6L-7

ソースコードへ自然言語による注釈の自動付与を行う手法の提案

杉浦 稜介　下里 祐介　中島 将之　濱川　礼
中京大学 情報理工学部 情報システム工学科

1. 概要

本論文は, C 言語で記述されたソースコードに対し自然言語による注釈(以下コメントと表記)の自動付与を行う手法の提案である. これにより, ユーザのコメント付与への所要時間を短縮させる.

2. 背景・目的

昨今, 技術の発展によりプログラマの負担は増加の一途を辿っており, 複雑化したソースコードに潜むバグが引き起こす社会問題が後を絶たない[1].

大規模なシステムにおいてバグの発生を完全に防ぐには, 発生しにくい開発を心掛けることが最善である. その手法の 1 つとして, ソースコードへのコメント付与がある[2]. 自然言語という性質上, 理解が容易であり, 一般的に適宜記述することで可読性を向上させることができる. しかしコメントは常に正しい内容を記述しなければ意味を成さない. アップデートを行う際にコメントの更新を怠り, 動作内容との不一致が生じれば結果としてバグの温床となる.

これはコメント付与の煩わしさが原因の 1 つと考えられ, 普遍的に存在する問題であるといえる.

そこで我々は, ソースコードへのコメントの自動付与を行い, プログラマの負担を軽減させることを目的とする.

3. システムの概要

コメントを記述する際, 関数や構造体に付加するコメント, 署名等, 自明ではあるが他者への配慮から機械的に付与する場面も多く存在する. これらを定型的なコメントとし, ソースコードの解析を行い, 自動的に挿入する.

処理についてのコメントは各々が特化した記述となる為, 自動化するのは現実的ではないが, サンプルとして 12 名のソースコードを検証した結果, その多くは単行のコメントであり, 推奨位置としてユーザに入力を促し, 各々のコーディングスタイルに沿ったパターン定義を追加可能にすることで対応可能であると判断した.

システムは大きく分けてユーザインターフェース部「Arctium」, 処理部「Origanum」からなり, 処理の流れを図 1 で示す.

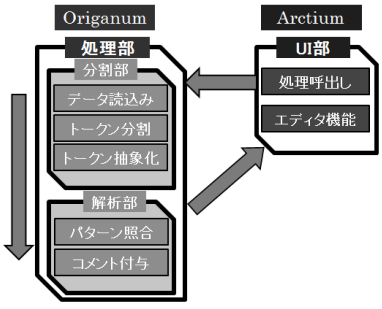

図 1：処理の流れ

Method for performing automatic addition of natural language annotations to the source code
Ryosuke Sugiura,Yusuke Shimosato,Masayuki Nakashima and Rei Hamakawa

3.1. ユーザインターフェース部「Arctium」

Origanum を効果的に使う方法としてコメント付与に特化した専用のユーザインターフェースを用いて操作を行う(図 2).

図 2 はファイルにコメントを付加した状態を示す.

ユーザインターフェースには一般的なエディタ機能の他にユーザのコメント入力の補助として決められた文字列に色を付けるシンタックスハイライトや入力補助としてオートコンプリート, 解析処理前後の差分表示等を実装した.

「ScintillaNET」[3]と「NonDiffNet」[4]を用いている.

図 2:本システムユーザインタフェース

3.2. 処理部「Origanum」

3.2.1. 分割部

定義ファイルの読み込み, ソースコードのトークン分割, 抽象化を行う.

ソースコード, トークン分割時の区切り文字, トークン抽象化処理, 解析処理, コメント付与処理のファイルを読み込み, ソースコードは行単位で分割し, リストに格納する.

トークンの区切り文字を元にリストを細分化し, 別のリストへ格納する. この際, メタデータとして元の行番号を付加しておく. これらをすべての行に対して行う. ソースコードの細分化の後, 解析処理を簡易化する為, トークンを型名であれば「TYPE」, 変数/関数名等プログラマが定義した文字列を「NAMED」のように抽象的な文字列への変換を行う(図 3). 抽象化文字列は 6 種類の抽象トークンと全ての記号が定義されている. なお, 構造体タグ名や列挙型の型名, 定数は文脈情報がなければ区別できない為, 「NAMED」とする.

トークン	抽象トークン
int	TYPE
Func	NAMED
(	(
int	TYPE
n	NAMED
)	)
;	;

抽象化文字列	
TYPE	型名
NAMED	ユーザ定義の文字列
COMMAND	制御構文
VALUE	数値
STRING	文字列
UNKNOWN	それ以外(判別不能なもの)

図 3:抽象化例

3.2.2. 解析部

分解部で生成されたトークン, および抽象化したトークンの並びをもとにパターン解析を行う. C言語の文法上, 関数, 構造体の定義は容易に抽出可能であるが, それ以外の処理内容について汎用性を高める為, 各パターン定義ファイルとの完全一致による比較を基本アルゴリズム(図 4,5)とし, 条件分岐や反復処理はネスト単位で扱い, 入れ子構造となっている場合はネスト毎に再帰的解析を行う他, クリーネ閉包のように部分的な繰り返しとして定義可能なトークン列を検出するアルゴリズムを加えている. コメント推奨位置の検出についても同様である.

パターンの一致が認められたトークン列は, コメント付与対象として, タスクに登録される. タスクにはコメント挿入位置を示す行番号とコメントの種類, および引数や戻り値等コメントブロックを生成する際に必要な情報を付加する.

デフォルトで組み込まれているパターン定義は関数の定義とプロトタイプ宣言, 構造体の定義とプロトタイプ宣言, 署名の 5 種類がある. また, 対象となる処理がトークン列としてパターン化できる場合に限られるが, ユーザ自身によるパターン定義の追加が可能である. 各パターン定義ファイルは, 対象となるトークン列とメタデータで構成されている. メタデータはコメントブロック生成の際に埋め込まれる文字列についての情報であり, 関数名であれば「FUNC_NAME」のように各抽象トークンの持つ文脈上の意味を記述する.

ソースコード全体の解析を終えた後, コメント付与を行う. まず, 上述のタスクを行番号により降順にソートする. 以後先頭に積まれたタスクから順次処理を行う. これにより挿入の際, タスクに登録された行番号の位置がずれることを回避している.

挿入するコメントブロックは, パターン名に対応する定義ファイルから読み込み, 関数名等の情報を文字列として埋め込む. 埋め込む位置はコメントブロック定義ファイル中に解析前に予め記述する.

全てのタスクについて処理を実行し, 結果を Arctium へ出力する.

```
1  FORトークン inトークン列 DO
2      IF Judgement(トークン, パターン定義内同オフセットのトークン) = ELSE    THEN
3          RETURN with FALSE
4      ENDIF
5  ENDFOR
6  RETURN with TRUE
7
8  Judgement(token, pattern)
9  IF token = pattern     THEN
10     RETURN with TRUE
11 ELSE
12     RETURN with FALSE
13 ENDIF
```

図 4:基本アルゴリズム

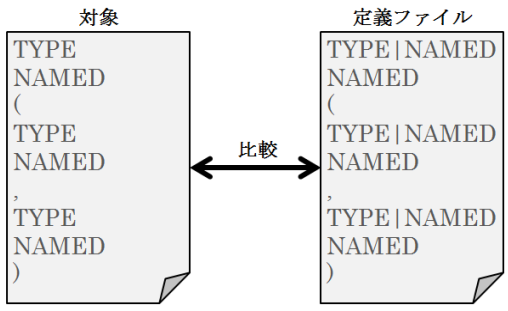

図 5:マッチング例

4. 評価

研究室に所属する学生 24 名に対し, 評価者が各自用意したソースコードを本システムで解析し評価を行なった. 評価の中で「コメントを付ける際, 全て手動で付けることと本システムを使用して付けること, どちらが手間を感じないか」という問いに対して 17 名から本システムを使用するほうが手間を感じないという回答を得た. また付加されたコメント量については「適量」と「やや不足」とでそれぞれ 10 名と評価が分かれた.

図 6 の左が解析前, 右が解析後である.

```
 8  void make4digits(int x[])
 9  {
10      int i, j, val;
11
12      for (i = 0; i < 4; i++) {
13          do {
14              val = rand() % 10;
15              for (j = 0; j < i; j++)
16                  if (val == x[j])
17                      break;
18          } while (j < i);
19          x[i] = val;
20      }
21  }
22
23  int check(const char s[])
24  {
25      int i, j;
26
27      if (strlen(s) != 4)
28          return (1);
```

```
13  /*****************************
14  * make4digits
15  *
16  *----------------------------
17  * 引数: x[](int)
18  * 戻値: void
19  *****************************
20  void make4digits(int x[])
21  {
22      int i, j, val;
23
24  /*****************************
25  *for
26  *
27  *----------------------------
28  *初期化値: i = 0
29  *****************************
30      for (i = 0; i < 4; i++) {
31          do {
32              val = rand() % 10;
33              for (j = 0; j < i; j++)
```

図 6:コメント付与例

5. 関連研究

関連研究として『C/C++ 関数コメントジェネレータ Boss』がある[5]. 戻り値の型, 関数名, 引数, 関数についての説明を入力すると整形されたコメントブロックとしてクリップボードにコピーされ, それを手動で貼り付けるというものである.

6. 考察

付加されるコメント量の評価結果において,「やや不足」という評価が半数を占めていることから, デフォルトのパターン定義だけでは不十分といえる. 別途ユーザによるパターン定義追加機能を用意していたが, 操作の煩雑さから敬遠されたと考えられる. これは組み込みパターン定義の充実とパターン定義作成用に専用 GUI を作成し, 簡略化することで対応する.

またエディタとは別のアプローチとして各種開発環境のプラグイン化することも今後の展望としたい.

なお, Windows7-64bit, Core2Duo, メモリ 4GB の計算機を用いて本システムを 1182 行のソースコードに対して解析を行った結果, 処理時間が平均 1.18 秒で, 192 行のコメントが付与され 1374 行となった.

7. 参考文献

[1] ITpro : 相次ぐシステム障害 http://itpro.nikkeibp.co.jp/trouble/

[2] ロベール著 ロベールの C++入門講座 毎日コミュニケーションズ 2007 年

[3] ScintillaNET http://scintillanet.codeplex.com/

[4] NonSoft http://homepage2.nifty.com/nonnon/index.html

[5] コメントでソースコードを見やすく http://www.forest.impress.co.jp/article/2000/08/02/sundayprog5.html