確率的言語モデルを用いたJavaソースコードの静的検証
トークンの属性を用いたフォールトの検出

2006MI074 金原　崇人　　2006MI117 成瀬　亮平
指導教員　野呂 昌満　蜂巣 吉成

1　はじめに

静的解析ツールは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．多くのツールで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を用いることでフォールトを検出している．フォールトをソフトウェア開発の早い段階で検出することで開発コストの削減，保守性の向上につながる．既存の静的解析ツールでフォールトのパターンや規則を定義することは大変な労力を必要とし，パターンや規則がない記述には対応ができない．

一方，自然言語処理の分野では確率を用いた手法で音声認識や翻訳などの処理の自動化を行っており，自然言語の曖昧性に対処している．我々はこの手法を用いることで，パターンや規則を定義する手間を省きソースコードの静的検証ツール作成の手間を削減できると考える．また確率という連続な数値を用いてフォールトの度合いを表現し，パターンや規則がない記述を検出できるようにすることで検出の精度が上がると考える．本研究では，Java ソースコードを対象とし，確率を用いた手法を用いてフォールトの検出やパターンや規則がない記述に対処できることを確認する予備実験を行った．

本研究の前提としてフォールトと判定するための確率の基準値を定める必要がある．その基準値を定めるために Java のソースコードを用いて実験を行う．実験の結果から基準値を決めようと考えたが，ただ確率が低いという理由だけでは基準値は決められないことが分かったので，トークンの並びからもフォールトと判断できるようにする必要がある．しかし現状ではこの判断ができないと考え，その解決策を考察した．解決策を考えるために，既存の静的解析ツールで検出できるフォールトの分類を行った．そのカテゴリごとで，どのように工夫をすれば確率を用いて検出可能になるのかを考察した．

2　研究背景

本研究では既存の静的解析にかわる技術として確率的言語モデルを用いた静的検証を提案する．既存の静的解析ツールの問題点と確率的言語モデル，N グラムモデルについて説明する．また確率的言語モデルで既存の静的解析で難しいフォールトの検出が行えると考えた理由を，自然言語で用いられる雑音のある通信路モデルを用いて示す．

2.1　ソースコードの静的解析

検証のための静的解析とはプログラムを実行せず制御やデータの流れを解析し，それにもとづいてプログラムの構造や意味を抽出，あるいは不適切と思われる部分の指摘を行うことである．既存の静的解析技術には抽出したいフォールトごとに規則やパターンを定義する必要があり，それらのパターンを定義する手間がかかるという問題点がある．また構文規則上誤りではない記述をフォールトとして検出する場合には，検出するパターンや規則がそもそも妥当であるかという判断が難しい点が挙げられる．本研究ではこれらの問題点を解消するために確率を用いた手法を提案する．

確率を用いた手法は自然言語処理では雑音のある通信路モデルが概念モデルとして一般に使用されている．雑音のある通信路モデルを図 1 に示す [2].

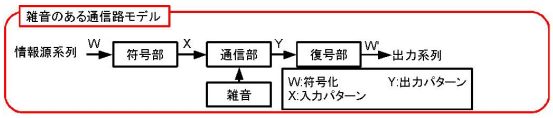

図 1　雑音のある通信路モデル

このモデルに基づいて機械翻訳を実現するのが統計的機械翻訳であり，図 2 にその概要を示す [2].

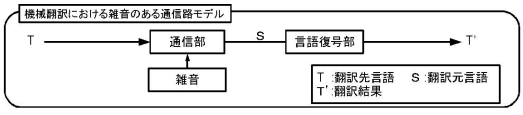

図 2　統計的機械翻訳のモデル

機械翻訳では翻訳元言語 S から翻訳先言語 T への翻訳を考える．このとき翻訳元言語 S は雑音のある通信路を通り，翻訳先言語 T が変換されたもので，翻訳元言語 S から翻訳先言語 T への復号化であると考える．復号化の誤りを最小化すると考えるとき $P(T \mid S)$ を最大化することを考える．このとき $P(S \mid T)$ は通信路の誤り確率である．

一方，ソフトウェアの開発では仕様に基づいてソフトウェアを作成する．ソフトウェア開発モデルを雑音のある通信路モデルに対比させると図 3 のようになる．

モデルは作成されたプロダクトを仕様復元部とし翻訳モデルに基づいている．翻訳モデルに基づいて誤り確率 $P(Y \mid W)$ が低い箇所を検出すればプログラミング言語でのフォールトを検出できると考えることができる．

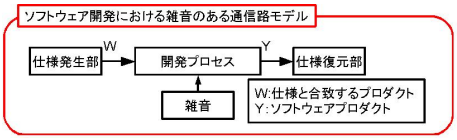

図 3　ソフトウェア開発のモデル

2.2　確率的言語モデル

確率的言語モデルとは，ある単語列 $w_1, w_2, \cdots, w_i$ の出現する確率 $P(w_1, w_2, \cdots, w_i)$ を与える役割を果たす．自然言語の分野で広範に用いられる N グラムモデルは，ある N 番目の単語の出現する確率は直前の $N-1$ 個のみに依存すると考えるモデルである．これらのモデルはコーパスよばれる大量のデータをもとに作成する．

2.2.1　N グラムモデルで算出される確率

本研究では N グラムモデルで算出される単語の出現する確率を出現確率と呼ぶ．

2.2.2　最適な N グラムモデル

N グラムモデルは直前の $N-1$ 個のみに依存するモデルであるので N の値によってモデルが変わるといえる．よってフォールトの検出に適した N グラムモデルを決める必要がある．N グラムモデルの精度はある単語の後に接続し得る単語数の平均（以下平均分岐数とする）によって決まる．また単語の分岐が一通りになると，その単語列がコーパス中に存在するかしないかでフォールトの検出が行われてしまい，確率であらわす必要がなくなると考える．単語の推定が十分に行える精度をもち単語の分岐が一通りになることの少ない N グラムモデルを探す実験を行った．

N グラムモデルの平均分岐数と単語の分岐が一通りになる割合のグラフは図 4 に載せる．図 4 から N の値が 2 以上になると N の値に比例して単語の分岐が一通りになる割合が増えることがわかる．また N の値を増やすと平均分岐数が減っていくことがわかり，N の値が 1 と 2 の間では平均分岐数大きな差があり，2 と 3 の間もそれ以降に比べると差が大きかった．上記の結果から，N の値を増やすほどモデルの精度は高くなり単語の分岐が一通りになることも増えることがわかる．これらの結果から N の値は 2 か 3 がフォールトの検出に適していると考える．また自然言語処理では 3 グラムモデルで単語の推定が十分に行えている結果が出ていることも考慮し本研究では 3 グラムモデルを使用する．

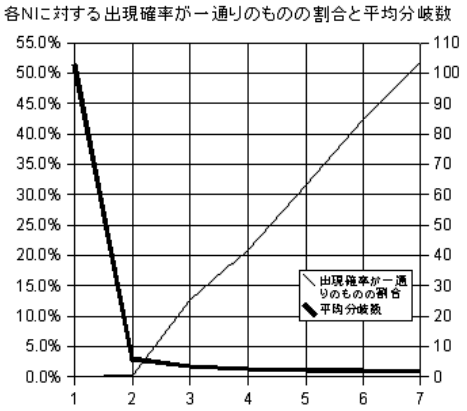

図 4　各Nの単語の分岐が一通りになる割合と平均分岐数

2.3　コーパス

確率的言語モデルの推定には，大規模な言語データベースが必要不可欠である．これらの言語データベースをコーパスと呼ぶ [2]．本研究では Java ソースコードを対象に N グラムモデルで単語の出現確率を参照し，フォールトを検出するので，コーパスとして Java ソースコードを集める必要がある．またコーパスにフォールトが含まれていると，フォールトとなる記述の確率が高くなり適していないと考える．

本研究ではオープンソースのコーパスを用いる．オープンソースは保守性を高めるために大勢の人が確認，修正を行っており，広く利用され，また長期間にわたり改版保守が行われているものほど，その品質は高い．このような理由から本研究ではオープンソースを収集しコーパスとして用いることとした．

2.4　抽象化

本研究における抽象化とは，あるトークン，またはトークン列を特定の構文要素に抽出して 1 トークンに統一することである．これにより必要ない情報を消去できるが，抽象化をし過ぎると必要な情報も消去してしまうという問題点もある．よってフォールト検出に適した抽象化を決める必要がある．これを踏まえ，フォールトと判定する基準値を決めるための抽象化を行った．表 1[*1] にどのように抽象化を行ったかを示す．

3　フォールトの基準となる値

確率を用いてフォールトを検出するには，フォールトであると判断する基準値を定める必要がある．そこで

[*1] 定義しているキーワード以外は抽象化を行わずそのまま表記をしている．

表 1　抽象化の定義

抽象化対象	定義
変数	1 トークンで統一
リテラル	数値，文字，真偽値，NULL
メッセージパッシング	MSG_PASS で統一

我々はコーパスから作成した確率表をもとにフォールト検出に適した基準値を定める．確率表とは 3 つのトークンの並びとその確率を表している．フォールトの候補として最有力となる出現確率が低いトークンの並びとその出現確率の一部を表 2[*2]に示す．

表 2　トークンの並びと出現確率

確率	1 番目	2 番目	3 番目
-4.37694	MSG_PASS	;	–
-4.22451	;	}	DOUBLE
-4.22907	)	{	FLOAT
-4.22412	ID	=	-
-4.21117	OBJECT	ID	{

確率で見れば上に挙げた例はフォールトであるといえる．しかし，3 並びの中身を見た場合，記述される場所，識別子の中身等がわからないことから本当にフォールトとなるかはわからない．どのような並びであればフォールトとして良いかを考える必要がある．そこで既存の静的解析ツールでフォールトとして検出される項目を確率で考えた場合ではどうなるかを考察する．これにより確率でフォールトを検出するには何をするべきかがわかるのではないかと考えた．

4　既存の静的解析ツールとの比較

言語モデルを用いた静的検証ではどのようなフォールトを検出でき，また検出できないかを知る必要がある．そのために既存の静的解析ツールで検出できるフォールトとなる検査項目に着目した．その検査項目の内，確率を用いて検出できるフォールトとできないフォールトに分類した．しかし，確率で検出できる検査項目はほとんどなかった．確率で検出できない検査項目の一部を例として以下に挙げる．

- LongVariable
  (長過ぎる変数名を付けている箇所を検出)
- LocalVariableCouldBeFinal
  (1 回しか代入されないローカル変数が final として宣言されていない記述を検出)

この例以外でも既存の静的解析ツールで検出できるようなフォールト全てを検出することはできないと考えた．検出できない原因にソースコードの抽象化を挙げる．

2.5 節で示した抽象化では，既存の静的解析ツールで検出できるフォールトと同様のフォールトを検出することはできない．識別子等一つのトークンにまとめて抽象化を行っている場合，この抽象化によってフォールトと判断できない記述が増える原因になっている．よって 2.5 節の抽象化の度合いでは抽象化し過ぎているという問題が発生していると考える．抽象化の度合いをフォールトごとに変更すればこの問題は解決できると考えるが，多くの種類がある抽象化の度合いをフォールトを検出する度に変更することは困難であるといえる．そこで我々は抽象化の度合いを変更せずに抽象化の方法を変更することを考え，トークンに属性を用いることを提案する．

5　トークンの属性を用いたフォールトの検出

抽象化の度合いを変更するだけでは，既存の静的解析ツールで検出できるフォールトを確率を用いて検出することはできない．そこで我々は抽象化したトークンに属性付けを行うことで検出できるフォールトを増やすことを提案する．これを行うことでフォールトの検出に適した基準値を決めることができると考える．本研究では，属性とは抽象化されたトークンに抽象化前の情報を付け加えることとする．

そこでトークンが “ どこに ” 記述されているかという属性を与える．その属性で検出できるフォールトとの例として，if-else 文の条件式で否定が使われている記述がある．3 グラムモデルでは 3 トークンの並びで見ているので，if 文か if-else 文のどちらであるかわからない．そこで抽象化時の構文解析でトークンにどこで記述されているかという属性を与える．それにより if 文では否定は使われることはあっても，if-else 文では否定が使われることはあまりないといった確率がわかるようになる．

図 6 にフォールト検出の手順を示す．これにより，フォールト検出に適した基準値を決めることができると考える．

5.1　考察

本研究ではトークンに属性を用いることの提案を行った．“ どこで ” 記述されているかという属性によって検出できるフォールトは if-else 文の条件式内での否定の記述以外にもあると考える．その例を挙げる．これら

[*2] 出現確率欄の確率値は，微小な値の差異を分かりやすく示すために常用対数で示している．

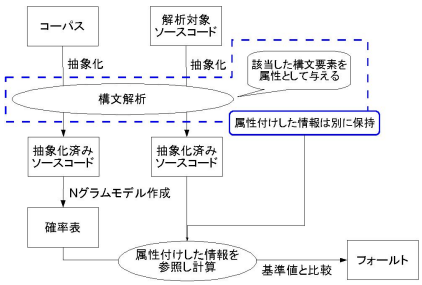

図 5　属性を用いたフォールト検出の手順

は “ どこで ” 記述されているかわかることで検出できるフォールトである．ここで挙げた例は既存の静的解析ツールで検出できるフォールトであり，空のブロックの検出に関しては if 文や for 文においても同様の検出が可能になる．

- finally ブロック内の return の記述
- ブロックが空の while 文
- ループ内でオブジェクトを生成している記述

また，属性付けによって検出できないフォールトもある．本研究では抽象化は Java のソースコードを考慮したものであり，他の記述方法に関しては考慮していない．よって既存の静的解析ツールで検出できる項目の中でも Java 以外の記述方法に踏み込んだフォールトの検出に関しては，どのような属性を用いても検出できないと考える．

4 節で既存の静的解析ツールで検出できるフォールトと同様のフォールトを検出できないと述べている．しかし，トークンの属性の種類を増やすことで検出できるフォールトの種類が増えるのではないかと考える．5.1 節ではトークンに “ どこに記述してあるか ” という属性を与えたが，既存の静的解析ツールで検出できるフォールトすべてをこの属性で検出することは不可能である．

4 節の，確率で検出できないフォールトについてそれぞれで必要な属性を決める．必要な属性を加えていくことでフォールト検出の精度が向上し，既存の静的解析ツールで検出できるフォールトと同様のフォールトも検出可能になると考える．

5.2　既存の静的解析ツールとの相違点

本研究で提案する属性を用いたフォールト検出では，既存の静的解析ツールで検出できるフォールトについて考察をした．しかし，属性を用いても既存の静的解析ツールと同じフォールトが検出できるとは限らないと考える．確率は収集したコーパスに依存し，コーパス中に if-else 文の条件式で否定が多く用いられていれば，このフォールトは検出できなくなる．基準値との比較によってフォールトの検出を行うので，既存の静的解析ツールで定義されていないフォールトに関しても検出が可能になると考える．

また，既存の静的解析ツールではフォールトと判断するための規則やパターンを定義する必要があるが，確率を用いた場合ではその必要がなくなる．それにより静的解析ツール作成の手間を減らすことができるようになる．

6　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 [4]．Fault-prone フィルタリングでは本研究と同様にソースコードから確率表を作成し検出を行うが，トークンの出現は独立であると仮定している．本研究では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ので異なっているといえる．よってトークン列でフォールトの判断をすることができないと考える．

7　おわりに

本研究では言語モデルを用いた静的検証を提案した．基準値を定めることで，その値以下の並びをフォールトの可能性がある記述として提示できるようになる．明確な基準値を定めるために，属性付けを行い，フォールト検出の精度をあげる．検出するフォールトは属性の種類を考察し，必要な属性を加えていくことで，増やすことができると考える．

今後は考察の妥当性を確認するための実験を行い，フォールトの基準値を定義できるようにしていく．

参考文献

[1] James Gosling, Bill Joy, Guy Steele, and Gilad Bracha: Java Language Specification: The 3rd Edition, Peason Education, 2006.
[2] 北研二，確率的言語モデル，東京大学出版会，1999.
[3] 玉井哲雄, ソフトウェア工学の基礎, 岩波書店, 2004.
[4] 菊野亨，水野修，“ フォールト・プローン・フィルタリング：不具合を含むモジュールのスパムフィルタを利用した予測手法, ”, SEC journal，vol.4，no.1，pp.6-15，2008.
[5] 沢田篤史, “ 確率的言語モデルを利用したソフトウェア解析の試み, ” ウインターワークショップ 2009・イン・宮崎　論文集，情報処理学会シンポジウムシリーズ,vol.2009，no.3，pp.5-6，2009.