(b_i) \in Term \setminus Var \Rightarrow m(p) = in)$
(BU) $m/\langle =_k,1\rangle = \overline{m/\langle =_k,2\rangle}$ if $h \equiv U_k$
(BV_+) $\forall X \in \mathsf{Vars}(h)\,\forall p(\overrightarrow{p}(h) \equiv X \Rightarrow \mathcal{R}(\{\!\{\overline{m/p}\}\!\} + \{\!\{m/q \,|\, \overrightarrow{q}(b_i) \equiv X\}\!\}))$
(BV_0) $\forall X \in \mathsf{Vars}(b_1,\ldots,b_n) \setminus \mathsf{Vars}(h)\ \mathcal{R}(\{\!\{m/q \,|\, \overrightarrow{q}(b_i) \equiv X\}\!\})$

ただし $\{\!\{\ldots\}\!\}$ は多重集合を表す記法とし、

$$\mathcal{R}(M) \stackrel{\text{def}}{\Leftrightarrow} \forall p \in Path\ \exists m \in M(m(p) = out \wedge \forall m' \in M - \{\!\{m\}\!\}(m'(p) = in))$$

とする。

プログラム $\boldsymbol{P}$ に対して、$\boldsymbol{P}$ の各節が課するモード制約の全体集合を $\mathcal{M}(\boldsymbol{P})$ と表記する。モード制約の集合 E に対して、E を満たすモード付け関数 m の全体集合を $M(E)$ と表記する。$M(\mathcal{M}(\boldsymbol{P})) \neq \{\}$ を満たす $\boldsymbol{P}$ は**モード付き**であると言う。[4] □

4.2 漸近的に等しいモード付け関数

定義 4.3 $\forall p \in Path(OUT(p) = out)$ および $IN \stackrel{\text{def}}{=} \overline{OUT}$ により、モード付け関数 IN, OUT を定義する。IN と OUT を**一様な**モード付け関数と呼ぶ。 □

定義 4.4 モード付け関数 m および u に対して、$\forall q\exists r(m/qr = u)$ が成り立つときそのときに限り、m は u に**収束する**と定義し、$m \to u$ と表記する。 □

定義 4.5 モード付け関数 m および m' に対して、$\exists u \in \{IN,OUT\}(m \to u \wedge m' \to u)$ が成り立つときそのときに限り、m と m' は**漸近的に等しい**と定義し、$m \simeq m'$ と表記する。 □

4.3 モード付けと無限構造の関係

定義 4.6 ゴール識別子 w とモード付け関数 m に対して、$I(w)(m) \stackrel{\text{def}}{=} m$ および $I(\overline{w})(m) \stackrel{\text{def}}{=} \overline{m}$ と定義する。また $G = \{\langle u_{2k-1}\rangle p_{2k-1} = \langle u_{2k}\rangle p_{2k} \,|\, k \in 1..n\}$ と変数 X に対して、$Nodes(G) \stackrel{\text{def}}{=} \{u_i \,|\, i \in 1..2n\}$ および $G^X \stackrel{\text{def}}{=} \{p_i \,|\, \langle u_i\rangle \equiv X\}$ と定義する。 □

補題 4.1 サイクル H に対して、次の式が成り立つ。

$$\forall\langle u'\rangle p' \in L(H)\,\forall\langle u''\rangle p'' \in R(H)\ \forall m \in M(\mathcal{M}(\boldsymbol{P}))(I(u')(m/p') \not\equiv I(u'')(m/p''))$$

[2]定義 2.4 で定義された制約とは無関係である。

証明 サイクルを $H = (\langle\langle u_{i-1}\rangle q_{i-1} = \langle\overline{u_i}\rangle p_i; s_i r_i\rangle \langle\langle\overline{u_i}\rangle p_i s_i = \langle u_i\rangle q_i t_i; r_i\rangle)_{i=1}^{n}$ および $\langle u_n\rangle q_n t_n r_n \equiv \langle u_0\rangle q_0 s_1 r_1 q$ とし、$\langle u'\rangle p' \in L(H)$, $\langle u''\rangle p'' \in R(H)$, $m \in M(\mathcal{M}(\boldsymbol{P}))$ を任意に選ぶ。$\langle u'\rangle p' \equiv \langle u_{j-1}\rangle q_{j-1}$, $\langle u''\rangle p'' \equiv \langle\overline{u_k}\rangle q_k$ とおける。$m' = I(u')(m)$, $m'' = I(u'')(m)$, $q' = s_j r_j$, $q'' = s_k r_k$ とおく。

サイクルおよび $\mathcal{G}$ と $\mathcal{M}$ の定義より、$\forall i \in 1..n$ $\forall p(I(u_{i-1})(m)(q_{i-1}p) = in \vee I(\overline{u_i})(m)(p_i p) = in)$ が成り立つ。したがって任意の $r \in Path$ に対して、$\forall i \in j..n(I(\overline{u_i})(m)(p_i s_i r_i r) = out)$ かつ $\forall i \in 1..k(I(\overline{u_i})(m)(p_i s_i r_i qr) = out)$ ならば $in = m'(p'q'r) \neq m''(p''q''qr) = out$ となり、さもなくば $out = m'(p'q'qqr) \neq m''(p''q''qqqr) = in$ となる。

かりに $m'/p' \simeq m''/p''$ と仮定すると、ある $u \in \{IN, OUT\}$ およびあるパス s, s', s'', s''' が存在して $m'/p'q's = m''/p''q''qss' = m'/p'q'qqss's'' = m''/p''q''qqqss's''s''' = u$ とおけるが、$r = ss's''s'''$ とおくと $m'/p'q'r = m''/p''q''qr$ かつ $m'/p'q'qqr = m''/p''q''qqqr$ となり、前段落と矛盾する。 □

補題 4.2 開始グラフの部分集合 T が与えられたとする。任意の $w \in Nodes(T)$ に対して次が成り立つ。

$$(\ \exists m \in M(\mathcal{M}(\boldsymbol{P}))\,\forall p, p' \in T^{\langle w\rangle}(m/p \simeq m/p') \wedge \mathrm{fingen}(\exists\langle w\rangle(T))\ ) \Rightarrow \mathrm{fingen}(T)$$

証明 $m \in M(\mathcal{M}(\boldsymbol{P}))$, $\forall p, p' \in T^{\langle w\rangle}(m/p \simeq m/p')$ かつ $\neg\mathrm{fingen}(T)$ とする。$\neg\mathrm{fingen}(\exists\langle w\rangle(T))$ を示す。

T の部分集合を開始部分とする任意のサイクルを H とする。かりに $p \in T^{\langle w\rangle}$ が存在すると仮定すると $\langle w\rangle p \in R(H) \cup L(H)$ となる。リダクションの履歴が木構造を成すという性質から、ある p', p'' が存在して $\langle w\rangle p' \in L(H)$ かつ $\langle w\rangle p'' \in R(H)$ となる。しかし補題 4.1 により $m/p' \not\simeq m/p''$ となり、矛盾する。よって $T^{\langle w\rangle} = \{\}$ より $\exists\langle w\rangle(T) = T$ が成り立つ。□

定理 4.1 プログラム $\boldsymbol{P}$ は、図 1 のアルゴリズムを実行して NSTO であると出力すれば NSTO である。

証明 NSTO と出力したときは、補題 4.2 および $\mathrm{fingen}(\{\})$ より、$\forall w, c(\mathrm{fingen}(\mathcal{T}(w, c)))$ が成り立つ。定理 3.2 より $\boldsymbol{P}$ は NSTO である。 □

5 解析例

図 2 の GHC (Guarded Horn Clauses) プログラムを使い、本手法による解析の例を示す。節中の | より左側は**ガード**と呼ばれ、その節の適用条件を表す。

Input: a program $\boldsymbol{P}$;
$E \leftarrow \mathcal{M}(\boldsymbol{P})$;
for each $c \in \boldsymbol{P}$ **do**
  $T \leftarrow \mathcal{T}(1, c)$;
  repeat
    $W \leftarrow Nodes(T)$;
    for each $w \in W$ **do**
      $E' \leftarrow \{m/p \simeq m/p' \mid p, p' \in T^{\langle w\rangle}\}$;
      if $E \cup E'$ is consistent **then**
        $E \leftarrow E \cup E'$; $T \leftarrow \exists\langle w\rangle(T)$
      end if
    end for
  until $W = Nodes(T)$;
  if $W \neq \{\}$ **return** 'Analysis failed'
end for;
return '$\boldsymbol{P}$ is NSTO'

図 1: Occurs-check アルゴリズム

解析を行うには、ガードの展開（節の正規化 [4]）またはボディ化により、ガードを除去する必要がある。ボディの組み込み述語は、呼び出しごとに異なる述語記号を持たせ、モード多相述語として扱う。以上の変換のあと、図 1 のアルゴリズムを適用する。

前処理 図 2 のプログラムに上記の前処理を行った後、モード制約を生成、簡約化する。そのうち漸近的一様補強に関係がある制約を抜き出すと、図 3 のように表せる。次に各節の開始グラフを作る。開始グラフがボディでの変数複数出現間に弧を持つ無向グラフであることに注意すると、図 4 のように表せる。

アルゴリズムの適用 図 1 のアルゴリズムに従い、各節の開始グラフから全てのゴールを取り除く。

最初に節 `merge-1` を選ぶ。ゴール `2:`を選び、ある一様なモード付け関数 `U1` $\in \{IN, OUT\}$ を用意して `<merge,3> -> U1` という補強を試みる。`<merge,1> -> -U1` を含めて矛盾なく追加できるため `2:`が取り除ける。すると単独出現になった変数 `Ms1` も取り除けるのでグラフが空になり、`merge-1` は成功する。

上記の補強により、`merge-2` と `merge-3` はともに`2:`を選べばこれ以上の補強をせずに取り除ける。

次に節 `mul-1` に移る。すでに `<u4,1><.,1> -> U1` かつ `<u4,1><.,2> -> U1` なので `1:`が取り除ける。

```
main :- true | print(Hs0), Hs0 =0 [1|Hs1], mul(2,Hs0,L2), mul(3,Hs0,L3),
               mul(5,Hs0,L5), merge(L2,L3,L23), merge(L23,L5,Hs1).
merge([A|As1],[B|Bs1],Ms) :- A < B | Ms =1 [A|Ms1], merge(   As1 ,[B|Bs1],Ms1).
merge([A|As1],[B|Bs1],Ms) :- A > B | Ms =2 [B|Ms1], merge([A|As1],   Bs1 ,Ms1).
merge([A|As1],[B|Bs1],Ms) :- A=:=B | Ms =3 [A|Ms1], merge(   As1 ,   Bs1 ,Ms1).
mul(N,[A|As1],Ms)         :- true  | Ms =4 [M|Ms1], M :=5 N*A, mul(N,As1,Ms1).
```

図 2: 双方向通信する並行論理プログラム

すると単独出現になった M, Ms1 が取り除けて、グラフには変数 N のみが残る。つぎにある U2 $\in \{IN, OUT\}$ を用意すると <is5,2><*,1> -> U2 が矛盾なく追加できるので 2:が取り除けて成功する。

最後に main-1 に移る。1:を選び、ある U3 $\in \{IN, OUT\}$ を用意して <print,1> -> U3 を試みると、U3 = -U1 とすれば <u0,1><.,2> -> U1 に矛盾なく追加できるので 1:が取り除ける。次に 3:を選ぶと、<mul,2> = IN より <mul,3> -> IN と補強するしかないがこれは U1 = IN とすれば無矛盾なので3:とL2 が取り除け、そのまま4:とL3, 5:とL5, 6:とL23, 7:とHs1 そして2:の順に全て取り除ける。

以上により、このプログラムは NSTO である。□

```
IN = <mul,2>
IN = <is5,2><*,2>
<merge,1> =  <merge,1><.,2>
<merge,1> =  <merge,2>
<merge,1> = -<merge,3>
<merge,1> = -<mul,3>
<merge,1> = -<u0,1><.,2>
<merge,1> = -<u1,1> = <u1,2>
<merge,1> = -<u2,1> = <u2,2>
<merge,1> = -<u3,1> = <u3,2>
<merge,1> = -<u4,1> = <u4,2>
<merge,1><.,1> = -<is5,1>
<merge,1><.,1> = -<u5,1> = <u5,2>
<mul,1> = <is5,2><*,1>
<print,1> = -<u0,1> = <u0,2>
```

図 3: モード制約集合のテキスト表現

```
main-1::
    Hs0  1:<print,1>, 2:<u0,1>,
         3:<mul,2>, 4:<mul,2>, 5:<mul,2>
    Hs1  2:<u0,2><.,2>, 7:<merge,3>
    L2   3:<mul,3>, 6:<merge,1>
    L3   4:<mul,3>, 6:<merge,2>
    L5   5:<mul,3>, 7:<merge,2>
    L23  6:<merge,3>, 7:<merge,1>
merge-1:: Ms1 1:<u1,2><.,2>, 2:<merge,3>
merge-2:: Ms1 1:<u2,2><.,2>, 2:<merge,3>
merge-3:: Ms1 1:<u3,2><.,2>, 2:<merge,3>
mul-1::   M   1:<u4,2><.,1>, 2:<is5,1>
          Ms1 1:<u4,2><.,2>, 3:<mul,3>
          N   2:<is5,2><*,1>, 3:<mul,1>
is5-1::
```

図 4: 開始グラフのテキスト表現

6 まとめと今後の課題

入出力モード付きの並行論理プログラムに対して、モード制約が漸近的に一様補強できるか検査するという方法により新たに双方向通信も扱えるようにした、静的 occurs-check アルゴリズムを提案した。

今後の課題として、本手法がカバーするプログラムの範囲を調査する実験を行うことが重要であろう。

参考文献

[1] R. Bagnara, R. Gori, P. M. Hill, and E. Zaffanella: “Finite-Tree Analysis for Constraint Logic-Based Languages”, *Static Analysis: 8th Int. Symp.*, LNCS 2126, pp. 165–184, 2001.

[2] L. Crnogorac, A. D. Kelly, and H. Søndergaard: “A Comparison of Three Occur-Check Analysers”, *Static Analysis: 3rd Int. Symp.*, LNCS 1145, pp. 159–173, 1996.

[3] K. R. Apt and A. Pellegrini: “On the Occur-check Free Prolog Programs”, *ACM Trans. Programming Languages and Systems*, 16(3), pp. 687–726, 1994.

[4] K. Ueda and M. Morita, “Moded Flat GHC and Its Message Oriented Implementation Technique”, *New Generation Computing*, Vol. 11, pp. 3–43, 1993.

[5] 加藤紀夫, 加藤昇嗣, 上田和紀: “並行論理プログラムの静的 occur-checker の設計と実装”, 情報処理学会全国大会論文集 2V-05, 2000.