

可変エラー距離を用いた構文エラー処理

木 山 真 人^{†1} 竹 森 彬^{†1} 芦 原 評^{†1}

従来のエラー処理では、修正の正しさを判断するため、ある閾値に設定されたエラー距離を用いて、この手法では、事前に適切な閾値を設定しなければならない。また、修正の判断を間違える可能性がある。そこで、本論文では、閾値を固定せず、もっともエラー距離が長い修正を正しい判断とする手法を提案し、実装した。評価の結果、従来手法よりも多くのエラーを修正できた。

Syntax Error Recovery Using Variable Valid Length

MASATO KIYAMA,^{†1} MAKOTO UEDA^{†1} and HYO ASHIHARA^{†1}

Current error recovery use a fixed valid length when you judge whether it is good recovery or not. This method has to set a valid length appropriately and has the possibility of misjudgment. We proposed and implemented new method that use a variable valid length. New proposed method recovered more errors than the former method.

1. はじめに

プログラムにはエラーが含まれている可能性がある。また、含まれているエラーは複数である場合が多い。そのため、1回のコンパイルで、できるだけ多くのエラーを発見できると良い。ここで、エラー処理とは、エラーの発見場所を出力したり、エラーを修正し解析を続行する処理である。

エラー処理は、エラーの修正を行った後、その修正が正しいかどうかの判断を行う必要がある。従来のエラー処理では、修正の正しさを判断するため、ある閾値に設定されたエラー距離を用いている。この手法には (1) 事前に適切な閾値を設定しなければならない、(2) 修正が正しいかどうかの判断を間違えるという2つの問題がある。

本論文では、この問題を解決するため、もっともエラー距離が長い修正を正しい判断とする手法を提案する。また、提案手法の実装、評価を行う。

2. エラーの種類

構文上のエラーは、構文解析処理で発見されるエラーである。構文上のエラーにはローカルなエラーとグローバルなエラーがある。以下にローカルなエラーとグローバルなエラーについて述べる。

● ローカルなエラー

ローカルなエラーはエラーの発見場所と実際のエラーの場所が同じエラーである。ローカルなエラーの例を図1に示す。

a	=	b	+	/	c	;
---	---	---	---	---	---	---

図1 ローカルなエラーの例

図1は演算子 '+' の後に演算子 '/' を続けているというエラーがある。このエラーは '/' を読み込んだ時点で発見される。これは '/' を修正（この場合は削除）すれば、取り除かれる。このようなエラーをローカルなエラーという。

● グローバルなエラー

グローバルなエラーはエラーの発見場所と実際のエラーの場所が異なるエラーである。グローバルなエラーの例を図2に示す。

a	=	d	*	b	+	c	)	;
---	---	---	---	---	---	---	---	---

図2 グローバルなエラーの例

図2には 'b' の前に '(' が抜けているというエラーがある。このエラーは 'b' を読み込んだ時点では発見されず、')' を読み込んだ時点で発見される。これは ')' の修正では取り除かれない。このような

^{†1} 熊本大学大学院自然科学研究科

Department of Computer Science and Electrical Engineering, Kumamoto University

エラーをグローバルなエラーという。

本論文においてはローカルなエラーについて取り扱う。

構文上のエラーが発見されるとそこで解析が停止し、それ以上の解析が行われない。実用的なコンパイラでは、一度により多くの入力文の解析が望まれる。

このため、構文上のエラーを発見した際に、エラーに対し修正を行い解析を続行可能にする“エラー処理”が必要となる。このエラー処理について3章で述べる。

3. エラー処理

エラー処理は、構文解析中にエラーを発見した際に修正を行い、解析を続行可能にする処理である。

本章では、まず3.1節で従来手法であるフレーズレベルエラー回復、3.2節で修正の正しを判断するために用いるエラー距離について述べる。さらに3.3節でエラー距離を用いた従来手法について述べた後、3.4節で従来手法の問題点について述べる。

3.1 フレーズレベルエラー回復

フレーズレベルエラー回復とは、エラーが発見された際、トークンの挿入、削除、置換を行い、エラーからの回復を試みる手法である。

修正の中には、解析の続行を不可能にする修正や、本来入力の中にはなかったエラーを引き起こす修正がある。このような修正を、本論文では“間違った修正”と定義する。また、このような間違った修正により引き起こされたエラーを“後遺症的なエラー”と定義する。これに対し、後遺症的なエラーを引き起こさない修正を“正しい修正”と定義する。

フレーズレベルエラー回復では、修正の後でエラーを発見した際、それが修正により引き起こされた後遺症的なエラーなのか、修正とは関係のない新しいエラーなのかを判断する必要がある。もし後遺症的なエラーであればその修正は間違った修正であったと判断し、新しいエラーであればその修正は正しい修正であったと判断する。

従来手法では、後遺症的なエラーかどうかの判断にエラー距離を用いる。このエラー距離について次節で述べる。

3.2 エラー距離

エラー距離とは、修正を行った後にエラーを発見した場所から次のエラーを発見する場所、または入力の最後の場所までの距離である。ここでいう距離とは解析したトークンの数である。図3にエラー距離の例を示す。

図3では、修正を行ったトークンから次のエラーの

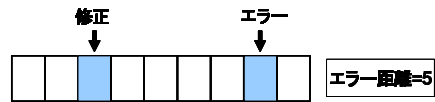


図3 エラー距離の例

トークンまでの間に、5つのトークンがある。そのため、この修正のエラー距離は5である。

次節でエラー距離を用いた従来手法について述べる。

3.3 従来手法

従来手法では、エラー距離と事前に設定した閾値により後遺症的なエラーかどうかの判断を行う。次のエラーを発見した際、エラー距離が設定した閾値を越えていればそのエラーは新しいエラーであるとみなし、正しい修正を行ったと判断する。エラー距離が閾値を越えていなければ、そのエラーは修正により引き起こされた後遺症的なエラーとみなし、間違った修正を行ったと判断する。yacc⁵⁾ではこの閾値が3に設定されている。

図4に後遺症的なエラーかどうかの判断の例を示す。

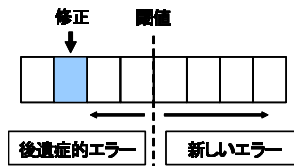


図4 後遺症的なエラーかどうかの判断の例（閾値=3の場合）

図4では、色のついたトークンに対し修正を行う。次にエラーが見つかった際、エラー距離が閾値を越えていなければ、修正により引き起こされた後遺症的なエラーと判断する。エラー距離が閾値を越えていれば、見つかったエラーは修正とは関係のない新しいエラーと判断する。

このように従来手法では、閾値を用いて後遺症的なエラーかどうかの判断を行う。このため、事前に閾値の値を決めておく必要がある。また、閾値が固定となり後遺症的なエラーかどうかの判断を間違えるという問題点がある。これについて次節で述べる。

3.4 従来手法の問題点

エラー距離により適切に後遺症的なエラーかどうかを判断するには、適切な閾値を設定する必要がある。

閾値を小さく設定すると、間違った修正であってもエラー距離が閾値を越え、後遺症的なエラーを新しいエラーとみなす。これにより間違った修正を正しい修正と判断してしまう。

閾値を大きく設定すると、エラーを発見した場所の

近くで発見したエラーを、新しいエラーであっても後遺症的なエラーとみなす。これによって正しい修正を間違った修正と判断してしまう。

図 5 にエラーを含んだ入力例を示す。

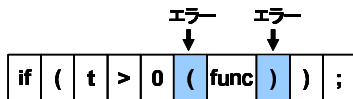


図 5 エラーを含んだ入力例

図 5 でエラーと示した '(' と ')' でエラーとなる。

まず、'(' を ')' に置換する修正を行う。この修正は正しい修正である。この修正では最初の ')' でエラーとなるのでエラー距離は 2 である。閾値を 3 と設定した従来手法では、このエラーは修正によって引き起こされた後遺症的なエラーとみなす。このため、修正 ')' への置換を、間違った修正であると判断する。しかし、実際には ')' への置換は正しい修正であるので、この判断は間違いである。

このように固定した閾値を用いた場合、後遺症的なエラーかどうかの判断を間違い、それにより修正の判断を間違える場合がある。

これらの問題を解決するために、本論文ではもっともエラー距離が長い修正を正しい判断とする手法を提案する。これにより、固定した閾値による判断がなくなり、上記の問題を解決する。これについて 4 章で述べる。

4. 提案手法

本章では提案手法について述べる。まず、4.1 節で提案手法の概要について述べた後、4.2 節で提案手法の例について述べる。

4.1 提案手法の概要

本論文ではエラー距離を “その修正の信頼度” とみなし、もっともエラー距離が長くなる修正がもっとも信頼できる修正であると判断する手法を提案する。

間違った修正は、その修正のすぐそばで後遺症的なエラーを引き起こしやすい。このような修正はエラー距離が短くなる。すなわち、エラー距離が長くなればなるほど、後遺症的なエラーを引き起こしにくいと考えられる。これは “後遺症的なエラーを引き起こさず、入力の最後まで解析を可能にする修正” という正しい修正の定義と一致する。また、 “閾値内にある新しいエラーを見逃す” という問題を回避できると考えられる。

4.2 提案手法の例

提案手法を用いて図 5 の修正を行う場合について述べる。考えられる修正としては ')' の挿入、 '(' の削除、 '(' の ')' への置換などが考えられる。

図 6 に修正の例を示す。

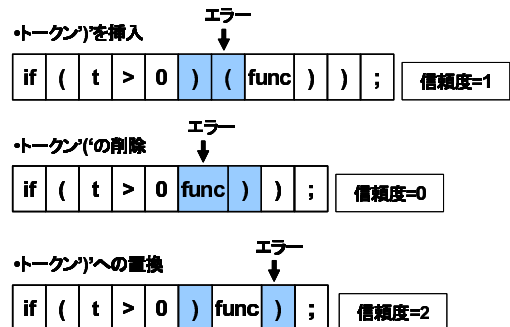


図 6 図 5 に対する修正の例

図 5 に対して挿入、削除、置換の修正を行った場合、それぞれのエラー距離は 1, 0, 2 となる。提案手法ではエラー距離がもっとも長くなる場合、つまりエラー距離が 2 となる修正をこのエラーに対する正しい修正とする。

これにより、従来手法で正しい修正であるにもかかわらず間違いであると判断されていた修正が、提案手法では正しい修正と判断される。

5. 評価

5.1 評価手法

提案手法を実装したパーサにローカルなエラーを含んだプログラムを入力し、正しい修正を行えるかどうかを調べる。

参考文献⁶⁾で使用されているローカルなエラーを含んだプログラム 100 個を、入力プログラムとして使用する。入力プログラムはすべて Java で書かれている。

比較対象として、従来手法である閾値を用いたエラー処理を用いる。この際の閾値は一般的に使用されている 3 と設定する。

5.2 評価結果

本節では評価結果について述べる。まず、5.2.1 項で全体の評価結果について述べる。次に、5.2.2 項で提案手法のみ正しい修正を行えた例について述べる。

5.2.1 全体の評価結果

評価結果を表 1 に示す。

表 1 に示すように、従来手法に比べ提案手法のほうが正しい修正を行えたプログラム数が増加した。

表 1 全体の評価結果	
	正しい修正を行えたプログラム数
従来手法	83
提案手法	90

前述した通り、従来手法は文法に合わせた閾値を事前に設定しなければならない。これに対し提案手法は、文法毎に閾値を設定する必要がある。それにも関わらず、提案手法は従来手法よりも良い結果を示した。

5.2.2 提案手法のみ正しい修正を行えたプログラム

提案手法のみ正しい修正を行えたプログラムの例を、図 7 に示す。

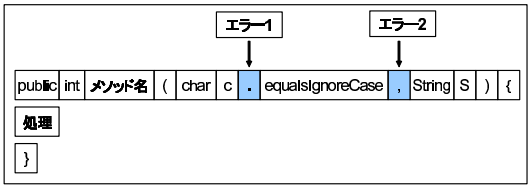


図 7 提案手法のみ正しい修正を行えたプログラムの例

図 7 に示したエラー 1 とエラー 2 にエラーがある。これを正しく修正した例を図 8 に示す。ここでは括弧内のみを示す。

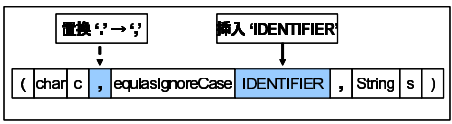


図 8 図 7 のプログラムを正しく修正した例

従来手法ではこのエラーの修正を行えない。従来手法は、エラー 1 に対して「`'` の `'` への置換」の修正を行った後、エラー 2 を発見するため、「`'` の `'` への置換」の修正はエラー距離が 2 となる。これは閾値 3 よりも小さいため、エラー 2 を後遺症的なエラーとみなし、「`'` の `'` への置換」の修正を間違った修正と判断する。

このように、図 7 では、従来手法は正しい修正であっても間違った修正と判断してしまう。これは、エラー 1 のすぐ近くにエラー 2 があるために起こる。このため、従来手法はエラー 1 に対するどのような修正も間違った修正と判断してしまい、修正できない。

これに対し、提案手法では正しい修正を行える。提案手法による修正結果は、図 8 と同様となる。

このように、従来手法で修正できないプログラムについて、提案手法は正しい修正を行える。

6. ま と め

従来の構文エラー処理では、事前に設定した閾値を用いて、正しい修正を行えたかどうかの判断を行う。しかし、固定の閾値では、判断を間違える場合があった。そのため、本論文ではエラー距離がもっとも長くなる修正を正しい修正であると判断するエラー処理を提案した。

評価の結果、正しい修正を行えたプログラムの数は従来手法よりも多くなった。

今後の課題としては、今回対応していなかった 2 つ続きのエラーへの対応が挙げられる。また、効率面についても議論していく必要があると考えられる。

謝辞 本論文の評価結果で用いた java 用パーサを作成した冷水 匠氏に心より感謝します。

参 考 文 献

- 1) 佐竹力, 中井央「オブジェクト指向に基づいた構文解析器生成法の提案」情報処理学会論文誌: プログラミング, Vol.45, No.SIG12(PRO23), pp.25-38(2004)
- 2) 真幡康徳, 中井央「構文解析器生成系と構文エラー処理」情報処理学会プログラミング研究会発表資料 (2005)
- 3) Micheal Burk, Gerald A. Fisher Jr「A PARTICAL METHOD FOR SYNTACTIC ERROR DIAGNOSIS AND RECOVERY」ACM(1982)
- 4) P. レッヒエンブルク, H. メッセンブルク共著, 玉井浩訳「マイクロコンピュータのためのコンパイラ・コンパイラ」サイエンス社 (1991)
- 5) Johonson, 「S.C.:YACC:Yet Another Compiler Compiler」UNIX Programmer's Manual, Vol.2, Holt, Rinebart, and Winston, New York, NY, USA, pp.353-387(1979), AT&T Bell Laboratories Technical Report July 31(1979)
- 6) Carl Cerecke「Repairing systax errors in LR-based parsers」Twenty-Fifth Australian Computer Science Conference(2001)