ソースコードの静的検証に関する研究
– N グラムモデルを用いて –

2005MT010 藤井 健義
指導教員　野呂 昌満

1　はじめに

既存の CDI ツールなどの静的検証ツールには構文解析やデータフロー解析，制御フロー解析などの技術が用いられている．これらの技術により文法的に誤っている箇所や，プログラムのデータや制御の流れから，誤りであると判断する箇所を検出することができる．誤りを開発の早い段階で検出することで開発コストの削減や品質の保証にもつながる．CDI ツールではソースコードをみる際に，あらかじめ決められた規則に基づいて規則とは異なっている箇所を誤りとして検出するので，規則がない箇所を検出できない．本研究の目的は統計的手法を用いて誤りの可能性のある箇所を検出することである．検出された箇所が誤りであるかどうかは人手によって判断をおこなう．目的とする箇所の検出をおこなうために動作しているソフトウェアのソースコードが正しい記述であると仮定し，オープンソースで統計をとる．その統計結果にない記述，頻度の低い記述は検出する規則は定義できないが正しい記述ではないものであると考える．この記述を目的とする箇所と考え，本研究では” 怪しい箇所” とよぶ．自然言語処理の分析に用いられている N グラムモデルを用いることでソースコードを様々な区切りで分割し，その分割した並びの統計をとることができる．本研究では統計的手法として N グラムモデルを用いて Java のソースコードのトークン列と構文規則の適用順序に適用することを考え，怪しい箇所の検出をおこなう．そして提案した手法が妥当であるか考察する．

2　背景技術

ソースコードの静的検証と言語モデル，N グラムモデルの果たす役割について述べる．

2.1　ソースコードの静的検証

ソースコードの静的検証とはソースコードに対し，コンパイラによる文法チェックなどによって，プログラムを実行せずに文法の誤りやコーディング規約に違反した箇所を検出することで誤りの原因を発見することである．

2.2　言語モデルと N グラムモデル

言語モデルはある文字列 $w_1, w_2, \cdots, w_i$ に対し，確率 $P(w_1, w_2, \cdots, w_i)$ を与える役割を果たす．自然言語処理で主に用いられている N グラムモデルでは，ある N 個の文字列 $w_1, w_2, \cdots, w_n$ が並んでいるとき，N 番目の文字の出現が直前の N-1 個に依存すると考え，N 番目の文字の出現確率を $P(w_n|w_1, w_2, \cdots, w_{n-1})$ として与える．N グラムとは任意の単位の N 個の並び $w_1, w_2, \cdots, w_n$ を指す．N グラムモデルは文献の分析で，文献の特徴から書かれた時代を推測することに用いられている [3].

3　Java ソースコードのトークン列への N グラムモデルの適用

本節の手法ではソースコードのトークン列に対し N グラムの統計をとり，怪しい箇所を検出する．N グラムの統計をとる単位を収集単位と呼ぶこととする．辞書とは N グラムモデル用に蓄積したデータを指す．怪しい箇所を検出する処理の手順を**図 1** に示す．

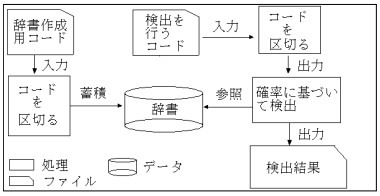

図 1　怪しい箇所検出までの処理の手順

辞書作成と検出に用いるソースコードは収集単位で分割しておく, 検出用のソースコードを分割したものと辞書のデータを比較して怪しい箇所を検出する．

3.1　収集単位の決定と辞書の作成

ソースコードの収集単位として文字，トークン，文，関数，クラス単位があると考える．トークンとは本稿ではプログラミング言語上の最小単位のことをさす．本研究ではトークン単位を収集単位とした．分割する際には空白で分割することにする．辞書の作成に用いるソースコードについて検討する．1 節でのべているようにオープンソースが正しい記述であると仮定し，オープンソースを辞書の作成に用いる．

3.2　実験

怪しい箇所が検出できるかを確かめるために実験をおこなった．オープンソースから 1000 メソッドを用いて辞書の作成をおこなった．検出用のソースコードとして 52 個のメソッドを用いた．結果，全体で 462 箇所が怪しい箇所として検出された．その中で人手により実際に怪しい箇所であると判断した箇所は 75 箇所となった．

3.3　実験結果

実験の結果，三項演算子や条件式中の代入文などの記述が検出された．検出結果の例を**表 1** に示す．

表 1　怪しい箇所と判断したトークンの並びの例

トークンの並び	怪しい箇所と判断した理由
( ) -	( ) の後に-が出現することはあまりない
= 0)	() 内での代入文はあやしい

4 Java の構文規則の適用順序への N グラムモデルの適用

本節の手法では Java のソースコードの構文解析をする際に，構文規則が適用される順番で N グラムモデルを作成する．この手法を用いてドメインや組織毎の怪しい箇所を検出する．

4.1 解析範囲の決定

式に関する部分は if などのキーワードや (などの記号と異なり，自由な記述ができるので怪しい箇所が多く存在すると考える．そこで本研究では Java の構文要素の一つである Expression にいたる範囲で実験をおこなう．

4.2 辞書の作成

N グラムの統計をとる際の収集単位を構文規則とする．ソースコードの構文解析をおこなうときに N グラムモデルの元となる辞書を作成する．統計データは一つのファイルにまとめるのではなく，構文規則ごとに保存する．わけることにより辞書のデータを参照する際のデータ数が減り，参照にかかる処理時間の削減ができると考えた．構文規則の順序の出現回数と過去の適用順序，その出現確率を辞書に保存するデータとしている．過去の適用順序に関しては最後に出現した構文規則ごとにデータをわけて保存しているので直前の N-1 個の順序だけを保存している．

4.3 怪しい箇所の検出

怪しい箇所の検出について検討する．検出用のソースコードを構文解析する際に，構文規則の適用順序と辞書のデータを比較し, あらかじめ決めておいた基準値より出現確率が低い場合に怪しい箇所として検出する．

4.4 実験

検出をおこなうソースコードに対し構文解析をおこなう際に三節と同様の手順で怪しい箇所の検出をおこなう．物理学関係のドメインを扱うソフトウェア二つから構文規則 Expression の単位でソースコードを各 100 個，合計 200 個をモデルとし，JavaPlatformSecondEdition6 のソースコード 200 個を検出用とした．怪しい箇所検出の基準値を 10% とした．

4.5 実験結果

実験の結果，ノイズが多く含まれて検出されたので検出の基準値をさげていった．0.5% まで下げたところで目的の箇所だけが検出されたと判断した．**表 2** に検出されたソースコードの一部を示す．id は識別子を表す．

表 2　検出されたソースコードの例

ソースコード	検出箇所の分析
= new Id(Id,id,true,"")	メソッドの引数に true を記述
_id =new String[id.size()]	インスタンス生成式にメソッド
id.add(new id(null))	コンストラクタに null を記述
id.set(!id.is(id.id))	メソッドの引数に!を記述
id |= id(key);	|=の演算子の記述
this.id.id("GMT") != -1	this の記述
&& (id() instanceof id)	instanceof の記述

5 考察

5.1 提案した手法の妥当性

本研究の手法で検出された箇所が目的である誤りの可能性のある箇所として妥当であるか考察する．実験をおこなった結果，三項演算子や this，論理演算の複合演算子を用いた記述が検出された．これらの記述はあまり用いられない記述であると考えられる．また，インスタンス生成時のコンストラクタの引数として NULL を用いている記述や，メソッドの引数に否定演算子がついている記述なども検出された．NULL や否定演算子自体はよく使用されるがこのような用いられ方はあまりみかけない．これらの記述は特徴的な記述であると考えられる．あまり用いられない記述や特徴的な記述は組織や人によって普段から用いている記述と比べ，記述することに慣れていないので誤りとなる可能性が高くなると考える．以上の考察から提案した手法で検出された箇所の中に妥当な箇所が含まれていると考える．

5.2 関連研究との比較

統計的手法を用いて不具合箇所を検出する研究として Fault-prone フィルタリングがある [2]．Fault-prone フィルタリングでは本研究と同じ様にソースコードから辞書を作成する．辞書のデータを利用して検出をおこなうがトークンの出現が独立であると仮定しており，本研究でトークンが直前のトークンに依存して出現すると仮定している点と異なっている．トークンの出現が依存すると仮定することで，例えば二つの演算子が同時に用いられることがないというような前後と関わりがある箇所も検出することができる．

6 おわりに

本研究では N グラムモデルをソースコードのトークン列と構文規則の適用順序に適用することで，ソースコードから誤りの可能性のある箇所の検出をおこなった．提案した手法の妥当性について考察した．今後の課題として，本研究では構文規則の適用順序へ適用する手法を Expression までの構文要素に限定して解析をおこなったので他の構文要素にも対応することが挙げられる．

参考文献

[1] J. Gosling, B. Joy, G. Steele, and G. Bracha, *The Java^{TM} Language Specification, Third Edition*, Addison-Wesley, 2006.

[2] 森井亮介, 水野修, 菊野亨, “ソースコード中に含まれる不具合トークンをテキスト分類に基づいて推定するツールの試作と評価,” 電子情報通信学会技術研究報告 (ソフトウェアサイエンス) , vol. 108, no. 64, May 2008, pp. 19-24.

[3] 山田崇仁,“N-gram 方式を利用した漢字文献の分析,” 立命館白川静記念東洋文字文化研究紀要, 第一號, Mar. 2007.