プログラム操作のためのホーア論理の拡張

森口 草介

本論文では、ソースコードをユーザ定義のデータ型によって表現し、編集をそのデータ型に対する操作として捉えることで、ソースコードを編集するプログラムに対する、ホーア論理を用いた検証手法を提案する。ソースコードを編集するプログラムは、コード生成からコンパイラまで広く存在する。編集対象のソースコードは、操作するプログラムからはただのデータに過ぎず、プログラムの誤りにより意図しない形に編集してしまう。提案手法では、編集する対象に関するホーアの三つ組を記述できるようにすることで、編集の前後におけるソースコードを検証し、それを通じて編集するプログラムの検証を行う。編集対象のホーアの三つ組は、意味上は他の表明と同様に扱われるため、これ自体には論理体系の大きな変更を必要としない。しかし、三つ組自体が成立するかは少なくとも通常の検証と同程度に難しく、これを検証するための手法が必要である。編集対象であるソースコードは操作する側のパラメータとして表現されるため、検証時にパラメータで表された複数のプログラム間の関係を議論しなければならない。このために、ホーア論理の拡張として、三つ組同士の関係を取り扱う体系を考える。

In this paper, we propose an extension of Hoare logic for source code manipulations. There are so many programs whose domain is a source code, e.g., code generators, compilers, and so on. In these programs, the target codes are denoted as data. Our method is a framework for viewing such data as other programs and describing the specifications of the data as behaviors of corresponging programs. These data are modified in execution, so our method introduces the rules for the data with variables of the programs.

1 はじめに

ソースコードを編集するプログラムは、コード生成器やコンパイラなど広く存在する。コード生成では与えられたデータに基づくプログラムの生成や、部分的なテンプレートの生成などがある。また、コンパイラやプログラム解析では入力としてソースコードを用い、別のデータを出力とする。以下、このように別のソースコードを生成、解析、編集することをプログラム操作と呼び、操作されるプログラムを対象プログラムと呼ぶ。

対象プログラムは、操作するプログラムからはただのデータに過ぎず、誤りにより意図しない形に編集される可能性がある。例えば、コンパイラでのバグとして、特定のソースコードに対して最適化前後で異なる結果を出す、といったものも多く報告されている。最適化などのプログラム操作における仕様は、操作前後の（または出力した）プログラムの挙動に関するものである。そのため、プログラム操作に関して検証を行うには、対象プログラムの挙動を記述できる必要がある。

本論文では、対象プログラムをユーザ定義のデータ型によって表現し、プログラム操作をそのデータ型に対する操作として記述したプログラムに対する、ホーア論理を用いた検証手法を提案する。提案手法では、対象プログラムに関するホーアの三つ組を直接記述できるようにすることで、仕様の記述を可能にする。しかし、記述されたホーアの三つ組は、プログラムを対象としている以上、少なくとも通常のホーアの三つ組と同程度に示すことが難しい。さらに、対象プログラムは操作するプログラムの変数として格納されている場合もあり、その結果を基に編集したプログラムとの関係を推論する必要もある。そこで、推論規則を

Extension of Hoare Logic for Program Manipulations
Sosuke Moriguchi, 関西学院大学, Kwansei Gakuin University.

拡張し、対象プログラムに関する三つ組を検証できるようにする。この規則では、意味論による結果と推論規則を組み合わせることで、既存の手法を利用して検証を行う。

本論文の構成は以下の通りである。2 節では、プログラムを記述する言語として単純な手続き型言語である While 言語にデータ型の概念を追加したものについて述べる。3 節では、仕様を記述するための表明言語に関して述べる。4 節では、基礎となる推論規則に関して述べる。5 節では、6 節では関連研究について述べ、最後に 7 節でまとめる。

2 対象言語

本論文では、簡易な手続き型言語である While 言語を用いる。ここでは、データ構造を作るためコンストラクタを持つものとする。また、このコンストラクタを展開するために文構造としてパターンマッチを導入する。構文は図 1 の通りである。

パターンマッチの文構造は、関数型言語に見られるパターンマッチに対して以下の制約を置いたものとなっている。

1. パターンマッチの対象が式ではなく単一の変数である。 この制約は直前に代入を行うことで容易に回避できる。
2. コンストラクタに関する宣言がないため、パターンの網羅性などが判定できない。 今回は宣言やコンストラクタとパターンのパラメータ数、網羅性などについては議論せず、適切に記述されているものとする。
3. 一つのパターンマッチにおいてそれぞれのパターンにおけるコンストラクタは相異なるものとする。
4. パターンマッチの対象および各パターン内の変数は全て相異なる変数とする。

意味論は図 2 のようになる。対象言語である While 言語には局所変数の概念を入れていないため、パターンマッチでは単純な代入が行われている。

次のプログラムは、nの初期値に応じてコンストラクタをネストして構造を作っている。

```
p := assign_s("x",
               add(var("x"),int(10)));
if n <= 0 then s := skip_s()
else s := assign_s("x", int(10));
     n := n - 1
end;
while 0 < n do
  s := seq_s(s, p);
  n := n - 1
end
```

例えば、nの初期値が 3 のとき、sには seq_s(seq_s(p, p), assign_s("x", int(10)))（ただし p=assign_s("x", add(var("x"), int(10))）が格納され、−1 のとき、sには skip_s()が格納される。

また、次のプログラムは、いくつかのコンストラクタで作られたデータに対して、パターンマッチを用いることで内部の情報を解析する例である。概要としては「assign_sの第一引数に nameに格納された値と同じものが格納されているようなものの数」を数えるものである。

```
count := 0;
rest := skip_s();
flag := 1;
while flag = 1 do
  match prog with
  | skip_s() =>
      prog := rest;
      rest := skip_s()
  | assign_s(v, a) =>
      if v = name
       then count := count + 1
       else skip end;
      prog := skip_s()
  | seq_s(s1, s2) =>
      prog := s1;
      rest := seq_s(s2, rest)
  end;
  if prog = skip_s()
    and rest = skip_s()
   then flag = 0
   else skip end
end
```

$$
\begin{array}{lcl}
v & ::= & n \mid \text{"}string\text{"} \mid c(v_1, v_2, \ldots) \\
E & ::= & v \mid x \mid E \texttt{+} E \mid E \texttt{-} E \mid E \texttt{*} E \mid c(E_1, E_2, \ldots) \\
b & ::= & \texttt{true} \mid \texttt{false} \\
B & ::= & b \mid E \texttt{=} E \mid E \texttt{<>} E \mid E \texttt{<=} E \mid E \texttt{<} E \mid B \ \texttt{and}\ B \mid B \ \texttt{or}\ B \mid \texttt{not}\, B \\
S & ::= & \texttt{skip} \mid x \texttt{:=} E \mid S \texttt{;} S \mid \texttt{if}\ B\ \texttt{then}\ S\ \texttt{else}\ S\ \texttt{end} \mid \texttt{while}\ B\ \texttt{do}\ S\ \texttt{end} \\
 & & \mid \texttt{match}\ x\ \texttt{with} \mid c(x_1, x_2, \ldots) \texttt{=>} S \mid \ldots\ \texttt{end}
\end{array}
$$

図 1　While 言語の構文。v が値、E が式、b がブール定数、B がブール式、S が文。

なお、再帰的な構造の走査を単純な繰り返しにするため、restに対象プログラムの残りを格納している。

次節にて述べるが、これらはそれぞれ「nの初期値が 0 以上のとき、その値に 10 をかけたものを変数xに格納するプログラムの生成」「progが表すプログラムにおいて、nameに入った名前の変数へ代入する文の数え上げ」に相当する。ただし、コードが長くなるため、後者における対象プログラムは代入文と skip 文を並べただけのものとなっている。

3　表明言語

表明言語は図 3 の通りである。ここで、f はユーザ定義の、値から表明への全域関数である。なお、本論文では、f を値の構造に関する再帰関数の形で定義する。

表明言語の定義において特徴的なのは以下の点である。

1. 表明言語中の変数にラベルが付与されている。表明中に文構造があり、文構造内の変数と外側の変数を区別するために用いられる。
2. 文を生成する関数 $\mathcal{F}$ が存在する。

以降、While 言語の変数（ラベルが付与されていない）と区別するために、表明言語の変数（ラベルが付与されている）をラベル付き変数と呼ぶ。通常は While 言語の変数を単に変数と呼ぶが、差を強調する場合には、While 言語の変数をラベル無し変数と呼ぶこともある。

$\mathcal{F}$ は与えられた式をプログラムに変換する関数である。これにより、対象プログラムを操作するプログラムにおけるデータと対応付けることが可能となる。ここでは、構文に対応するコンストラクタを用いて、以下のように定義する。

$$
\begin{array}{lcl}
\mathcal{F}(\texttt{skip\_s}()) & = & \texttt{skip} \\
\mathcal{F}(\texttt{assign\_s}(\text{"}str\text{"}, a)) & = & str := \mathcal{A}(a) \\
\mathcal{F}(\texttt{seq\_s}(s_1, s_2)) & = & \mathcal{F}(s_1); \mathcal{F}(s_2) \\
\mathcal{F}(\texttt{if\_s}(b, s_1, s_2)) & = & \texttt{if}\ \mathcal{B}(b)\ \texttt{then}\ \mathcal{F}(s_1) \\
 & & \texttt{else}\ \mathcal{F}(s_2)\ \texttt{end}
\end{array}
$$

この中では $\mathcal{B}$ や $\mathcal{A}$ を利用している。それぞれブール式や式を生成する関数であるが、ここでは詳細を省略する。$\mathcal{F}$ から while 文やパターンマッチ文を生成する場合が欠落しているが、本論文では用いず、またほぼ if 文の場合と同じであるため、今回は省略している。

3.1　表明の意味論

表明はラベル付き変数の状態に対して妥当かを判定する。意味論は図 4 のように定義されるが、以下の記法を用いている。

- σ はラベル付き変数に関する環境（ラベル付き変数から値への関数）である。
- E がラベル付きの式であるとき、$E[\sigma]$ は各ラベル付き変数を σ によって値に置換した While 言語の式である。$S[\sigma]$ も同様である。
- $\sigma_1^l \cdot \sigma_2(x^{l'}) = \begin{cases} \sigma_1(x^{l'}) & (\text{if } l = l') \\ \sigma_2(x^{l'}) & (\text{otherwise}) \end{cases}$ と定義する。
- $\overline{\sigma}^l$ はラベル無し変数に関する環境であり、$\overline{\sigma}^l(x) = \sigma(x^l)$ とする。

意味論での記述に ϵ が用いられているが、これは空の環境である。実際には、$E[\sigma]$ によって変数がすべて値に置換されているため、式の評価は環境の影響を受けず、任意の環境で良い。

$$v, \sigma \Downarrow_E v \qquad x, \sigma \Downarrow_E \sigma(x) \qquad \frac{E_1, \sigma \Downarrow_E n_1 \quad E_2, \sigma \Downarrow_E n_2}{E_1 \texttt{ + } E_2, \sigma \Downarrow_E n_1 + n_2} \qquad \frac{E_1, \sigma \Downarrow_E n_1 \quad E_2, \sigma \Downarrow_E n_2}{E_1 \texttt{ - } E_2, \sigma \Downarrow_E n_1 - n_2}$$

$$\frac{E_1, \sigma \Downarrow_E n_1 \quad E_2, \sigma \Downarrow_E n_2}{E_1 \texttt{ * } E_2, \sigma \Downarrow_E n_1 * n_2} \qquad \frac{E_1, \sigma \Downarrow_E v_1 \quad E_2, \sigma \Downarrow_E v_2 \quad \ldots}{c(E_1, E_2, \ldots), \sigma \Downarrow_E c(v_1, v_2, \ldots)}$$

$$b, \sigma \Downarrow_B b \qquad \frac{E_1, \sigma \Downarrow_E v \quad E_2, \sigma \Downarrow_E v}{E_1 \texttt{ = } E_2, \sigma \Downarrow_B \texttt{true}} \qquad \frac{E_1, \sigma \Downarrow_E v_1 \quad E_2, \sigma \Downarrow_E v_2 \quad v_1 \neq v_2}{E_1 \texttt{ = } E_2, \sigma \Downarrow_B \texttt{false}}$$

$$\frac{E_1, \sigma \Downarrow_E v_1 \quad E_2, \sigma \Downarrow_E v_2 \quad v_1 \neq v_2}{E_1 \texttt{ <> } E_2, \sigma \Downarrow_B \texttt{true}} \qquad \frac{E_1, \sigma \Downarrow_E v \quad E_2, \sigma \Downarrow_E v}{E_1 \texttt{ <> } E_2, \sigma \Downarrow_B \texttt{false}}$$

$$\frac{E_1, \sigma \Downarrow_E n_1 \quad E_2, \sigma \Downarrow_E n_2 \quad n_1 \leq n_2}{E_1 \texttt{ <= } E_2, \sigma \Downarrow_B \texttt{true}} \qquad \frac{E_1, \sigma \Downarrow_E n_1 \quad E_2, \sigma \Downarrow_E n_2 \quad n_1 > n_2}{E_1 \texttt{ <= } E_2, \sigma \Downarrow_B \texttt{false}}$$

$$\frac{E_1, \sigma \Downarrow_E n_1 \quad E_2, \sigma \Downarrow_E n_2 \quad n_1 < n_2}{E_1 < E_2, \sigma \Downarrow_B \texttt{true}} \qquad \frac{E_1, \sigma \Downarrow_E n_1 \quad E_2, \sigma \Downarrow_E n_2 \quad n_1 \geq n_2}{E_1 < E_2, \sigma \Downarrow_B \texttt{false}}$$

$$\frac{B_1, \sigma \Downarrow_B \texttt{true} \quad B_2, \sigma \Downarrow_B \texttt{true}}{B_1 \texttt{ and } B_2, \sigma \Downarrow_B \texttt{true}} \qquad \frac{B_1, \sigma \Downarrow_B \texttt{false} \quad B_2, \sigma \Downarrow_B \texttt{false}}{B_1 \texttt{ or } B_2, \sigma \Downarrow_B \texttt{false}}$$

$$\frac{B_1, \sigma \Downarrow_B b_1 \quad B_2, \sigma \Downarrow_B b_2 \quad b_1 \text{ か } b_2 \text{ の少なくとも一方が } \texttt{true} \text{でない}}{B_1 \texttt{ and } B_2, \sigma \Downarrow_B \texttt{false}}$$

$$\frac{B_1, \sigma \Downarrow_B b_1 \quad B_2, \sigma \Downarrow_B b_2 \quad b_1 \text{ か } b_2 \text{ の少なくとも一方が } \texttt{true}}{B_1 \texttt{ or } B_2, \sigma \Downarrow_B \texttt{true}}$$

$$\frac{B, \sigma \Downarrow_B \texttt{true}}{\texttt{not } B, \sigma \Downarrow_B \texttt{false}} \qquad \frac{B, \sigma \Downarrow_B \texttt{false}}{\texttt{not } B, \sigma \Downarrow_B \texttt{true}}$$

$$\texttt{skip}, \sigma \Downarrow \sigma \qquad \frac{E, \sigma \Downarrow_E v}{x \texttt{ := } E, \sigma \Downarrow \sigma[v/x]} \qquad \frac{S_1, \sigma \Downarrow \sigma_1 \quad S_2, \sigma_1 \Downarrow \sigma_2}{S_1 \texttt{ ; } S_2, \sigma \Downarrow \sigma_2}$$

$$\frac{B, \sigma \Downarrow_B \texttt{true} \quad S_1, \sigma \Downarrow \sigma'}{\texttt{if } B \texttt{ then } S_1 \texttt{ else } S_2 \texttt{ end}, \sigma \Downarrow \sigma'} \qquad \frac{B, \sigma \Downarrow_B \texttt{false} \quad S_2, \sigma \Downarrow \sigma'}{\texttt{if } B \texttt{ then } S_1 \texttt{ else } S_2 \texttt{ end}, \sigma \Downarrow \sigma'}$$

$$\frac{B, \sigma \Downarrow_B \texttt{true} \quad S, \sigma \Downarrow \sigma_1 \quad \texttt{while } B \texttt{ do } S \texttt{ end}, \sigma_1 \Downarrow \sigma_2}{\texttt{while } B \texttt{ do } S \texttt{ end}, \sigma \Downarrow \sigma_2} \qquad \frac{B, \sigma \Downarrow_B \texttt{false}}{\texttt{while } B \texttt{ do } S \texttt{ end}, \sigma \Downarrow \sigma}$$

$$\frac{x, \sigma \Downarrow_E c_k(v_1, v_2, \ldots) \quad S_k, \sigma[v_1/x_1, v_2/x_2, \ldots] \Downarrow \sigma'}{\texttt{match } x \texttt{ with} \mid \ldots \mid c_k(x_1, x_2, \ldots) \texttt{ => } S_k \mid \ldots \texttt{ end}, \sigma \Downarrow \sigma'}$$

図 2　While 言語の意味論。σ は変数環境であり、変数から値への関数である。$\Downarrow_E$ が式の評価、$\Downarrow_B$ がブール式の評価、$\Downarrow$ が文の評価をそれぞれ表す。

3.2　ホーアの三つ組

P_g、Q_g をそれぞれ g のみをラベルとして持つ表明、S を While 言語の文としたとき、$\{P_g\}\, S\, \{Q_g\}$ をホーアの三つ組（または単に三つ組）と呼ぶ。ホーアの三つ組と表明における構文としての三つ組を区別するため、後者を三つ組の表明と呼ぶ。

三つ組の意味論は、言語と表明の意味論を用いて以下のように定義される。

$$\frac{\forall \sigma_1\, \sigma_2, \sigma_1 \vDash P_g \quad \Rightarrow \quad S, \overline{\sigma_1}^g \Downarrow \overline{\sigma_2}^g \quad \Rightarrow \quad \sigma_2 \vDash Q_g}{\vDash \{P_g\}\, S\, \{Q_g\}}$$

この記述は、図 4 における三つ組の意味論と非常に似ている。P_g と Q_g は g 以外にラベルが出現しないため、表明の三つ組における意味論と同様の式と考えたとき、既存の環境である σ に依存した変数が現れない。この意味で、既存の環境を補うことで、両者を同一と見なせる。

$$
\begin{array}{lll}
E_L & ::= & v \mid x^l\ (l \in L) \mid A \mid E_L + E_L \mid E_L - E_L \mid E_L * E_L \mid c(E_L, E_L, \ldots) \\
S_L & ::= & \texttt{skip} \mid x := E \mid S_L ; S_L \mid \texttt{if}\ B\ \texttt{then}\ S_L\ \texttt{else}\ S_L\ \texttt{end} \mid \texttt{while}\ B\ \texttt{do}\ S_L\ \texttt{end} \\
 & \mid & \texttt{match}\ x\ \texttt{with} \mid c(x_1, x_2, \ldots) \Rightarrow S \mid \ldots\ \texttt{end} \\
 & \mid & \mathcal{F}(E_L) \\
P_L, Q_L & ::= & b \mid E_L = E_L \mid E_L \neq E_L \mid E_L \leq E_L \mid E_L < E_L \mid \neg P_L \\
 & \mid & P_L \wedge Q_L \mid P_L \vee Q_L \mid \texttt{constr}(E_L, c) \mid [\{P_{L,l}\}\, S_L\, \{Q_{L,l}\}]^l\ (l \notin L) \\
 & \mid & f(E_L)
\end{array}
$$

図 3　表明言語の構文。L がラベルの列、E_L がラベル付き変数を含む式、S_L が変数を用いた文、P_L が表明をそれぞれ表す。

$$
\sigma \models \mathit{true}
\qquad
\frac{E_1[\sigma], \epsilon \Downarrow_E v \qquad E_2[\sigma], \epsilon \Downarrow_E v}{\sigma \models E_1 = E_2}
\qquad
\frac{E_1[\sigma], \epsilon \Downarrow_E v_1 \qquad E_2[\sigma], \epsilon \Downarrow_E v_2 \qquad v_1 \neq v_2}{\sigma \models E_1 neq E_2}
$$

$$
\frac{E_1[\sigma], \epsilon \Downarrow_E v_1 \qquad E_2[\sigma], \epsilon \Downarrow_E v_2 \qquad v_1 \leq v_2}{\sigma \models E_1 \leq E_2}
\qquad
\frac{E_1[\sigma], \epsilon \Downarrow_E v_1 \qquad E_2[\sigma], \epsilon \Downarrow_E v_2 \qquad v_1 < v_2}{\sigma \models E_1 < E_2}
$$

$$
\frac{\sigma \models P \qquad \sigma \models Q}{\sigma \models P \wedge Q}
\qquad
\frac{\sigma \models P}{\sigma \models P \vee Q}
\qquad
\frac{\sigma \models Q}{\sigma \models P \vee Q}
$$

$$
\frac{E[\sigma], \epsilon \Downarrow_E c(v_1, v_2, \ldots)}{\sigma \models \texttt{constr}(E, c)}
\qquad
\frac{E[\sigma], \epsilon \Downarrow_E v \qquad \sigma \models f(v)}{\sigma \models f(E)}
$$

$$
\frac{\forall \sigma_1\, \sigma_2, \sigma_1'' \cdot \sigma \models P \quad \Rightarrow \quad S[\sigma], \overline{\sigma_1}^{l''} \Downarrow \overline{\sigma_2}^{l''} \quad \Rightarrow \quad \sigma_2'' \cdot \sigma \models Q}{\sigma \models [\{P\}\, S\, \{Q\}]^{l''}}
$$

図 4　表明の意味論

4　推論規則

ここでは、ホーアの三つ組のための推論規則を定義する。本論文で用いる規則は図 5 のように定義される。この中で B^l という表現を用いているが、これは B の中の変数にラベル l を付けたものである。

変数のラベルの付け外しがあることやパターンマッチ構文のための規則は特徴的だが、その他は広く用いられている規則である。そのため、以下の健全性の証明も、ほぼ通常のホーア論理の証明同様である。

Theorem 4.1. 推論規則は健全である。つまり、$\vdash_g \{P\}\, S\, \{Q\}$ ならば $\models \{P\}\, S\, \{Q\}$ である。

証明　推論規則に基づく帰納法による。パターンマッチ以外については教科書通りであるため、省略する。

パターンマッチに関しては、意味論側の前提として $\sigma_1 \models \bigvee_{i \in \{1,\ldots,n\}} \texttt{constr}(x^g, c_i)$ が成り立つため、ある i が存在して、$\overline{\sigma_1}^g(x) = \sigma_1(x^g) = c_i(v_1, v_2, \ldots)$ となる。また、前提として $\overline{\sigma_1}^g$ という環境からパターンマッチ文を実行すると $\overline{\sigma_2}^g$ となる。文の意味論は反転補題が成り立つことと合わせ、環境を $\overline{\sigma_1}^g[v_1/x_{i1}, v_2/x_{i2}, \ldots]$ として S_i を実行すると、$\overline{\sigma_2}^g$ という環境となることが示せる。この環境は $\overline{\sigma_1}^g[v_1/x_{i1}, v_2/x_{i2}, \ldots] = \overline{\sigma_1[v_1/x_{i1}^g, v_2/x_{i2}^g, \ldots]}^g$ を満たす。これを $\sigma = \sigma_1[v_1/x_{i1}^g, v_2/x_{i2}^g, \ldots]$ とすると、$\overline{\sigma}^g(x) = c_i(v_1, v_2, \ldots)$ かつ $\overline{\sigma}^g(x_{ij}) = v_j$ となる。また、P には $x_1, x_2, \ldots$ は含まれないため、$P[\sigma] = P[\sigma_1]$ である。したがって、$\sigma \models x^g = c_i(x_{i1}^g, x_{i2}^g, \ldots) \wedge P$ を満たす。これらと帰納法の仮定から、$\sigma_2 \models Q$ を満たすことが示される。　□

5　推論による三つ組の取り扱い

実際にこの推論規則を用いて検証をする場合、問題が存在する。図 5 に示した推論規則では、三つ組の表明を示すために意味論を用いている。しかし、3.2 節で述べたとおり、三つ組の表明に関する意味論はほぼ三つ組の意味論と同様であることから、三つ組の表明が妥当であることを示すことは、少なくとも三つ組を示すのと同程度に難しいことがわかる。

$$\frac{P \Rightarrow P' \quad \vdash_l \{P'\}\, S\, \{Q'\} \quad Q' \Rightarrow Q}{\vdash_l \{P\}\, S\, \{Q\}} \qquad \vdash_l \{P\}\ \mathbf{skip}\ \{P\} \qquad \vdash_l \{P[x^l \mapsto E^l]\}\ x\mathbf{:=}E\ \{P\}$$

$$\frac{\vdash_l \{P\}\, S_1\, \{Q\} \quad \vdash_l \{Q\}\, S_2\, \{R\}}{\vdash_l \{P\}\, S_1; S_2\, \{R\}} \qquad \frac{\vdash_l \{P \wedge B^l\}\, S_1\, \{Q\} \quad \vdash_l \{P \wedge \mathtt{not}\ B^l\}\, S_2\, \{Q\}}{\vdash_l \{P\}\ \mathbf{if}\ B\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2\ \mathbf{end}\ \{Q\}}$$

$$\frac{\vdash_l \{P \wedge B^l\}\, S\, \{P\}}{\vdash_l \{P\}\ \mathbf{while}\ B\ \mathbf{do}\ S\ \mathbf{end}\ \{P \wedge \mathtt{not}\ B^l\}}$$

$$\frac{\forall i \in \{1,\ldots,n\}, P \text{ does not contain } x^l_{i1}, x^l_{i2}, \ldots \qquad \vdash_l \{x^l = c_i(x^l_{i1}, x^l_{i2}, \ldots) \wedge P\}\, S_i\, \{Q\}}{\vdash_l \{P \wedge \bigvee_{i\in\{1,\ldots n\}} \mathtt{constr}(x^l, c_i)\}\ \mathtt{match}\ x\ \mathtt{with} \mid \ldots \mid c_i(x_{i1}, x_{i2}, \ldots) \Rightarrow S_i \mid \ldots \mathbf{end}\ \{Q\}}$$

図 5　ホーア論理の推論規則

例えば 2 節で挙げた乗算の例では、while 文による繰り返しによって、文を表すデータが繰り返しの中で大きくなるが、文を実行した結果に関する不変条件を記述し、それが不変条件であることを確かめることになる。実際に、nの初期値を N として、以下の三つ組を考える。

$$\{\mathtt{n}^g = N \wedge 0 < \mathtt{n}^g\}\, S\, \{[\{true\}\, \mathcal{F}(s^g)\, \{x^l = N * 10\}]^l\}$$

読みやすさのために対象のプログラムは S としている。

S には while 文が含まれるため、意味論によってこの三つ組が成り立つかを議論することは現実的でない。推論規則を用いて while 文の範囲に限定すると、推論する三つ組は以下のようになる。ただし、pの等式に関しては定数であるため記述の都合上省略した。

$$\{\mathtt{p}^g = \ldots \wedge \mathtt{n}^g = N - 1 \wedge 0 \leq \mathtt{n}^g \wedge \mathtt{s}^g = \mathtt{assign\_s}(\text{"}x\text{"}, \mathtt{int}(10))\}$$
$$\mathbf{while} \ldots \mathbf{end}$$
$$\{[\{true\}\, \mathcal{F}(s^g)\, \{x^l = N * 10\}]^l\}$$

ここで while 文の不変条件を次のようにする。

$$\mathtt{p}^g = \ldots \wedge \wedge 0 \leq \mathtt{n}^g \wedge [\{true\}\, \mathcal{F}(s^g)\, \{x^l = (N - \mathtt{n}^g) * 10\}]^l$$

この不変条件を用いることで while 文の事前条件、事後条件については容易に導出できるため、while 文の本体についての推論を行う。代入文が連続しているので、最弱事前条件をもとに検証する式は以下の通りになる。

$$\mathtt{p}^g = \ldots \wedge \wedge 0 < \mathtt{n}^g \wedge [\{true\}\, \mathcal{F}(s^g)\, \{x^l = (N - \mathtt{n}^g) * 10\}]^l$$
$$\Rightarrow \mathtt{p}^g = \ldots \wedge 0 \leq \mathtt{n}^g - 1$$
$$\wedge [\{true\}\, \mathcal{F}(s^g); x := 10\, \{x^l = (N - \mathtt{n}^g + 1) * 10\}]^l$$

三つ組の表明以外は明らかであるため消去すると、

$$[\{true\}\, \mathcal{F}(s^g)\, \{x^l = (N - \mathtt{n}^g) * 10\}]^l$$
$$\Rightarrow [\{true\}\, \mathcal{F}(s^g); x := 10\, \{x^l = (N - \mathtt{n}^g + 1) * 10\}]^l$$

これは意味論から明らかである。しかし、$\mathcal{F}(s^g)$ の後ろが複雑化した場合を考えると、次のような規則が可能であることが望ましい。

$$\frac{\sigma \vDash [\{P\}\, S\, \{Q\}]^l \qquad \vdash_l \{Q[\sigma_l]\}\, S'[\sigma_l]\, \{R[\sigma_l]\}}{\sigma \vDash [\{P\}\, S; S'\, \{R\}]^l}$$

ここで、$Q[\sigma_l]$ という記述は、l や内側に含まれるラベル**以外**のラベルが付いた変数について、環境 σ によって値へ変換するものである。これは、表明において意味論上示された三つ組と、推論による三つ組の妥当性を組み合わせて示すものである。この規則の妥当性は健全性から示されるが、現在の健全性では推論のラベルが g に限定されている。そのため、この規則の妥当性を示すには、ラベルを一般化し、表明に他のラベルの変数が出現しないことを言及した上で証明を行う。

この規則により、データとして構築された範囲を推論によって示し、変数を用いて表されている部分を

また、次のプログラムを考える。

```
match p with
| seq_s(p1, p2) =>
   match p2 with
   | assign_s(v, a) =>
      p := seq_s(p1, assign_s("a", a)
   end
end
```

これは、文として $S; x := E$ のような形をしたプログラムを表すデータが格納されたpに対し、パターンマッチで内部を見ることで、$S; a := E$ という形に変更するものである。つまり、初期のpに格納された元のプログラムで実行後にxに入る値を v とすると、変

更後のプログラムでは実行後に aに入る値が v となるよう変更するプログラムである。

この検証時には、先ほどと同様に変数のまま扱われる文（上で言う S）について、aに代入する文を表すデータと連結する部分も問題となるが、もう一点、S に関する三つ組を導出しなければならないという問題がある。プログラム全体を S' として、次の三つ組を考える。

$$\{[\{true\}\,\mathcal{F}(p^g)\,\{x = v\}]^l \wedge p = \texttt{seq\_s}(P, \texttt{assign\_s}(\text{"}x\text{"}, E))\}$$
$$S'$$
$$\{[\{true\}\,\mathcal{F}(p^g)\,\{a = v\}]^l\}$$

ここでパターンマッチ文の推論規則を適用すると、代入文の前後で成り立たなければならない式は以下の通りである。

$$\{[\{true\}\,\mathcal{F}(P); x := \mathcal{A}(E)\,\{x = v\}]^l \wedge p1^g = P \wedge a^g = E\}$$
$$\texttt{p :=seq\_s(p1, assign\_s("a", a)}$$
$$\{[\{true\}\,\mathcal{F}(p^g)\,\{a = v\}]^l\}$$

ただし、事前条件に存在した等式やパターンマッチで生成される等式のうち余分なものは消去している。代入と事前条件の強化によって、

$$[\{true\}\,\mathcal{F}(P); x := \mathcal{A}(E)\,\{x = v\}]^l \wedge p1^g = P \wedge a^g = E$$
$$\Rightarrow [\{true\}\,\mathcal{F}(p1^g); a := \mathcal{A}(E)\,\{a = v\}]^l$$

を示せば良いことが分かる。等式を適用すれば、含意の帰結部を示すには先ほどと同様、$\sigma \models [\{true\}\mathcal{F}(P)\{E = v\}]^l$ と $\vdash_l \{E = v\}\, a := \mathcal{A}(E)\,\{a = v\}$ である。後者は単純な代入に関する推論とラベルの付け替えのみなので、前者に着目すると、含意の前提部に存在するのは複合文の三つ組であり、その一部のみを取り出したものが示すべき命題となる。

これらを示すには、意味論を元に以下の規則のようにかける定理を示す必要がある。

$$\frac{\sigma \models [\{P\}\, S; x := E\,\{Q\}]^l}{\sigma \models [\{P\}\, S\,\{Q[E^l/x^l]\}]^l}$$

定理自体は容易に示せるため省略する。以上の推論とから S' に関する三つ組が証明できる。

このように、推論と意味論の組み合わせを利用することで、既存の推論規則を用いた検証が可能となる。ただし、先ほどの定理に関して、複合文を一般化した以下の規則が成り立つことが望ましいが、一般には成立しない。

$$\frac{\models \{P\}\, S; S'\,\{Q\} \qquad \models \{R\}\, S'\,\{Q\}}{\models \{P\}\, S\,\{R\}}$$

これが成り立たない原因は、S' の事前条件 R をいくらでも強化できてしまうためである。

体系の健全性という視点では、三つ組の表明を導入した影響はほとんどないが、実際に検証を行う際には、対象プログラムの具体化できた箇所の検証を進めるために推論規則を用いたい。本論文では場当たり的に導入すべき規則について述べたが、今後は、各構文に対して推論規則を利用可能にできるよう体系を定式化する。

6 関連研究

Berger らによる研究[1] は、マルチステージ計算に基づくメタプログラミングにおけるプログラム論理の提案であり、操作するプログラムと操作されるプログラムが同一の言語であるという意味で本論文の目的に近いが、以下の点で異なる。

- 内部情報の書き換え。 Berger らの研究では、対象プログラムにおける内部構造を観察、編集することはできない。 一方、本論文ではこれらが単なる構造付きのデータであるため、言語内で自由に操作が可能である。 ただし、そのことからプログラムの正当性を示す段階で適当なプログラムとなること、またはならないことを示す必要がある。
- 対象言語での最適化。 Berger らの研究が対象としている PCF_{DP} では操作されるプログラムの簡約も操作するプログラムと同様に行える。 しかし本論文の手法はそういったことはサポートしておらず、必要であればデータ構造における書き換えをプログラムとして実装する必要がある。

データを変換してコードを得るという観点で、本研究の手法に近いのは eval を持つ言語を対象とした Charlton の研究である [2]。本研究との主な差異は、Charlton の研究では文字列を対象として、主にプログラムの生成に着目している点がある。本研究では、プログラムは既にデータ構造によって表現されているように前提を置いているため、本来は前段階として構文解析が必要である。一方で、Charlton の研究は、文字列に対しての構文解析についても定義し、その正当性を示している。

Charlton の研究では、文字列の構文解析を推論規則側に置いており、データ構造などと混在させて考えることが難しい。 また、構文解析を推論規則側で判定するため、生成されるプログラムの構造をプログラム中で常に文字列として扱わなければならず、構造の一部を書き換えるなどの作業において、他の箇所の構造を破壊しないことを保証することが非常に難しい。逆に本研究に構文解析を導入するには、$\mathcal{F}$ に関して拡張すれば良いため、比較的容易である。この拡張をさらに進め、構文解析部分に対応する操作プログラムを記述、検証することで、操作プログラムの入力から出力まで全体の検証が可能となると考えている。

7 終わりに

本論文では、メタプログラミングのうち、対象のプログラムをデータとして保持し、そのプログラムを操作する場合について、検証する手法を提案した。操作する側のプログラムが対象プログラムをデータとして取り扱うため、コード生成のみだけではなく、編集作業も可能であることを示した。

将来課題として、本研究の用いている推論規則は、編集作業に対してそれほど自由度がないため、規則を強化することが挙げられる。また、本論文では $\mathcal{F}$ に関して基本的な要素として与えておりほぼ議論されていないが、どのような関数が望ましいのかなどの性質について議論したい。特に、6 にも挙げた構文解析について、$\mathcal{F}$ の拡張と $\mathcal{F}$ の関係性を議論することで行われる。また、この関係性を通じて構文解析を行う操作プログラムを記述、検証する。

参考文献

[1] Berger, M. and Tratt, L.: Program Logics for Homogeneous Meta-programming, *Logic for Programming, Artificial Intelligence, and Reasoning - 16th International Conference, Revised Selected Papers*, Clarke, E. M. and Voronkov, A.(eds.), Lecture Notes in Computer Science, Vol. 6355, Springer, 2010, pp. 64–81.

[2] Charlton, N.: Reasoning about string-based runtime code generation, *Unpublished*, (2011).