有限状態機械に基づくプログラミングでの goto 文使用の是非：Hoare 論理の観点から

金藤栄孝[†] 二木厚吉[††]

Dijkstra の **goto** 文有害説とそれに引き続く構造的プログラミングの提唱以降，**goto** 文の使用に関する問題は永く議論された．**goto** 文の使用法に関し理論的裏付けを持つ研究としては，逐次的プログラムでの任意の制御フローは順次接続・条件分岐・反復の 3 基本構造のみで表現可能であるという結果に基づく Mills らの **goto** 文排斥論以外は皆無である．Dijkstra 本来の正しさを示しやすいプログラムを書くための構造化という立場――つまりプログラム検証論の立場――からの **goto** 文使用の是非は考察されていない．本論文では検証手段としての Hoare 論理に基づき有限状態機械モデルに基づくプログラミングでの **goto** 文の使用を検討する．その結果，状態をラベルで表し状態遷移を **goto** 文での飛び越しで行うプログラミングスタイルが，状態を表す変数を追加し **goto** 文を除いたプログラミングスタイルと比べ，Hoare 論理による検証での表明が簡単で自然な形となり機械的検証の時間的コストも少ない．ゆえにプログラムの正しさの示しやすさという観点からは有限状態機械モデルに基づくプログラミングでの状態変数導入による **goto** 文除去は有害であり **goto** 文を用いたスタイルの方が望ましいことを示す．

On the Use of goto's in Programming Based on Finite State Machines: From the Hoare Logic Viewpoint

Hidetaka Kondoh[†] and Kokichi Futatsugi[††]

There have been a vast amount of debates on the issue on the use of **goto** statements initiated by the famous Dijkstra's Letter to the Editor of CACM and his proposal of "Structured Programming". Except for the **goto**-less programming style by Mills based on the fact that any control flows of sequential programs can be expressed by the sequential composition, the conditional (**if-then-else**) and the indefinite loop (**while**), there have not been, however, any scientific accounts on this issue from the Dijkstra's own viewpoint of verifiability of programs. In this work, we reconsider this issue from the viewpoint of Hoare Logic, the most standard framework for correctness-proving, and we see that the use of **goto**'s for expressing state transitions in programs designed with the finite state machine modelling can be justified from the Hoare Logic viewpoint by showing the fact that constructing the proof-outline of a program using **goto**'s for this purpose is easier than constructing the proof-outline of a Mills-style program without **goto** by introducing a new variable.

1. はじめに――本論文の動機

本論文が扱う問題は「**goto** 文有害説」と「構造的プログラミング」というきわめて古典的な主題に属する．かくも古い主題を改めて議論する理由は，著者の考えでは，これら主題が提唱者 Dijkstra 本来の意図に添った形ではいまだ十分に検討されていないことにある．

分かりやすいプログラムを書くことがソフトウェア工学的に大切なのは論をまたない．CACM の編集者への手紙の形で Dijkstra[5] はプログラムの分かりやすさに対する **goto** 文の悪影響を警告した．彼は，この手紙に引き続き提唱した[6),7)]「構造的プログラミング」で分かりやすいプログラムを書く目的はプログラムの正しさを示しやすくするためとし「プログラムを構造化するのが必要なことを，プログラムの正しさを証明できる必要があることの帰結であるとして提唱しました」（訳書[7)]，p.47）と述べ，最初の手紙での「プログラムの分かりやすさ」という漠然とした基準を「プログラムの正しさの示しやすさ」というより明確な基準へと洗練した．なお，**goto** 文有害説を述べた最初の

† 株式会社日立製作所システム開発研究所
Systems Development Laboratory, Hitachi, Ltd.
†† 北陸先端科学技術大学院大学
Japan Advanced Institute of Science and Technology (JAIST)

手紙の末尾で，Böhm ら[1] の仕事に代表される制御フローの表現力の等価性に基づく機械的な **goto** 文の除去はプログラムの分かりやすさの向上という——彼が **goto** 文の問題を提起した本来の——趣旨からは無意味である，と Dijkstra 自身がすでに指摘していることを注意しておく．**goto** 文有害説と構造的プログラミングに関するより詳細な歴史的流れと関連する文献リストについては著者らによる別稿[12] の 1 章を参照していただきたい．

手続き的プログラミング言語の言語機能としての **goto** 文に関する理論的研究は表示的意味論での接続法や **goto** 文の検証規則をはじめとする様々な成果を生み出した．しかし，Dijkstra が彼の手紙で問題提起し本論文の主題でもある **goto 文の使用法**への指針で理論的な裏付けを持つものは，制御フローの表現力に関する理論的結果に基づき Dijkstra 本来の意図とはまったく異なる**「構造的プログラム = goto-less プログラム」**として Mills が提唱し産業界に流布した Mills 流の構造的プログラミング[13] での**「goto 文なしで書けぬものはない．ゆえに使用は許さん」**以外は知られていない．

本論文では，Plauger のエッセイ[14] での感覚的主張——有限状態機械を用いて設計されたプログラムの場合，**goto** 文を用いたスタイルのプログラムの方が状態を表現する新たな変数（以下「状態変数」と呼ぶ）を導入し 3 基本構造のみで記述する Mills 流の構造的スタイルのプログラムよりも分かりやすい——に対し，Hoare 論理での検証の立場から以下の裏付けを与える．すなわち，有限状態機械モデルに基づくプログラミングでは，状態変数を導入しループと状態変数の値による場合分けとで記述するスタイルは状態を文ラベル——つまり **goto** 文の飛び先ラベル——で状態遷移を **goto** 文によるジャンプの形で表現するスタイルと比べ検証がより複雑になることを示す．ここで「より複雑」とは，前者の「構造的スタイル」で導入されるループの不変条件が **goto** 文を用いたスタイルでの各状態に対する事前条件より導かざるをえず，しかも前者のプログラムに対する表明が後者のプログラム中の対応する表明と同じかより複雑な形となり，機械的検証での時間計算量も増加する，という意味である．この有限状態機械モデルからのプログラムは Mills 流の構造的プログラミングでは **goto** 文を削除すべき典型的ケースであることを注意しておく．

本論文の構成を以下に示す．次章では本論文で使用する最小限の Hoare 論理の体系を与える．3 章では具体例に関し **goto** 文によって状態遷移を表現したプログラムと Mills 流の「構造的プログラミング」に従い状態変数を導入し **goto** 文を消去したプログラムとの各々を Hoare 論理によって検証し両者を比較する．4 章では **goto** 文で状態遷移を表したプログラムから **goto** 文のないプログラムへと状態変数を導入して変形したときの検証での表明の変化を一般的なプログラムスキーマに対して示し，機械的検証での時間計算量の比較も与える．5 章は以上の議論のまとめである．

2. Hoare 論理

本論文では，プログラミング言語としては Pascal を用いる．Pascal に対する Hoare 論理[10] の規則群の全貌に関しては Hoare らによる Pascal の公理的定義[11] があるが，本章では以下の議論で必要最小限の規則を**図 1** に示す．なお図 1 中の公理や推論規則は式の評価中に変数の値を変更する「式の副作用」はないという前提の下のものである．**goto** 文の推論規則に関しては Clint ら[3] の付録で与えられ de Bruin[2] で用いられている形を採用する．Hoare 論理に関する用語については林[9] に従う．

表明を記述する論理式で用いる論理記号は次のとおり．“∧”，“∨”，“⊃” は論理積，論理和，含意を，“⊥” は命題定数の偽を，それぞれ表す．Pascal の Boolean 型の式は論理式としても用いるが，本来の論理式か Pascal の Boolean 型の式由来かの区別のため，Boolean 型の演算子や定数は “**and**”，“true”，“false” と，論理記号とは区別して表す．ただし論理演算上は対応する論理記号/命題定数と同一視する．

以下,「プログラム」とは，Pascal の文法上のプログラム（“**program**” から “**end**.” まで）でなく，Hoare 論理の議論でよく見られる[9] 実行文（代入文・空文・複合文・制御構造を与える文）の 1 つ以上が “;” で区切られた並びを表す．変数の宣言等は省略する．

goto 文の推論規則に関する説明は別稿[12]2 章を参照されたい．ポイントは**「goto 文を使うなら飛び先の文の事前条件を明確にすべし」**である．

ファイル入出力に関して

本論文では例題記述でファイル入出力を用いるが，Pascal でのファイル入出力の仕様を原因とする複雑さで議論が不明瞭にならぬよう，ファイル入出力に関連する事項に対しては，検証での取扱いが可能な限り簡潔にできる仕様を以下で定め記述に用いる．

f をファイルとするとき，f は 2 つの列——f_{L} と f_{R}——からなると考える．つまり $f \stackrel{\mathrm{def}}{=} \langle f_{\mathrm{L}}, f_{\mathrm{R}} \rangle$ である．f_{L}，f_{R} は通常のプログラムコード中では参照で

$$\{A\}\,(*\ \text{空文}\ *)\,\{A\}\quad[\text{空文}]$$

$$\{A[e/x]\}\,x := e\,\{A\}\quad[\text{代入文}]$$

$$\frac{\{A\}\,S_1\,\{B\}\qquad\{B\}\,S_2\,\{C\}}{\{A\}\,S_1; S_2\,\{C\}}\quad[\text{順次接続}]$$

$$\frac{\{A\wedge(\bigwedge_{k=1}^{i-1}\neg b_k)\wedge b_i\}\,S_i\,\{B\}\ (i=1,\ldots,n)\qquad\{A\wedge(\bigwedge_{k=1}^{n}\neg b_k)\}\,S_{n+1}\,\{B\}}{\{A\}\,\textbf{if}\ b_1\ \textbf{then}\ S_1\ \textbf{else if}\ b_2\ \textbf{then}\ S_2\ \ldots\ \textbf{else if}\ b_n\ \textbf{then}\ S_n\ \textbf{else}\ S_{n+1}\,\{B\}}\quad[\text{多分岐 }\textbf{if}\text{ 文}]$$

$$\frac{\{A\wedge e=c_1\}\,S_1\,\{B\}\qquad\ldots\qquad\{A\wedge e=c_n\}\,S_n\,\{B\}}{\{A\wedge e\in\{c_1,\ldots,c_n\}\}\,\textbf{case}\ e\ \textbf{of}\ c_1\colon S_1; c_2\colon S_2;\ \ldots;c_n\colon S_n\ \textbf{end}\,\{B\}}\quad[\textbf{case}\text{ 文}]$$

$$\frac{\{A\wedge b\}\,S\,\{A\}}{\{A\}\,\textbf{while}\ b\ \textbf{do}\ S\,\{A\wedge\neg b\}}\quad[\textbf{while}\text{ 文}]$$

$$\frac{\{A\}\,S\,\{B\}}{\{A\}\,\textbf{begin}\ S\ \textbf{end}\,\{B\}}\quad[\text{複合文}]$$

$$\frac{A'\supset A\qquad\{A\}\,S\,\{B\}\qquad B\supset B'}{\{A'\}\,S\,\{B'\}}\quad[\text{帰結規則}]$$

$$\frac{\{B\}\,\textbf{goto}\ L\,\{\bot\}\vdash\{A\}\,S_1\,\{B\}\qquad\{B\}\,\textbf{goto}\ L\,\{\bot\}\vdash\{B\}\,S_2\,\{C\}}{\{A\}\,S_1; L\colon S_2\,\{C\}}\quad[\textbf{goto}\text{ 文}]$$

図 1 Pascal の Hoare 論理の公理と推論規則（本論文で用いるもののみ）
Fig. 1 Axioms and inference rules of Hoare Logic for a Pascal subset.

きず表明中でのみ使用可能な変数（仕様変数）とする．f_{L} は，入力ファイルの場合はすでに読んだ部分を，出力ファイルの場合は書いた部分を，それぞれ表す．f_{R} は，入力ファイルの場合はまだ読んでいない部分を，出力ファイルの場合はつねに空の列 ε を，それぞれ表す．

表明記述のための列に関する記法を以下に定める．eof は列の要素として現れない特別の値を表す定数名とし，ファイルの終了判定のためにプログラム中でも使用可能とする．つまり D を列の要素の集合とするとき $\mathrm{eof}\notin D$ と定める．

列の基本演算に関しては以下の約束をする．これら演算はプログラムコード中では使用が許されず表明中でのみ使用できると定める．以下，r，s，t，u は列を，c，d は列の要素を，それぞれ表すメタ変数とする．

列の分解演算に関しては次のように約束する．$\mathrm{fst}(s)$ と $\mathrm{bwd}(s)$ は列 s の先頭（first）要素とそれ以外の残り（backward）の部分列をそれぞれ与える関数とする．部分関数での関数値の未定義に起因する表明記述上の複雑さを避けるため，空列 ε に対する関数値として $\mathrm{fst}(\varepsilon)=\mathrm{eof}$ および $\mathrm{bwd}(\varepsilon)=\varepsilon$ と定める．

列の構成演算に関しては次のように約束する．$s::t$ は 2 つの列 s，t をこの順に連結した列を，$c\cdot\!:s$ は列 s に先頭要素として c を付加した列を，$s:\!\cdot c$ は列 s に最終要素として c を追加した列を，それぞれ表す．

これら列の基本演算は（分解演算の空列に対する値以外は）常識的な各種公理——結合則等——を満たすとする．基本演算のアリティは次のようにまとめられる：

$$\begin{aligned}\mathrm{fst}&: D^*\to D\cup\{\mathrm{eof}\},\\ \mathrm{bwd}&: D^*\to D^*,\\ (\_::\_)&: D^*\times D^*\to D^*,\\ (\_\cdot\!:\_)&: D\times D^*\to D^*,\\ (\_:\!\cdot\_)&: D^*\times D\to D^*.\end{aligned}$$

上記のアリティで括弧の付け方が一意に定まる場合や結合則から括弧の付け方に依存しない場合は括弧を省く．たとえば “$s::c\cdot\!:d$” は “$(s::c)\cdot\!:d$” である．以上のアリティから “$c\cdot\!:s$” とか “$s:\!\cdot c$” と書くときはつねに $c\neq\mathrm{eof}$ が要請されるので，その文脈では条件 $c\neq\mathrm{eof}$ を省く．

上で定義したファイル（および列）の概念を用いて本論文の例題記述で使う入力手続き getitem と出力手続き putitem とを**図 2** に示す各三つ組により仕様が与えられる手続きとして定義する．

3. 例題：注釈除去問題

本章では，問題を有限状態機械としてモデル化し設計した場合は状態遷移を **goto** 文で表す方が状態変数を導入し Mills の意味でプログラムを構造化するより

$\{f_{\mathrm{R}} = \varepsilon \wedge P[\mathrm{eof}/x] \vee f_{\mathrm{R}} \neq \varepsilon \wedge Q[(f_{\mathrm{L}} :\cdot \mathrm{fst}(f_{\mathrm{R}}))/f_{\mathrm{L}}, \mathrm{bwd}(f_{\mathrm{R}})/f_{\mathrm{R}}, \mathrm{fst}(f_{\mathrm{R}})/x]\}$
$\mathrm{getitem}(f, x)$
$\{x = \mathrm{eof} \wedge f_{\mathrm{R}} = \varepsilon \wedge P \vee x \neq \mathrm{eof} \wedge Q\}$

$\{e = \mathrm{eof} \wedge P[\varepsilon/f_{\mathrm{R}}] \vee e \neq \mathrm{eof} \wedge P[(f_{\mathrm{L}} :\cdot e)/f_{\mathrm{L}}, \varepsilon/f_{\mathrm{R}}]\}$
$\mathrm{putitem}(f, e)$
$\{P\}$

図 2 入出力手続きに関する三つ組
Fig. 2 Triples satisfied by input/output procedures.

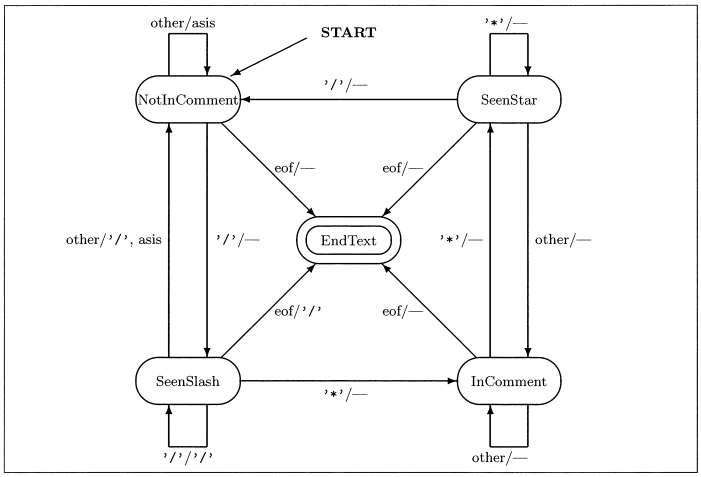

図 3 注釈除去の状態遷移図
Fig. 3 The state transition diagram of removing comments.

も自然に見える，との Plauger の著書[14] の主張に関し，Plauger 自身による次の問題に即して両者のプログラミングスタイルでの検証上の違いを検討する．

> **問題** 入力ファイル in から入力手続き getitem でテキストを 1 文字ずつ読み，開始文字列 `"/*"` から始まり終了文字列 `"*/"` で終わる部分（注釈と呼ぶ）を除いて出力ファイル out に出力手続き putitem で書き込む．注釈の入れ子は考えない．つまり注釈開始文字列から最初に出現する終了文字列までを（その途中で新たな開始文字列が出現しているか否かによらず）注釈として扱う．注釈の途中で入力が尽きた場合，注釈終了と看做し正常に終了する．

この問題を有限状態機械でモデル化した状態遷移図を**図 3** に示す．この問題に対し，状態遷移を **goto** 文による飛び越しで表したプログラムを**図 4** に，また，状態変数と **while** 文・**case** 文の導入により **goto** 文を除き Mills の意味で構造化したプログラムを**図 5** に，それぞれ示す．なお，図 3 の状態遷移図や図 4，図 5 の各プログラムは Plauger の本[14] でのプログラムを元に修正☆を加え Pascal に書き改めたが識別子等に関しては変更していない．ただし図 3 では Plauger の状態遷移図での遷移条件のほかに各遷移での出力を “/” の右側に示した．出力欄の “—” は出力がないことを，“asis” はそのときの入力値を出力することを，複数の値を “,” で区切った並びはすべてをその順に出力することを，それぞれ表す．

goto 文を用いるスタイルと用いないものとの間の見掛け上の差（Pascal の構文上の制約に起因）を最小にし図 4 と図 5 との対応を明確にするため，**goto** 文を用いた図 4 のプログラムには冗長な **begin** — **end** 対と **goto** 文とが含まれていることを注意しておく．図 4 のプログラムと図 5 のとの違いは次の 3 点であ

☆ Plauger の著書の状態遷移図やプログラムは小さな紙面に収めるために端折ったせいか，場合分けには抜けがある．たとえば彼のプログラムでは注釈終了記号 `"*/"` の直前の文字が `'*'` の場合—注釈の最後の部分が `"**/"` となっているとき—に `"*/"` が注釈終了記号として認識されない．

```
 1        goto NotInComment;
 2      NotInComment:
 3        begin
 4          getitem(in, c);
 5          if c = eof then
 6            goto EndText
 7          else if c = '/' then
 8            goto SeenSlash
 9          else
10            begin
11              putitem(out, c);
12              goto NotInComment
13            end
14        end;
15      SeenSlash:
16        begin
17          getitem(in, c);
18          if c = eof then
19            begin
20              putitem(out, '/');
21              goto EndText
22            end
23          else if c = '/' then
24            begin
25              putitem(out, '/');
26              goto SeenSlash
27            end
28          else if c = '*' then
29            goto InComment
30          else
31            begin
32              putitem(out, '/');
33              putitem(out, c);
34              goto NotInComment
35            end
36        end;
37      InComment:
38        begin
39          getitem(in, c);
40          if c = eof then
41            goto EndText
42          else if c = '*' then
43            goto SeenStar
44          else
45            goto InComment
46        end;
47      SeenStar:
48        begin
49          getitem(in, c);
50          if c = eof then
51            goto EndText
52          else if c = '*' then
53            goto SeenStar
54          else if c = '/' then
55            goto NotInComment
56          else
57            goto InComment
58        end;
59      EndText:
60        (* 空文 *)
```

図 4 注釈除去のプログラム（**goto** 文使用）
Fig. 4 The comment-removing program with **goto**'s.

```
 1      state := NotInComment;
 2      while ¬(state = EndText) do
 3        case state of
 4          NotInComment:
 5            begin
 6              getitem(in, c);
 7              if c = eof then
 8                state := EndText
 9              else if c = '/' then
10                state := SeenSlash
11              else
12                begin
13                  putitem(out, c);
14                  state := NotInComment
15                end
16            end;
17          SeenSlash:
18            begin
19              getitem(in, c);
20              if c = eof then
21                begin
22                  putitem(out, '/');
23                  state := EndText
24                end
25              else if c = '/' then
26                begin
27                  putitem(out, '/');
28                  state := SeenSlash
29                end
30              else if c = '*' then
31                state := InComment
32              else
33                begin
34                  putitem(out, '/');
35                  putitem(out, c);
36                  state := NotInComment
37                end
38            end;
39          InComment:
40            begin
41              getitem(in, c);
42              if c = eof then
43                state := EndText
44              else if c = '*' then
45                state := SeenStar
46              else
47                state := InComment
48            end;
49          SeenStar:
50            begin
51              getitem(in, c);
52              if c = eof then
53                state := EndText
54              else if c = '*' then
55                state := SeenStar
56              else if c = '/' then
57                state := NotInComment
58              else
59                state := InComment
60            end
61        end
```

図 5 注釈除去のプログラム（**goto** 文不使用）
Fig. 5 The comment-removing program without **goto**.

る：

(1) 前者で次状態のラベルを飛び先とする状態遷移を表す **goto** 文は，後者では次状態名を状態変数 state に設定する代入文となっていること；

(2) 前者での終状態 EndText に至るまで状態遷移を繰り返すことは，後者では **while** 文という明示的な反復で表されていること；

(3) 状態ごとの処理の振り分けは，前者では当該状態に対するラベルの付いた文に **goto** 文で飛ぶことで自動的に達成されるのに対し，後者では

$$
\begin{array}{rcl}
\mathit{Pre} & \stackrel{\text{def}}{=} & \mathrm{in_L} = \varepsilon \wedge \mathrm{out_L} = \varepsilon \wedge \mathit{FileInv}, \\
\mathit{Post} & \stackrel{\text{def}}{=} & \mathrm{in_L} = \mathit{whole} \wedge \mathrm{in_R} = \varepsilon \wedge \mathit{CmtRmvd}(\mathrm{out_L}, \mathit{whole}); \\
 & & \\
\mathit{CmtRmvd}(t, s) & \stackrel{\text{def}}{=} & \mathit{AtEndText}(t, s, \varepsilon); \\
 & & \\
\mathit{PreNotInComment} & \stackrel{\text{def}}{=} & \mathit{NotInCommentAux}(\mathrm{out_L}, \mathrm{in_L}), \\
\mathit{NotInCommentAux}(t, s) & \stackrel{\text{def}}{=} & \mathrm{in_R} = \varepsilon \wedge \mathit{AtEndText}(t, s, \mathrm{in_R}) \wedge \mathit{FileInv} \\
 & & \quad \vee\, \mathrm{in_R} \neq \varepsilon \wedge \mathit{SlashOrNot}(t, s, \mathrm{bwd}(\mathrm{in_R}), \mathrm{fst}(\mathrm{in_R})) \wedge \mathit{FileInv}', \\
\mathit{SlashOrNot}(t, s, r, c) & \stackrel{\text{def}}{=} & \quad c = \texttt{'/'} \wedge \mathit{AtSeenSlash}(t, s :\cdot c, r) \\
 & & \quad \vee\, c \neq \texttt{'/'} \wedge \mathit{AtNotInComment}(t :\cdot c, s :\cdot c, r); \\
 & & \\
\mathit{PreSeenSlash} & \stackrel{\text{def}}{=} & \mathit{SeenSlashAux}(\mathrm{out_L}, \mathrm{in_L}), \\
\mathit{SeenSlashAux}(t, s) & \stackrel{\text{def}}{=} & \mathrm{in_R} = \varepsilon \wedge \mathit{AtEndText}(t, s, \mathrm{in_R}) \wedge \mathit{FileInv} \\
 & & \quad \vee\, \mathrm{in_R} \neq \varepsilon \wedge \mathit{SlashStarOrOther}(t, s, \mathrm{bwd}(\mathrm{in_R}), \mathrm{fst}(\mathrm{in_R})) \wedge \mathit{FileInv}', \\
\mathit{SlashStarOrOther}(t, s, r, c) & \stackrel{\text{def}}{=} & \quad c = \texttt{'/'} \wedge \mathit{AtSeenSlash}(t :\cdot \texttt{'/'}, s :\cdot c, r) \\
 & & \quad \vee\, c = \texttt{'*'} \wedge \mathit{AtInComment}(t, s :\cdot c, r) \\
 & & \quad \vee\, c \notin \{\texttt{'/'}, \texttt{'*'}\} \wedge \mathit{AtNotInComment}(t :\cdot \texttt{'/'} :\cdot c, s :\cdot c, r); \\
 & & \\
\mathit{FileInv} & \stackrel{\text{def}}{=} & \mathrm{in_L} :: \mathrm{in_R} = \mathit{whole}, \\
\mathit{FileInv}' & \stackrel{\text{def}}{=} & (\mathrm{in_L} :\cdot \mathrm{fst}(\mathrm{in_R})) :: \mathrm{bwd}(\mathrm{in_R}) = \mathit{whole}.
\end{array}
$$

図 6 プログラム全体の事前/事後条件および状態 NotInComment の事前条件と補助的な述語
Fig. 6 Pre-/Postconditions of the program, preconditions of states NotInComment and SeenSlash, and their auxiliary predicates.

状態変数 state の値により明示的に処理を振り分ける **case** 文を必要としていること．

以上の 2 種類のプログラムについて，Hoare 論理による正しさの証明を考える．ここで，注釈除去問題の場合のプログラム全体の事前条件 *Pre* は，

$\mathrm{in_L} = \varepsilon \wedge \mathrm{out_L} = \varepsilon \wedge \mathrm{in_L} :: \mathrm{in_R} = \mathit{whole}$

である．これは，入力ファイル in からまだ何も入力しておらず出力ファイル out に何も出力していない，ということを表す．論理積の最後の成分は，入力ファイル全体の内容を以後の表明中で参照するために *whole* と名付ける（仕様変数で表す）という意味である．一方，この問題でのプログラム全体の事後条件 *Post* は，

$\mathrm{in_L}$=*whole* $\wedge\, \mathrm{in_R}$=$\varepsilon \wedge$ *CmtRmvd*$(\mathrm{out_L}, \mathit{whole})$

であり，入力 in を読み終わり（$\mathrm{in_R} = \varepsilon$）出力内容 $\mathrm{out_L}$ は元の全入力 *whole* から注釈を除いたものだ（*CmtRmvd*$(\mathrm{out_L}, \mathit{whole})$）ということを表す．

以上の事前/事後条件に対し両プログラムの証明を与えるが，**goto** 文を用いた図 4 のプログラムでは 2 章の **goto** 文の規則のポイントとして述べたとおり **goto** 文の飛び先文——この場合は各状態での処理——の事前条件を把握しなければならない．状態 NotInComment と SeenSlash の各々の事前条件 *PreNotInComment*, *PreSeenSlash* の定義をプログラム全体の事前/事後条件とともに**図 6** に示す．

それら各状態の事前条件の記述に用いる述語群が満たすべき論理的性質（公理）を**図 7** に示す．たとえば，述語 *AtNotInComment*(t, s, r) は，処理後 (*t*arget) の文字列 t は元 (*s*ource) の文字列 s に対して残り (*r*est) の文字列 r という文脈の下で注釈を除いたものだ，という意味である．図 7 の各述語が第 3 の引数を持つ理由は，たとえば文字 `'/'` が注釈の開始記号 `"/*"` の最初の文字なのか出力すべき本文中の斜線という文字なのかは次の文字という文脈に基づかないと判別不能なことによる．このことは述語 *AtSeenSlash* に関する公理に現れている．*AtSeenSlash*$(t, s :\cdot \texttt{'/'}, d :\cdot r)$ という形を含意の左辺とする公理が図 7 に 3 つ存在するが，残りの部分の先頭文字 d が `'/'` か `'*'` かそれ以外かで含意の右辺が異なる．つまり *AtSeenSlash*$(t, s :\cdot \texttt{'/'}, d :\cdot r)$ は `'/'` を読み込んだ（ゆえに第 2 引数は $s :\cdot$ `'/'` と `'/'` が付加された形）が `'/'` が注釈の一部か否かの判定を保留している（ゆえに `'/'` は出力されていず第 1 引数は $t :\cdot$ `'/'` でなく t）状況にプログラムがあることを表している．**以上のように，図 7 に示した公理群で規定される述語群——以下「状態遷移述語」と呼ぶ——はプログラムの状態遷移を論理式の形でシミュレートしている**．

状態遷移を **goto** 文で行う図 4 のプログラムの証明アウトラインは**図 8** となる．ただし本図は図 4 のプログラム中の状態 NotInComment での処理に対する部分のみである．他状態に対する証明アウトラインも同様なので省く．一部の表明で「テキスト上の次の状態の事前条件」とあるのは，その部分の表明の論理的な任意性を強調するためで，図 4 の場合はテキスト上で NotInComment の次に書かれた状態が SeenSlash

$$
\begin{aligned}
&\mathit{AtNotInComment}(\varepsilon,\varepsilon,r),\\
\mathit{AtNotInComment}(t,s,\varepsilon) &\supset \mathit{AtEndText}(t,s,\varepsilon),\\
\mathit{AtNotInComment}(t,s,c\cdot r)\wedge c\neq \texttt{'/'} &\supset \mathit{AtNotInComment}(t\cdot c,s\cdot c,r),\\
\mathit{AtNotInComment}(t,s,\texttt{'/'}\cdot r) &\supset \mathit{AtSeenSlash}(t,s\cdot \texttt{'/'},r);\\
\\
\mathit{AtSeenSlash}(t,s\cdot \texttt{'/'},\varepsilon) &\supset \mathit{AtEndText}(t\cdot \texttt{'/'},s\cdot \texttt{'/'},\varepsilon),\\
\mathit{AtSeenSlash}(t,s\cdot \texttt{'/'},c\cdot r)\wedge c\neq \texttt{'/'}\wedge c\neq \texttt{'*'} &\supset \mathit{AtNotInComment}(t\cdot \texttt{'/'}\cdot c,s\cdot \texttt{'/'}\cdot c,r),\\
\mathit{AtSeenSlash}(t,s\cdot \texttt{'/'},\texttt{'/'}\cdot r) &\supset \mathit{AtSeenSlash}(t\cdot \texttt{'/'},s\cdot \texttt{'/'}\cdot \texttt{'/'},r),\\
\mathit{AtSeenSlash}(t,s\cdot \texttt{'/'},\texttt{'*'}\cdot r) &\supset \mathit{AtInComment}(t,s::\texttt{"/*"},r);\\
\\
\mathit{AtInComment}(t,s::\texttt{"/*"}::u,\varepsilon) &\supset \mathit{AtEndText}(t,s::\texttt{"/*"}::u,\varepsilon),\\
\mathit{AtInComment}(t,s::\texttt{"/*"}::u,c\cdot r)\wedge c\neq \texttt{'*'} &\supset \mathit{AtInComment}(t,s::\texttt{"/*"}::u\cdot c,r),\\
\mathit{AtInComment}(t,s::\texttt{"/*"}::u,\texttt{'*'}\cdot r) &\supset \mathit{AtSeenStar}(t,s::\texttt{"/*"}::u\cdot \texttt{'*'},r);\\
\\
\mathit{AtSeenStar}(t,s::\texttt{"/*"}::u\cdot \texttt{'*'},\varepsilon) &\supset \mathit{AtEndText}(t,s::\texttt{"/*"}::u\cdot \texttt{'*'},\varepsilon),\\
\mathit{AtSeenStar}(t,s::\texttt{"/*"}::u\cdot \texttt{'*'},c\cdot r)\wedge c\neq \texttt{'*'}\wedge c\neq \texttt{'/'} &\supset \mathit{AtInComment}(t,s::\texttt{"/*"}::u\cdot \texttt{'*'}\cdot c,r),\\
\mathit{AtSeenStar}(t,s::\texttt{"/*"}::u\cdot \texttt{'*'},\texttt{'*'}\cdot r) &\supset \mathit{AtSeenStar}(t,s::\texttt{"/*"}::u\cdot \texttt{'*'}\cdot \texttt{'*'},r),\\
\mathit{AtSeenStar}(t,s::\texttt{"/*"}::u\cdot \texttt{'*'},\texttt{'/'}\cdot r) &\supset \mathit{AtNotInComment}(t,s::\texttt{"/*"}::u::\texttt{"*/"},r).
\end{aligned}
$$

図 7 状態遷移を表現する各述語とそれらに関する諸公理
Fig. 7 Axioms for predicates expressing state transitions.

なので *PreSeenSlash* となる．

一方，状態変数を導入し **goto** 文を除去した図 5 のプログラムの証明アウトラインのためには **while** 文のループ不変条件が必要である．当該ループ前後にはほとんど処理がなくループ不変条件をトップダウンで決められず，ループ本体からボトムアップに求めなければならない．それで得られる **while** ループの不変条件は

$$
\begin{aligned}
&\text{state=NotInComment} \wedge \mathit{PreNotInComment}\\
\vee\ &\text{state=SeenSlash} \wedge \mathit{PreSeenSlash}\\
\vee\ &\text{state=InComment} \wedge \mathit{PreInComment}\\
\vee\ &\text{state=SeenStar} \wedge \mathit{PreSeenStar}\\
\vee\ &\text{state=EndText} \wedge \mathit{Post}
\end{aligned}
$$

であり，要は「どれかの状態にありその状態での事前条件が成立している」という検証にとってほとんど有効な情報を持たない表明である．このループ不変条件に基づいて NotInComment 状態の処理部分の証明アウトラインを構成したのが**図 9** である．

図 9 の証明アウトラインと図 8 のとの対応は明らかであるが，前者は状態変数 state の値に関する余分な論理積成分を正しく維持しなければならないため，後者よりも少しばかり複雑な表明となっている．

以上のとおり，本章での注釈除去問題のプログラムの場合，状態変数の導入で **goto** 文を除き Mills 流の意味で「構造化」したプログラムの検証は状態遷移を **goto** 文で表したプログラムの検証と比べ何ら簡単にならない．むしろ「構造化」された方が検証での表明は複雑で帰結規則の適用回数もより多く（図 8 では 3 回，図 9 では 5 回．次章の一般形での議論から分かるが「構造化」スタイルではループ不変条件から各状態の事前条件を取り出すため余分な帰結規則の適用がつねに必要と）なっている．

何より「構造化」にともない導入された **while** ループの不変条件が **goto** 文を用いたプログラムの検証での知識（各状態に対する事前条件）なくしては導けない，ということは「構造化」されたプログラムを検証するうえでの致命的な問題である．これでは検証の立場からは状態変数の導入で「構造化」した意義がまったくない．

ゆえに本章の例では Plauger による「**goto** 文を用いた方が分かりやすい」という感覚的な指摘は Hoare 論理による検証の容易さと一致すると結論してよい．

本章の最後として，本論文の主題とは少し外れるが，本章での Hoare 論理の検証が何に対する正しさを証明したことになるのか，という点についてまとめておく．

以上の検証は図 4 や図 5 のプログラムが設計結果としての図 3 の状態遷移図を正しく実装していることの証明である．すなわち，本章の検証は図 3 の状態遷移図の正しさを前提としており，「全注釈を除去する」という問題自体に対する正当性の証明ではない．

実際，図 6 での *PreNotInComment* の事前条件の論理式の形が図 3 での状態 NotInComment からの状態遷移の矢印の出方に対応している（*PreSeenSlash* についても同様）のは明らかで図 7 の各公理が図 3 の状態遷移と（各時点でのファイルの状況を引数で表現しつつ）正確に対応していることも一目瞭然である．

問題仕様に対する正当性証明のためには，述語

1 $\{PreNotInComment\}$
2 NotInComment:
3 **begin**
4 $\{PreNotInComment\}$
5 getitem(in, c)
6 $\left\{\begin{array}{l} \mathrm{c} = \mathrm{eof} \wedge \mathrm{in_R} = \varepsilon \wedge AtNotInComment(\mathrm{out_L}, \mathrm{in_L}, \mathrm{in_R}) \wedge FileInv \\ \vee\, \mathrm{c} \neq \mathrm{eof} \wedge \left(\begin{array}{l} \mathrm{c} = \texttt{'/'} \wedge AtSeenSlash(\mathrm{out_L}, \mathrm{in_L}, \mathrm{in_R}) \\ \vee\, \mathrm{c} \neq \texttt{'/'} \wedge AtNotInComment(\mathrm{out_L} :\cdot \mathrm{c}, \mathrm{in_L}, \mathrm{in_R}) \end{array}\right) \wedge FileInv \end{array}\right\}$;
7 **if** c = eof **then**
8 $\left\{\left(\begin{array}{l} \mathrm{c} = \mathrm{eof} \wedge \mathrm{in_R} = \varepsilon \wedge AtNotInComment(\mathrm{out_L}, \mathrm{in_L}, \mathrm{in_R}) \wedge FileInv \\ \vee\, \mathrm{c} \neq \mathrm{eof} \wedge \left(\begin{array}{l} \mathrm{c} = \texttt{'/'} \wedge AtSeenSlash(\mathrm{out_L}, \mathrm{in_L}, \mathrm{in_R}) \\ \vee\, \mathrm{c} \neq \texttt{'/'} \wedge AtNotInComment(\mathrm{out_L} :\cdot \mathrm{c}, \mathrm{in_L}, \mathrm{in_R}) \end{array}\right) \wedge FileInv \end{array}\right) \wedge \mathrm{c} = \mathrm{eof}\right\}$
9 $\{Post\}$
10 **goto** EndText
11 $\{\bot\}$
12 { テキスト上の次の状態の事前条件 }
13 **else if** c = '/' **then**
14 $\left\{\left(\begin{array}{l} \mathrm{c} = \mathrm{eof} \wedge \mathrm{in_R} = \varepsilon \wedge AtNotInComment(\mathrm{out_L}, \mathrm{in_L}, \mathrm{in_R}) \wedge FileInv \\ \vee\, \mathrm{c} \neq \mathrm{eof} \wedge \left(\begin{array}{l} \mathrm{c} = \texttt{'/'} \wedge AtSeenSlash(\mathrm{out_L}, \mathrm{in_L}, \mathrm{in_R}) \\ \vee\, \mathrm{c} \neq \texttt{'/'} \wedge AtNotInComment(\mathrm{out_L} :\cdot \mathrm{c}, \mathrm{in_L}, \mathrm{in_R}) \end{array}\right) \wedge FileInv \end{array}\right) \wedge \neg(\mathrm{c} = \mathrm{eof}) \wedge \mathrm{c} = \texttt{'/'}\right\}$
15 $\{PreSeenSlash\}$
16 **goto** SeenSlash
17 $\{\bot\}$
18 { テキスト上の次の状態の事前条件 }
19 **else**
20 $\left\{\left(\begin{array}{l} \mathrm{c} = \mathrm{eof} \wedge \mathrm{in_R} = \varepsilon \wedge AtNotInComment(\mathrm{out_L}, \mathrm{in_L}, \mathrm{in_R}) \wedge FileInv \\ \vee\, \mathrm{c} \neq \mathrm{eof} \wedge \left(\begin{array}{l} \mathrm{c} = \texttt{'/'} \wedge AtSeenSlash(\mathrm{out_L}, \mathrm{in_L}, \mathrm{in_R}) \\ \vee\, \mathrm{c} \neq \texttt{'/'} \wedge AtNotInComment(\mathrm{out_L} :\cdot \mathrm{c}, \mathrm{in_L}, \mathrm{in_R}) \end{array}\right) \wedge FileInv \end{array}\right) \wedge \neg(\mathrm{c} = \mathrm{eof}) \wedge \neg(\mathrm{c} = \texttt{'/'})\right\}$
21 $\{\mathrm{c} = \mathrm{eof} \wedge NotInCommentAux(\mathrm{out_L}, \mathrm{in_L}) \vee \mathrm{c} \neq \mathrm{eof} \wedge NotInCommentAux(\mathrm{out_L} :\cdot \mathrm{c}, \mathrm{in_L})\}$
22 **begin**
23 $\{\mathrm{c} = \mathrm{eof} \wedge NotInCommentAux(\mathrm{out_L}, \mathrm{in_L}) \vee \mathrm{c} \neq \mathrm{eof} \wedge NotInCommentAux(\mathrm{out_L} :\cdot \mathrm{c}, \mathrm{in_L})\}$
24 putitem(out, c)
25 $\{PreNotInComment\}$;
26 **goto** NotInComment
27 $\{\bot\}$
28 **end**
29 $\{\bot\}$
30 { テキスト上の次の状態の事前条件 }
31 { テキスト上の次の状態の事前条件 }
32 **end**
33 { テキスト上の次の状態の事前条件 }

図 8 図 4 中の状態 NotInComment の処理に対する証明アウトライン
Fig. 8 The proof-outline of the state NotInComment in Fig. 4.

$CmtRmvd(t, s)$ が表そうとする「t は s から全注釈を除いたもの」という内容を論理式の形で与え，それに基づき図 7 中の各状態遷移述語を論理式で定義し，それら論理式が図 7 の公理群を満たすことを示す必要がある．

4. 有限状態機械に基づくプログラムの一般形に対する表明付け

本章では，前章の例題で観察した，有限状態機械に基づくプログラムで状態遷移に **goto** 文を使用した場合と状態変数の導入で **goto** 文を除去した場合とで，Hoare 論理による検証での表明付けがどのように変化するかを，一般のプログラムスキーマの形で検討する．

有限状態機械モデルに基づいたプログラムに関する一般論を展開するため，状態遷移しながら各状態で入力ファイル *infile* から変数 v にデータを読み込み値に応じ処理を分けるプログラムの一般形について，**goto** 文で状態遷移を表現したものと状態変数 sv の導入により Mills の意味で構造化したものとの双方を**図 10** に示す．なお，初期状態は Σ_1，終状態は Σ_{n+1} とする．各 i, j について $\Sigma_{i,j}$ は $\Sigma_1, \ldots, \Sigma_{n+1}$ のどれかである．

まず，状態遷移を **goto** 文による飛び越しで表現したプログラムスキーマの状態 Σ_i での処理 S_i に対す

1 $\{(\mathrm{state} = \mathrm{NotInComment} \wedge \mathit{PreNotInComment} \vee \ldots \vee \mathrm{state} = \mathrm{EndText} \wedge \mathit{Post}) \wedge \mathrm{state} = \mathrm{NotInComment}\}$
2 NotInComment:
3 **begin**
4 $\{(\mathrm{state} = \mathrm{NotInComment} \wedge \mathit{PreNotInComment} \vee \ldots \vee \mathrm{state} = \mathrm{EndText} \wedge \mathit{Post}) \wedge \mathrm{state} = \mathrm{NotInComment}\}$
5 $\{\mathit{PreNotInComment}\}$
6 getitem(in, c)
7 $\left\{\begin{array}{l} \mathrm{c} = \mathrm{eof} \wedge \mathrm{in_R} = \varepsilon \wedge \mathit{AtNotInComment}(\mathrm{out_L}, \mathrm{in_L}, \mathrm{in_R}) \wedge \mathit{FileInv} \\ \vee\, \mathrm{c} \neq \mathrm{eof} \wedge \left(\begin{array}{l} \mathrm{c} = \texttt{'/'} \wedge \mathit{AtSeenSlash}(\mathrm{out_L} :\cdot \mathrm{c}, \mathrm{in_L}, \mathrm{in_R}) \\ \vee\, \mathrm{c} \neq \texttt{'/'} \wedge \mathit{AtNotInComment}(\mathrm{out_L} :\cdot \mathrm{c}, \mathrm{in_L}, \mathrm{in_R}) \end{array}\right) \wedge \mathit{FileInv} \end{array}\right\}$
8 $\left\{\begin{array}{l} \mathrm{c} = \mathrm{eof} \wedge \mathrm{in_R} = \varepsilon \wedge \mathit{AtNotInComment}(\mathrm{out_L}, \mathrm{in_L}, \mathrm{in_R}) \wedge \mathit{FileInv} \\ \vee\, \mathrm{c} \neq \mathrm{eof} \wedge \left(\begin{array}{l} \mathrm{c} = \texttt{'/'} \wedge \mathit{AtSeenSlash}(\mathrm{out_L} :\cdot \mathrm{c}, \mathrm{in_L}, \mathrm{in_R}) \\ \vee\, \mathrm{c} \neq \texttt{'/'} \wedge \mathit{AtNotInComment}(\mathrm{out_L} :\cdot \mathrm{c}, \mathrm{in_L}, \mathrm{in_R}) \end{array}\right) \wedge \mathit{FileInv} \end{array}\right\};$
9 **if** c = eof **then**
10 $\left\{\left(\begin{array}{l} \mathrm{c} = \mathrm{eof} \wedge \mathrm{in_R} = \varepsilon \wedge \mathit{AtNotInComment}(\mathrm{out_L}, \mathrm{in_L}, \mathrm{in_R}) \wedge \mathit{FileInv} \\ \vee\, \mathrm{c} \neq \mathrm{eof} \wedge \left(\begin{array}{l} \mathrm{c} = \texttt{'/'} \wedge \mathit{AtSeenSlash}(\mathrm{out_L} :\cdot \mathrm{c}, \mathrm{in_L}, \mathrm{in_R}) \\ \vee\, \mathrm{c} \neq \texttt{'/'} \wedge \mathit{AtNotInComment}(\mathrm{out_L} :\cdot \mathrm{c}, \mathrm{in_L}, \mathrm{in_R}) \end{array}\right) \wedge \mathit{FileInv} \end{array}\right) \wedge \mathrm{c} = \mathrm{eof}\right\}$
11 $\{\mathrm{EndText} = \mathrm{EndText} \wedge \mathit{Post}\}$
12 state := EndText
13 $\{\mathrm{state} = \mathrm{EndText} \wedge \mathit{Post}\}$
14 $\{\mathrm{state} = \mathrm{NotInComment} \wedge \mathit{PreNotInComment} \vee \ldots \vee \mathrm{state} = \mathrm{EndText} \wedge \mathit{Post}\}$
15 **else if** c = '/' **then**
16 $\left\{\left(\begin{array}{l} \mathrm{c} = \mathrm{eof} \wedge \mathrm{in_R} = \varepsilon \wedge \mathit{AtNotInComment}(\mathrm{out_L}, \mathrm{in_L}, \mathrm{in_R}) \wedge \mathit{FileInv} \\ \vee\, \mathrm{c} \neq \mathrm{eof} \wedge \left(\begin{array}{l} \mathrm{c} = \texttt{'/'} \wedge \mathit{AtSeenSlash}(\mathrm{out_L} :\cdot \mathrm{c}, \mathrm{in_L}, \mathrm{in_R}) \\ \vee\, \mathrm{c} \neq \texttt{'/'} \wedge \mathit{AtNotInComment}(\mathrm{out_L} :\cdot \mathrm{c}, \mathrm{in_L}, \mathrm{in_R}) \end{array}\right) \wedge \mathit{FileInv} \end{array}\right) \wedge \neg(\mathrm{c} = \mathrm{eof}) \wedge \mathrm{c} = \texttt{'/'}\right\}$
17 $\{\mathrm{SeenSlash} = \mathrm{SeenSlash} \wedge \mathit{PreSeenSlash}\}$
18 state := SeenSlash
19 $\{\mathrm{state} = \mathrm{SeenSlash} \wedge \mathit{PreSeenSlash}\}$
20 $\{\mathrm{state} = \mathrm{NotInComment} \wedge \mathit{PreNotInComment} \vee \ldots \vee \mathrm{state} = \mathrm{EndText} \wedge \mathit{Post}\}$
21 **else**
22 $\left\{\left(\begin{array}{l} \mathrm{c} = \mathrm{eof} \wedge \mathrm{in_R} = \varepsilon \wedge \mathit{AtNotInComment}(\mathrm{out_L}, \mathrm{in_L}, \mathrm{in_R}) \wedge \mathit{FileInv} \\ \vee\, \mathrm{c} \neq \mathrm{eof} \wedge \left(\begin{array}{l} \mathrm{c} = \texttt{'/'} \wedge \mathit{AtSeenSlash}(\mathrm{out_L} :\cdot \mathrm{c}, \mathrm{in_L}, \mathrm{in_R}) \\ \vee\, \mathrm{c} \neq \texttt{'/'} \wedge \mathit{AtNotInComment}(\mathrm{out_L} :\cdot \mathrm{c}, \mathrm{in_L}, \mathrm{in_R}) \end{array}\right) \wedge \mathit{FileInv} \end{array}\right) \wedge \neg(\mathrm{c} = \mathrm{eof}) \wedge \neg(\mathrm{c} = \texttt{'/'})\right\}$
23 $\{\mathrm{c} = \mathrm{eof} \wedge \mathit{NotInCommentAux}(\mathrm{out_L}, \mathrm{in_L}) \vee \mathrm{c} \neq \mathrm{eof} \wedge \mathit{NotInCommentAux}(\mathrm{out_L} :\cdot \mathrm{c}, \mathrm{in_L})\}$
24 **begin**
25 $\{\mathrm{c} = \mathrm{eof} \wedge \mathit{NotInCommentAux}(\mathrm{out_L}, \mathrm{in_L}) \vee \mathrm{c} \neq \mathrm{eof} \wedge \mathit{NotInCommentAux}(\mathrm{out_L} :\cdot \mathrm{c}, \mathrm{in_L})\}$
26 putitem(out, c)
27 $\{\mathit{PreNotInComment}\}$;
28 $\{\mathrm{NotInComment} = \mathrm{NotInComment} \wedge \mathit{PreNotInComment}\}$
29 state := NotInComment
30 $\{\mathrm{state} = \mathrm{NotInComment} \wedge \mathit{PreNotInComment}\}$
31 $\{\mathrm{state} = \mathrm{NotInComment} \wedge \mathit{PreNotInComment} \vee \ldots \vee \mathrm{state} = \mathrm{EndText} \wedge \mathit{Post}\}$
32 **end**
33 $\{\mathrm{state} = \mathrm{NotInComment} \wedge \mathit{PreNotInComment} \vee \ldots \vee \mathrm{state} = \mathrm{EndText} \wedge \mathit{Post}\}$
34 $\{\mathrm{state} = \mathrm{NotInComment} \wedge \mathit{PreNotInComment} \vee \ldots \vee \mathrm{state} = \mathrm{EndText} \wedge \mathit{Post}\}$
35 $\{\mathrm{state} = \mathrm{NotInComment} \wedge \mathit{PreNotInComment} \vee \ldots \vee \mathrm{state} = \mathrm{EndText} \wedge \mathit{Post}\}$
36 **end**
37 $\{\mathrm{state} = \mathrm{NotInComment} \wedge \mathit{PreNotInComment} \vee \ldots \vee \mathrm{state} = \mathrm{EndText} \wedge \mathit{Post}\}$

図 9 図 5 中の状態 NotInComment の処理に対する証明アウトライン
Fig. 9 The proof-outline of the state NotInComment in Fig. 5.

る証明アウトラインを**図 11** に示す．一方，状態変数 sv の導入によって **goto** 文を除去し Mills 流の構造化を行ったプログラムスキーマの対応する処理 T_i に対する証明アウトラインを**図 12** に示す．ここで，各状態 Σ_j の事前条件を $Pre\Sigma_j$ で表す．なお，終状態 Σ_{n+1} の事前条件 $Pre\Sigma_{n+1}$ はプログラム全体の事後条件 $Post$ である．

これら 2 つの証明アウトラインの対応は明らかだが，注釈除去問題に対するプログラムでの観察と同様，「構造化」したプログラムスキーマの場合のループ不変条件は，各々の状態の事前条件と状態変数がその状態値となっていることとの論理積の全状態についての論理和である．また，状態変数の値に関する論理積成分を正しく維持するため，**goto** 文を用いた場合と比べ一部の表明に余分な複雑さが入ることを余儀なくされている．また，図 12 では 5 行目の getitem(infile, v)

goto 文で状態遷移を表現した場合

```
        goto Σ1;
Σ1: S1;
    ⋮
Σn: Sn;
Σn+1: (* 終状態 *)
```

ここで，各 $1 \leq i \leq n$ について

```
Si ≡ begin
       getitem(infile, v);
       if cond_{i,1} then
        ⋮
       else if cond_{i,j} then
         begin
           F_{i,j};
           goto Σ_{i,j}
         end
        ⋮
     end
```

状態変数 sv の導入により「構造化」した場合

```
sv := Σ1;
while ¬(sv = Σn+1) do
  case sv of
    Σ1: T1;
      ⋮
    Σn: Tn
  end
```

ここで，各 $1 \leq i \leq n$ について

```
Ti ≡ begin
       getitem(infile, v);
       if cond_{i,1} then
        ⋮
       else if cond_{i,j} then
         begin
           F_{i,j};
           sv := Σ_{i,j}
         end
        ⋮
     end
```

図 10 有限状態機械モデルに基づくプログラムの一般形（**goto** 文を用いる場合と用いない場合）
Fig. 10 Program schemes based of the finite state machine model (both with and without **goto**'s).

に対し図 11 にはない余分な帰結規則の適用が必要となっている．

以上の一般形に関する比較より，有限状態機械によるモデル化に基づいたプログラムの場合，**goto** 文を用いて状態遷移を表現するプログラムの方が状態変数を導入して構造化したプログラムよりも，Hoare 論理による正しさの証明が自然に行えることが一般的に成立する．

2 つのプログラミングスタイルをの比較した以上の議論は，Hoare 論理での検証表明の比較とはいえ，主に表明が人間にとって自然な形であるか否かという主観性を免れない面があった．そこで 1 階述語論理の証明支援ツールの類を用いて純粋に機械的な証明を与えるうえでの計算コスト（時間量）の観点から両者を比較する．ここで「純粋に機械的な証明」とは完全な証明アウトライン中の証明責務を証明支援ツールによって記号的に証明することである．ここでの証明責務とは，証明アウトラインにおける帰結規則の適用（たとえば図 12 での getitem$(infile, v)$ への適用）での外側の事前条件から内側のそれへの含意（その例では 3 行目から 4 行目への含意）と内側の事後条件から外側のそれへの含意（同じく 6 行目から 7 行目への含意）とを示すことである．

図 10 左側の **goto** 文を用いたスタイルのプログラムと比べ，右側の状態変数を導入したスタイルでは新たに導入された **while** 文のループ不変条件を見出す必要がある．これは検証者が本論文の結果のとおり $\bigvee_{k=1}^{n+1} sv = \Sigma_k \wedge Pre\Sigma_k$ と与え，そのコストは考えない．ただし状態数が $n+1$（最後が終状態）のとき，この式の長さが $O(n)$ であることを注意しておく．

状態変数を導入したスタイルのプログラムに対する機械的証明では，導入した状態を表現する $n+1$ 個の定数名 $\Sigma_1, \ldots, \Sigma_{n+1}$ を規定するために，公理として

$$\bigwedge_{1 \leq i \leq n+1} \bigwedge_{j \neq i} \Sigma_i \neq \Sigma_j \qquad (\bigstar)$$

という長さ $O(n^2)$ の論理式を証明支援系に与える必要がある．この公理 ($\bigstar$) からたとえば自然演繹での $\wedge$-除去規則を用いて特定の添字対 i'，j' $(i' \neq j')$ に対する不等式 $\Sigma_{i'} \neq \Sigma_{j'}$ を結論するには $O(n^2)$ 回の $\wedge$-除去規則の適用が必要である．

さて，各 i $(1 \leq i \leq n)$ について，その状態 Σ_i での処理に対する証明アウトラインを，**goto** 文を用いたスタイルにおける S_i の場合は図 11 に，また，状態変数を用いたスタイルにおける T_i の場合は図 12 に，

```
1    {Pre\Sigma_i}
2    begin
3      {Pre\Sigma_i}
4      getitem(infile, v)
5      {Q};
6      if cond_{i,1} then
         ⋮
7      else if cond_{i,j} then
8        {Q ∧ (⋀_{k=1}^{j-1} ¬cond_{i,k}) ∧ cond_{i,j}}
9        begin
10         {Q ∧ (⋀_{k=1}^{j-1} ¬cond_{i,k}) ∧ cond_{i,j}}
11         F_{i,j}
12         {Pre\Sigma_{i,j}};
13           {Pre\Sigma_{i,j}}
14           goto \Sigma_{i,j}
15           {⊥}
16         {Pre\Sigma_{i+1}}
17       end
18       {Pre\Sigma_{i+1}}
         ⋮
19     {Pre\Sigma_{i+1}}
20   end
21   {Pre\Sigma_{i+1}}
```

図 11 **goto** 文により状態遷移を表現したプログラムスキーマの状態 Σ_i での処理 S_i に対する証明アウトライン
Fig. 11 The proof-outline for S_i of the state Σ_i of the program with **goto**'s.

それぞれ示す．両者を比較すると 2 つの点で異なる．

第 1 に，後の T_i では getitem(infile, v) に対して帰結規則が適用されている．この帰結規則の適用による証明責務を機械的に示すコストは以下のとおりである．帰結規則の適用における事前条件についての証明責務に関しては，3 行目から 4 行目への含意を示さなければならない．そのためには，3 行目の論理式を $\Sigma_i = \Sigma_i \wedge \mathit{Pre}\Sigma_i \vee \bigvee_{k \neq i} \Sigma_k = \Sigma_i \wedge \mathit{Pre}\Sigma_k$ という形に変形する必要がある．この変形には，少なくとも $O(n)$ の証明ステップ（$\wedge$ の $\vee$ に対する分配則や等号に関する公理等の適用）が必要である．さらに，この変形された論理式について，$k \neq i$ である n 個の論理和成分 $\Sigma_k = \Sigma_i \wedge \mathit{Pre}\Sigma_k$ が上の公理 (★) に矛盾していることを示すには，個々の対 k, i に対する $\Sigma_k \neq \Sigma_i$ を (★) から導くこと（$O(n^2)$ ステップ）と，$\Sigma_k \neq \Sigma_i$ と $\Sigma_k = \Sigma_i$ とから偽 $\perp$ を導くこと（$O(1)$ ステップ）と，$\perp \wedge \mathit{Pre}\Sigma_k$ から偽 $\perp$ を導くこと（$O(1)$ ステップ）とが必要である．事後条件についての証明責務は，この場合，2 つの事後条件（6 行目と 7 行目）が同一の論理式 Q であるので，$O(1)$ ステップで示せる．したがって，状態変数を用いたスタイルの場合の各 i 番目の状態の処理 T_i 各々での getitem(infile, v) への帰結規則の適用による証明責務を示すために，$O(n^3)$（個々の論理和成分に対し $O(n^2)$ で論理和成分が $n+1$ 個あるから）の証明ステップが必要である．これに相当するコストは **goto** 文を用いる S_i の検証には発生しない．

第 2 の違いは，**goto** 文を用いたスタイルの各 S_i では 12–16 行目の **goto** $\Sigma_{i,j}$ に対する帰結規則の適用があり，一方，状態変数を用いたスタイルの各 T_i では 14–18 行目の $sv := \Sigma_{i,j}$ に対する帰結規則の適用があることである．詳しくは述べないが，それらにともなう証明責務を示す時間計算量は，前者については $O(1)$ で，一方，後者については $O(n)$ 時間（17 行目の $sv = \Sigma_{i,j} \wedge \mathit{Pre}\Sigma_{i,j}$ から 18 行目の $\left(\bigvee_{k=1}^{n+1} sv = \Sigma_k \wedge \mathit{Pre}\Sigma_k\right)$ への含意の推論ステップ数は $O(1)$ だが，18 行目の論理式の長さが $O(n)$ なのでその中の論理和成分として 17 行目の論理式を見出すために $O(n)$ 時間）必要である．以上のコストは状態 Σ_i からの状態遷移の数（m_i とする）だけ必要である．ゆえに，第 2 の違いに関しては，**goto** 文を用いた S_i 特有のコストは $O(m_i)$，一方，状態変数を用いた T_i 特有のは $O(n \cdot m_i)$ となる．

プログラム全体では以上の S_i あるいは T_i が n 個存

```
1   { (⋁_{k=1}^{n+1} sv = Σ_k ∧ PreΣ_k) ∧ sv = Σ_i }
2   begin
3     { (⋁_{k=1}^{n+1} sv = Σ_k ∧ PreΣ_k) ∧ sv = Σ_i }
4       {PreΣ_i}
5       getitem(infile, v)
6       {Q}
7     {Q};
8     if cond_{i,1} then
        ⋮
9     else if cond_{i,j} then
10      { Q ∧ (⋀_{k=1}^{j-1} ¬cond_{i,k}) ∧ cond_{i,j} }
11    begin
12      { Q ∧ (⋀_{k=1}^{j-1} ¬cond_{i,k}) ∧ cond_{i,j} }
13      F_{i,j}
14      {PreΣ_{i,j}};
15        {Σ_{i,j} = Σ_{i,j} ∧ PreΣ_{i,j}}
16        sv := Σ_{i,j}
17        {sv = Σ_{i,j} ∧ PreΣ_{i,j}}
18      { ⋁_{k=1}^{n+1} sv = Σ_k ∧ PreΣ_k }
19    end
20    { ⋁_{k=1}^{n+1} sv = Σ_k ∧ PreΣ_k }
        ⋮
21    { ⋁_{k=1}^{n+1} sv = Σ_k ∧ PreΣ_k }
22  end
23  { ⋁_{k=1}^{n+1} sv = Σ_k ∧ PreΣ_k }
```

図 12 状態変数 sv の導入で構造化したプログラムスキーマの状態 Σ_i での処理 T_i に対する証明アウトライン
Fig. 12 The proof-outline for T_i of the state Σ_i of the program without **goto**'s.

在するので，状態遷移の総数 $\Sigma_{i=1}^{n} m_i$ を M と書くと，非常に素朴な証明支援系を用いる場合，**goto** 文を用いたスタイル特有の証明責務を示すためには $O(M)$ 時間が，一方，状態変数を用いたスタイル特有の証明責務を示すには $\Sigma_{i=1}^{n}(O(n^3)+O(n\cdot m_i)) = O(n^4+n\cdot M)$ 時間が，それぞれ必要である．

ただし状態変数を導入したスタイル特有の検証でのオーダ表記中の n^4 の項は第 1 の違いでの個々の対 k, i に対する $\Sigma_k \neq \Sigma_i$ を (★) から導くことを n 回ずつ（つまり，個々の $1 \leq i \leq n$ について）行っていることに起因する．この処理を共通化すればその項は n^3 とすることが可能となり，結果としてプログラム全体でも $O(n^3+n\cdot M)$ 時間で行える．さらに，以上では状態を表すのに定数名 $\Sigma_1, \ldots, \Sigma_{n+1}$ を用いるとしたので長さ $O(n^2)$ の公理 (★) を必要としたが，証明支援系に自然数が組み込まれており自然数の等否判定は $O(1)$ で行える場合に状態の表現を定数名 Σ_i でなく自然数 i そのもので行うとすれば，公理 (★) が原因の $O(n^2)$ がなくなるので，第 1 の違いでの各 T_i における証明責務を示すのに $O(n)$ ステップで済ませることができる．

以上の効率化を行えば，状態変数を用いたスタイル特有の証明責務全体は $O(n^2+n\cdot M)$ 時間で示せるが，それでも **goto** 文を用いたスタイル特有の証明責務を示す時間計算量のオーダー $O(M)$ より真に大きい．

以上より，表明の単純さ・自然さという非形式的な点でも機械的検証での時間計算量の点でも **goto** 文を用いたスタイルの方が優れていることが示された．

最後に，前章ならびに本章での検証は部分正当性に関してであったが，プログラムの停止性の証明をも含む完全正当性に関する検証について次のことを指摘しておく．**goto** 文を用いたスタイルと状態変数を導入し **goto** 文を除去したスタイルとでは，停止性の証明はまったく同じ方法で同じ難しさで行える．停止性の証明は，値がつねに非負であることが保証される式——以下「帰納式」と呼ぶ——を選び，その式の値が，ループの場合には各反復ごとに，また，**goto** 文を用いている場合には各飛び越しごとに，必ず減少することを示すのが常套手段である．そのような式としては，前章の例題の場合には入力ファイルの残りの部

分 $\mathrm{in_R}$ の長さを選べばよく，図 4 も図 5 も同一の表明付けにより各々の停止性を証明できる．一般形としての図 10 左右の両プログラムスキーマに関しても，**while** ループの 1 回の反復と **goto** 文による 1 回の飛び越しとが正確に対応しており，また，状態変数 sv への終状態名 Σ_{n+1} の代入文と終状態ラベルへの飛び越し文 **goto** Σ_{n+1} とが対応する箇所に必ず現れるので，両者の停止性の証明が同一の帰納式を用いて同じ形の表明付けで行えるのは明らかである．

5. まとめ——goto 文除去のための変数導入の問題

本論文で議論の対象とした有限状態機械モデルに基づくプログラミングの場合，図 4 のような **goto** 文を用いるプログラマよりも図 5 の「構造化」したプログラムを作る方が多数派であろう．Plauger が彼のエッセイで注釈除去問題での **goto** 文使用の是非を取り上げたのも，そういった「常識」がプログラマの間で広まっていること☆に対する警鐘ととらえることができる．Plauger 自身も，彼のエッセイ（訳書，p.45）で有限状態機械をシミュレートする様々なプログラムすべてでの共通に見られるスタイルとして，状態変数の導入とそれに基づく **while** 文およびそのループ本体が状態変数の値による場合分けであることを指摘している．

本論文では，Hoare 論理によるプログラムの正しさの証明の立場からすると，上の「常識」とは反し，有限状態機械に基づくプログラミングの場合には Mills 流の構造化を行わずに **goto** 文で状態遷移を表現したプログラムの方が表明が単純で自然な形にできる（と同時に機械的検証でも時間的計算量が小さい）ことを示した．

著者らは別稿[12] でループからの脱出の場合には **goto** 文を用いた方が Boolean 型変数を導入して **goto** 文を用いないよりも Hoare 論理による検証での表明が単純で自然な形であることを示した．本論文で述べた有限状態機械モデルからのプログラミングでの **goto** 文の除去のための状態変数の導入も，ループからの脱出で **goto** 文を回避するための Boolean 型変数の導入も，Mills ら[13] が「構造定理」（Structure Theorem, 同書 §4.4.3）——すなわち任意の逐次的プログラムの制御構造は変数 1 個の追加で 3 基本構造のみで表現できること——と呼び，彼らの構造的プログラミング提唱の基盤とした一般的結果の事例に該当する．

しかし，別稿[12] でのループ脱出の場合や本論文の例題およびプログラムスキーマの検証での観察のとおり，変数の導入による **goto** 文の除去は表明を複雑にし機械的検証でのコストも増大させる結果となっており，検証の立場からすると本末転倒である．この結果は，Dijkstra が最初の CACM の手紙の末尾で指摘した機械的な構造化の無意味さを裏付けている．

Hoare 論理による検証での表明が **goto** 文を用いた場合に比べ，**goto** 文を除去するために変数を導入した場合がなぜ複雑になるのか，という原因は次のようにまとめることができる．**goto** 文とその飛び先ラベルの対応は**静的**に定まっている．一方，導入された変数の値は代入によって**動的**にしか決らない．その結果，**検証の立場からは，静的に定まっていた情報に関する表明が動的にしか決められない情報に関する表明へと変化せざるをえない．これが表明の複雑化の原因である**．

構造的プログラミングとは何かを研究するうえでプログラムの正当性や検証のことも考察すべきだ，ということは Gries[8] も指摘している．検証方法を背景としたプログラミングスタイルの検討が正しさを示しやすく信頼性の高いプログラムを書くうえで重要であり，そのような検討を積み重ねることこそが構造的プログラミングの原点としての Dijkstra 本来の意図だと著者らは考えるが，そのような方向での研究は著者らの知る限りでは行われてこなかった．本論文はその方向への非常に小さな一歩を目指し，有限状態機械モデルに基づくプログラミングでのより良いスタイルについて Hoare 論理という科学的立場からの根拠を与えた．

本論文の結果は，たとえば並行プログラムの場合のように状態空間が直積となっている場合でも，その状態空間を直積成分に分解し個々の直積成分ごとに図 3 と同様の有限状態機械の状態遷移図としてモデル化しプログラミングする場合には適用できることを注意しておく．

最後に，本論文の結果が実用的な面へ提言できることは次のとおりである．有限状態機械モデルを用いてプログラム設計を行う機会は少なくない．特に，GUI 等のイベント駆動的なプログラムの開発では，状態遷

☆ 実際，第 1 著者が彼の職場で 10 年以上ソフトウェア研究開発に従事してきた周囲の 5 人の技術者に 3 章の例題をプログラミングするとしたら図 4 と図 5 とのどちらのスタイルで書くか，または別のスタイルで書くか，とインタビューしたところ，全員が図 5 に相当するスタイルと答えた．5 人では統計的に意味のある結論を出せないが，有限状態機械としてモデル化してからのプログラミングでは，図 5 や図 10 右側に示した状態変数を導入し **goto** 文を除去した Mills 流の構造的スタイルがソフトウェア技術者にある程度は普及しているといっても間違いではなかろう．

移を設計し開発ツールによりコードスケルトンを生成することが多い．その自動生成されたコードスケルトンは巨大なループと巨大な **case** 文を用いた図 5 風の構造化された形である．しかし，自動生成のためにツールに入力した設計情報は失われソースコードだけが残っているという現実のプログラム保守でよく見受けられる状況では，そのようにして開発された巨大ループと巨大 **case** 文とからなるプログラムのスタイルは，**goto** 文で状態遷移を表したスタイルと比べ，(非形式的にせよ）論理的に内容を理解し正しさを保証しようとする立場からは障害となる危険性がある．そのことが本論文の結果より分かる．

謝辞 大槻繁氏（現エクイティ・リサーチ取締役．元（株）日立製作所システム開発研究所主任研究員）は，書籍の執筆等の機会を通じ，著者にクリーンルーム手法・ホーア論理・プログラム検証について考察する契機を与え，また励まし続けてくださいました．野木兼六氏（現神奈川工科大学教授．元（株）日立製作所基礎研究所主任研究員）は Hoare 論理のみならずソフトウェア工学やプログラミング理論全般に関して様々な議論を通じ忍耐強く導き続けてくださいました．田辺裕一氏（(株）日立製作所ソフトウェア事業部技師）には，Plauger が彼の著書で状態遷移に基づくプログラムでの **goto** 文使用の問題を取り上げていることを教えていただきました．これらの方々に心より感謝いたします．

参 考 文 献

1) Böhm, C. and Jacopini, G.: Flow Diagrams, Turing Machines and Languages with Only Two Formation Rules, *Comm. ACM*, Vol.9, No.5, pp.366–371 (1966).
2) de Bruin, A.: **Goto** Statements: Semantics and Deduction Systems, *Acta Inf.*, Vol.15, pp.385–424 (1981).
3) Clint, M. and Hoare, C.A.R.: Program Proving: Jumps and Functions, *Acta Inf.*, Vol.1, pp.214–224 (1972).
4) Dahl, O.-J., Dijkstra, E.W.D. and Hoare, C.A.R.: *Structured Programming*, Academic Press (1972). 野下，川合，武市（訳）：構造化プログラミング，サイエンス社 (1975).
5) Dijkstra, E.W.D.: Goto Statement Considered Harmful, *Comm. ACM*, Vol.11, No.3, pp.147–148 (1968). 木村（訳）：**go to** 文有害説，*bit*, Vol.7, No.5, pp.346–378 (1975).
6) Dijkstra, E.W.D.: Notes on Structured Programming, in *Structured Programming*, Dahl, O.-J., et al. (Eds.), Chapter 1, pp.1–82 (1972).
7) Dijkstra, E.W.D.: Structured Programming, *Software Engineering: Concepts and Techniques*, Naur, P. et al. (Eds.), pp.222–226, Petrocelli/Charter (1976).
8) Gries, D.: On Structured Programming, *Comm. ACM*, Vol.17, No.11, pp.655–657 (1974).
9) 林 晋：プログラム検証論，共立出版 (1995).
10) Hoare, C.A.R.: An Axiomatic Basis for Computer Programming, *Comm. ACM*, Vol.12, No.10, pp.576–580, 583 (1969).
11) Hoare, C.A.R. and Wirth, N.: An Axiomatic Definition of Programming Language PASCAL, *Acta Inf.*, Vol.2, No.4, pp.335–355 (1973).
12) 金藤栄孝，二木厚吉：多重ループからの脱出での **goto** 文の是非：Hoare 論理の観点から，情報処理学会論文誌，Vol.45, No.3, pp.770–784 (2004).
13) Linger, R.C., Mills, H.D. and Witt, B.I.: *Structured Programming: Theory and Practice*, Addison-Wesley (1979).
14) Plauger, P.J.: *Programming on Purpose: Essays on Software Design*, Essay 4, Prentice Hall (1993). 安藤（訳）：プログラミングの壷 I—ソフトウェア設計編，pp.42–50, 共立出版 (1995).

(平成 15 年 5 月 27 日受付)
(平成 16 年 4 月 5 日採録)

金藤 栄孝（正会員）
1981 年東京大学大学院理学系研究科博士課程中退．同年（株）日立製作所入社．プログラミング言語の意味論・型理論・ソフトウェア工学（なかでも形式手法等の高信頼化技術）に関心を持つ．EATCS，ACM，IEEE-CS，ソフトウェア科学会各会員．

二木 厚吉（正会員）
1975 年東北大学大学院工学研究科博士課程修了．工学博士．同年電子技術総合研究所（電総研）入所．1983 年〜1984 年 SRI International 客員研究員．1992 年電総研主席研究官．1993 年北陸先端科学技術大学院大学情報科学研究科教授．主な研究分野：フォーマルメソッド，プログラミング方法論，言語設計学，ソフトウェア工学．ACM，IEEE-CS，ソフトウェア科学会各会員．