確率的言語モデルを用いたJavaソースコードの静的検証
構文要素ごとに長さを変更したソースコードの静的検証

2006MI137 恩田　勇気　　2006MI162 杉浦　良祐
指導教員　沢田　篤史

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減らせると考える．また確率という連続的な数値を用いてフォールトの度合いを表現することで，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．

予備実験の結果，フォールトの検出には固定した長さの並びだけでは検出できないという問題点が挙げられた．またフォールトの検出に必要な長さは構文要素ごとに違うことも解った．本研究では，この問題に対処するために，構文要素ごと長さを変化させるソースコードの静的検証を提案する．成果として並びの長さを構文要素ごとに変更してのフォールトとの検出，またパターンや規則のないフォールトに確率の数値をみてフォールトとする等の対処ができた．

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　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，ソフトウェアに内在するフォールトを検出する技術のことである．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．

2.1.1　雑音のある通信路モデル

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].

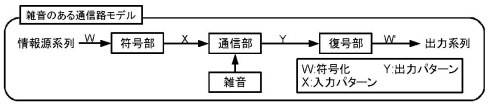

図 1　雑音のある通信路モデル

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].

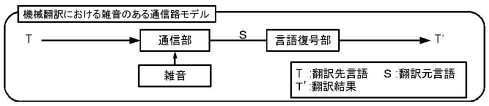

図 2　統計的機械翻訳のモデル

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 [5].

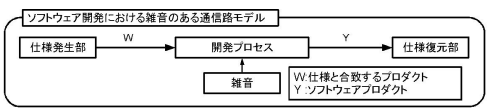

図 3　ソフトウェア開発のモデル

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．

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．また N グラムモデルを用いて算出できる単語列の出現する確率を単語列確率と呼ぶ．単語列確率は出現確率の乗算により算出される確率である．単語列確率の算出方法を図 4 に示す．

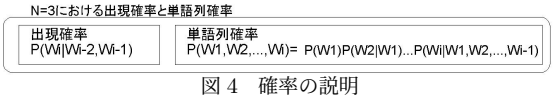

図 4　確率の説明

3 Nグラムモデルのソースコードへの適用

3.1 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なので 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．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 グラムモデルの調査を行った．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 5 に載せる．

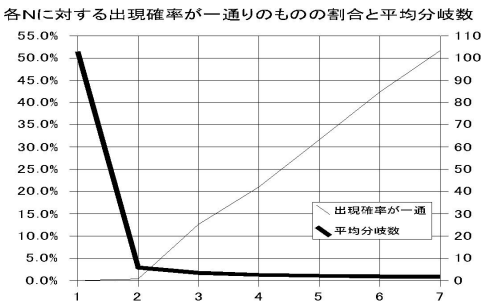

図 5　各Nの単語の分岐が一通りになる割合と平均分岐数

図 5 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．

3.2 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ソースコードの静的検証を行うので，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．

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．

3.3 抽象化

手を加えていない Java ソースコードに N グラムモデルを適用すると，識別子やリテラル等の確率は名前や数字ごとの確率が算出されるが，あまり書かれない識別子の名前等をフォールトとして検出することは，本研究の目的ではない．識別子を ID のように一つにまとめることで，必要な情報を抽出することを本研究では抽象化と呼ぶ．

3.4 基準値

本研究では N グラムモデルで得られた確率と基準値の比較でフォールトの検出を行うので，フォールトの検出に適した基準値の設定を行う必要がある．しかしフォールトとなる記述は決まっていないので，あらかじめ適切な基準値を決めるのは難しい．なのでツール使用者が基準値の変更を行う必要がある．本研究ではフォールトの度合いを表現することで基準値の設定を補助できると考える．

4 確率的言語モデルを用いたソースコードの静的検証

本節では確率的言語モデルを用いた静的検証が有効であることを確認するための実験について説明する．実験の目的は次の二点を確認することである．

- 確率を用いた手法でフォールトを検出できること
- 出現確率を見てフォールトであるか判断できること

4.1 実験の手順と設定

また実験の手順は図 6 に示す通りである．コーパスを抽象化し，3 グラムモデルを作成する．なおコーパスはオープンソースの既存の静的解析ツールから 446528 トークン取得した．各トークンの出現確率を収集したものを確率表と呼ぶ．コーパス以外の Java ソースコードを対象ソースコードとし同じ抽象化を行い，確率表から同じ並びの出現確率を抽出し基準値と比較を行う．基準値以下の記述のフォールトの度合いを算出しフォールトかどうか判断する．本研究では確率の数値と基準値との比較でフォールトの検出を行うと仮定しており，その仮定が成立するならば，確率的言語モデルから計算できる確率と基準値との差をフォールトの度合いとすることができる．フォールトの度合いは次式で示す．

(出現確率 - 基準値) / (出現確率の最小値 - 基準値)

また実験では基準値を出現確率の最大値と最小値の中間の値に設定し検出を行う．

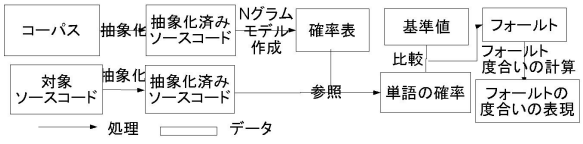

図 6　実験手順

4.2 実験結果

基準値 $10^{-2.18846}$ で検出された結果の一部を表 1 に示す．

表 1　出現確率の実験結果

出現確率	抽出したトークン	フォールトの度合い
-3.89498	IF (-	100 %
-3.41786	IF (--	72 %
-2.53403	RETURN (INT	100 %

出現確率欄の確率表は，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．また抽出されたトークンの 1 トークン目，2 トークン目は出現確率の条件部である．

4.3 考察

3 グラムモデルを用いた実験の結果，フォールトの検出を行うことは出来たが，3 並びを見るだけでは検出できるフォールトには限りがあることが分かった．フォールトの検出をより精度高く行えるようにすることを考えるとトークンの並びの長さを変更すべきである．既存の静的解析ツールの検出項目を調べると，例えば if 文であれば条件式内での代入演算子を検出する項目がある．これを確率を用いた手法で検出するには IF ((ID = のように 5 トークン以上必要である．また return 文であれば必要の無い return 文を検出する項目がある．これを確率を用いた手法で検出するには RETURN ; のようにを 2 トークン必要である．このように各構文要素ごとにフォールトの検出に必要なトークン数が違うと考える．検出の精度を上げることを考えるのであれば構文要素ごとに長さを変更すべきであるといえる．

またフォールトの度合いに関しては IF (- や RETURN (INT は実験結果より 100 % となっている．これらの記述は条件部が IF (や RETURN (の後に来る単語の中では一番確率が低い記述であるので，フォールトの度合いが表せていると考える．しかしフォールトの度合いを計算するには条件部ごとの最小値を取得して計算しなければならない．これは出現確率は条件付き確率であるので同じ条件部を持ったの単語の出現確率の合計が 100 %となるからである．出現確率の条件部の個数は非常に多いので実際に行うことは難しいと考える．

フォールトの検出をより精度高く行うこととフォールトの度合いの問題点を対処するために構文要素ごとに長さを変更しての静的検証を提案する．

5 構文要素ごとに長さを変更したソースコードの静的検証

出現確率を用いた静的検証の問題点から構文要素ごとに長さを変更してフォールトの検出を行う必要があり，単語列確率を用いることで上記の問題点に対処できると考える．単語列確率は 3 グラムモデルを用いて単語列の出現する確率を表すので 4 トークン以上のフォールトにも対応でき，構文要素ごとに長さを動的に変更することも可能である．また単語列確率の最小値は単語列の長さごとに変わるので最小値を取得する回数は少ないと考えフォールトの度合いの問題にも対処できる．本節では単語列確率を用いて構文要素ごとに長さを変更した静的検証が行えるか実験を行い，実験結果から各構文要素に必要なトークン数の考察を行う．実験は出現確率を用いた静的検証と同様の手順で行い，フォールトが多く含まれると考える構文要素 Statement に着目し実験を行った．

実験の目的は次の三点を確認することである．

- 単語列確率を用いて複数の長さのフォールトを検出できること
- 単語列確率を用いた検出でフォールトの度合いが表せること
- 構文要素ごとの検出に必要なトークン数

5.1 実験の手順と設定

実験の手順は 4 節と同様に図 6 に示すとおりである．本節では基準値と単語列確率の比較で実験を行う．コー

パスは 4 節と同様のものを使い，抽象化は同じ操作を行った．またフォールトの度合いは 4 節の計算式を用いるが，最小値は条件部ごとに取得するのでなく，長さごとの単語列確率の最小値を取得する．

5.1.1 実験結果

基準値 $10^{-3.69608}$ で検出された結果の一部を表 2，表 3 に示す．

表 2　トークン数 3 の実験結果

単語列確率	抽出したトークン列	フォールトの度合い
-5.17272	IF (--	75.6 %
-5.64985	IF (-	100 %
-5.64985	RETURN (INT	100 %

表 3　トークン数 4 の実験結果

単語列確率	抽出したトークン列
-5.52647	IF (ID +
-5.45589	IF (ID -
-5.94068	IF (ID =

5.2 考察

5.2.1 フォールトの検出と度合い

実験結果から 3 グラムモデルを用いて検出に 3 トークン必要なフォールトと 4 トークン必要なフォールトを検出することが確認できた．また実験結果から単語列確率でフォールトの度合いを表現できることが確認できた．最小値を長さごとに変更する必要があるが，問題点であった条件部ごとに変更することと比べると改善されていると考える．以上の考察から単語列確率を用いてフォールトの検出が行えたと考える．

5.3 検出に必要なトークン数

実験の結果から検出に必要なトークン数を考察した結果，必要なトークン数が多い構文要素と少ない構文要素に分けることができた．

if 文，while 文，for 文

実験結果から分かるように if 文に関してはトークン数 4 でフォールト検出ができた．また 4 節で必要なトークン数 5 以上の例として挙げた IF ((ID = も検出することができた．これらの結果から if 文は最低でも 5 トークン以上必要であると考える．また while 文や for 文に関しても同様の結果が得られた．またこれらの構文要素は条件式を含み条件式の長さは決まっていないので長くトークン数を取る必要があると考える．

return 文，switch 文，throw 文

これらの構文要素のフォールトは長さ 3 トークンで検出が可能であった．仮に 4 トークン以上とっても以下の例のようになり別の構文要素を含むフォールトが多く検出されるだけであった．

例)RETURN STR_LIT ; DEFAULT

以上の考察から構文要素ごとに必要なトークン数は異なることが実験結果からもわかり，単語列確率を用いて構文要素ごとに長さを変更した静的検証を行うことで出現確率を用いた実験の問題点に対処できたと考える．

6 考察

6.1 確率を用いたソースコードの静的検証の考察

本研究では確率を用いた手法で Java ソースコードのフォールトを検出した．単語列確率を用いて構文要素 Statement に着目し，構文要素ごとに長さを変更する実験を行うことでフォールトの検出や度合いの表現が行えることが確認できた．また検出できたフォールトの特徴として，特定の構文要素で演算子などが使われることが少ないというフォールトが多く検出された．以上の考察から提案した手法でソースコードの静的検証ができると考える．

6.2 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 [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．

7 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に適用し，構文要素ごとに検出の長さを動的に変更する静的検証を行い，提案した手法の妥当性について考察した．今後の課題として我々の研究では構文要素 Statement に着目して静的検証を行ったので他の構文要素にも着目して実験を行う必要があると考える．

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.