Vol.2010-SE-167 No.28
2010/3/19

メタ情報とコンテキスト情報を用いた入力補完機能とXPath入力への応用

林　英志[†1]　日高隆博[†2]　山本晋一郎[†1]
小林隆志[†2]　上原正太[†3]　間瀬順一[†3]
鈴村延保[†4]　阿草清滋[†2]

構造化文書に対して検索を行う場合，クエリ言語の内容が構造化文書の構造に依存するため，入力支援機能として補完機能が有用である．我々はXML文書に対するクエリ言語であるXPathに対して，メタ情報としてXPathの構文規則と対象XML文書のDTDの情報を用い，さらに記述中のXPathのコンテキスト情報を組み合わせることを例として，補完機能を系統的に実現できることを示した．

Code Completion Method Using Meta Data and Context Data and its Application for XPath Language

Eiji Hayashi,[†1] Takahiro Hidaka,[†2]
Shinichiro Yamamoto,[†1] Takashi Kobayashi,[†2]
Shota Uehara,[†3] Junichi Mase,[†3]
Nobuyasu Suzumura[†4] and Kiyoshi Agusa[†2]

When retrieving from the structured document, the code completion function is useful as the input support function because query language description depends on the structure of the structured document. We showed that the code completion method was able to be implemented systematically combining those infomation as an example that are syntax rule of XPath and DTD of XML document as the meta information and XPath in process of creation as context information for XPath that is the query language to the XML document.

1. はじめに

我々は先導的ITスペシャリスト育成推進プログラム[1)]のもとOJLプロジェクトを実施しており，カスタマイズ可能なコーディングチェッカであるCX-Checker[2)]の開発を行っている．CX-Checkerはコーディング違反検出対象であるC言語を中間表現であるCX-modelにしたがったXML文書に変換した後，そのXML文書に対してコーディングルールを記述することでコーディングチェックすることができる．

CX-Checkerではコーディングルールを記述する方法の1つとしてXPathを用いた記述方法を持ち，XPathを記述する補助インタフェースであるXPathViewerを備えている．XPathViewerは，ルール記述者のための入力支援環境であり，記述したXPathルールによって得られる結果を対話的に確かめながら記述を進めていくことができる機能を持つ．この機能において補完機能を実現することによってルール記述者の作業効率をさらに向上させることができる．

この例を一般化して考えると構造化された文書に対して対話的にクエリ言語を用いて検索・抽出することは広く行われているため，そのクエリ言語に対する補完機能を実現することは応用範囲が広く有用である．

通常のプログラミング言語では補完を行うために構文規則の情報と入力途中のコンテキスト情報を用いているが，それだけでは有効な補完ができないためにそれらの情報に加え，様々な手法を用いて補完機能を実現している[3)]．一方，クエリ言語ではクエリ言語の構文規則に加えて検索対象文書の情報を利用することができ，これらの情報をうまく組み合わせることで補完機能が実現できると考えた．そこで我々はメタ情報としてクエリ言語の構文規則と検索対象の構文規則を用い，入力中のコンテキスト情報を用いて検索対象例となる文書から抽出した情報を組み合わせることで補完を行うことを提案する．

本報告ではXML文書に対するクエリ言語であるXPathを適用例として，XPathViewer

†1 愛知県立大学
Aichi Prefectual University
†2 名古屋大学
Nagoya University
†3 アイシン・コムクルーズ株式会社
AISIN COMCRUISE Co.,Ltd.
†4 アイシン精機株式会社
AISIN SEIKI Co.,Ltd.

に補完機能を拡張する実装を行い，本手法についての有用性についての評価を行った．

2. CX-Checker

本章では我々のプロジェクトにおいて開発を行っているカスタマイズ可能なコーディングチェッカ：CX-Checker[2] について述べる．

2.1 概　　要

CX-Checker とは以下のような特徴を持つ．

- 容易にルールを追加できる言語を持つ
- ルール追加を補助するインタフェースを持つ
- 複雑なルールにも対応可能なインタフェースを持つ
- 制御フローを使ったルールにも対応可能である．

CX-Checker の全体図を**図 1** に示す．

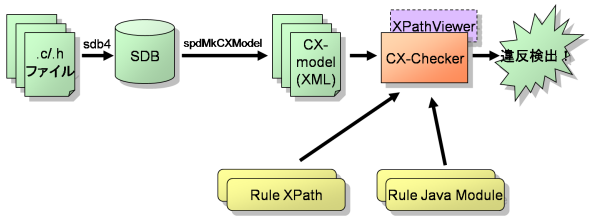

図 1　CX-Checker の全体図
Fig. 1　Overall view of CX-Checker

CX-Checker は検査したいソースコード群とルール群を入力とし，検査を実行することで，警告を出力する．ソースコード内部は CASE ツールプラットフォーム Sapid[4] の CX-model[6] にしたがった XML 文書に変換される．CX-model とはソースコードの構成要素を抽象化したモデルである．CX-Checker ではこの XML 文書に対して違反を検出するルールを記述する (**図 2**)．ルール記述方法は 2.2 節で述べる．

2.2 ルールの記述方法

CX-Checker では CX-model にしたがってタグ付けされた XML 文書に対してルールを記述する．XML 文書を扱う方法として XPath と Java 言語による方法がある．CX-Checker では以下の 3 つの方法を用いてルールを記述することができる．

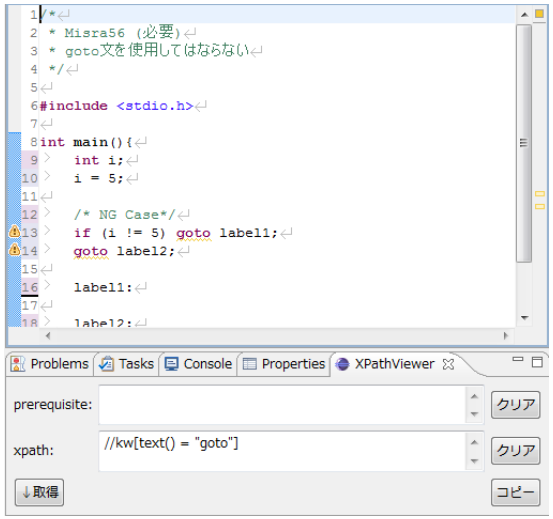

図 2　対話的検索環境 XPathViewer
Fig. 2　XPathViewer : interactive query environment

- DOM(Document Objcet Model) を用いたルール
- ラッパーを用いたルール
- XPath を用いたルール

2.2.1 XPath を用いたルール記述

XPath(XML Path Language)[5] とは XML に準拠した文書の特定の部分を指定する言語構文である．この特定の部分を指定するために XPath では文字列や数値，ブール値を操作するための基本的な機能も提供している．XPath を用い，XML 文書の特定の要素をアドレッシングすることにより違反を検出することができる．

リスト 1 で示した C ソースコードに対して Sapid によりリスト 2 のような XML 文書に変換される．その XML 文書に対してリスト 3 に示すような XPath を記述することによって違反検出を行うことができる．リスト 3 で示した例は，main をテキスト要素として持つ File 要素を検出している．

XPath はロケーションステップを繋ぐことによって構成される．ロケーションステップの基本的な記述形式は軸::ノードテスト [述部] である．軸とはコンテキストノードからみたノードの方向．ノードテストはノードの型と名前を表す．述部とはノードを絞り込むための

リスト 1　C ソースコード

```
main() {
        int a = 0;
}
```

リスト 2　XML 文書 (整形済み)

```
<File id="s8388608">
  <Function id="s33554432">
    <ident defid="s33554432">main</ident>
      <op>(</op>
      <op>)</op>
      <op>{</op>
      <nl line="1" offset="7">
      </nl>
      <sp> </sp>
      <Local id="s33554433">
        <Type>
          <kw sort="type">int</kw>
        </Type>
        <sp> </sp>
        <ident defid="s33554433">a</ident>
        <sp> </sp>
        <op>=</op>
        <sp> </sp>
        <litera defid="s75497472">0</literal>
      </Local>
      <op>;</op>
      <nl line="2" offset="22">
      </nl>
      <op>}</op>
  </Function>
  <nl line="3" offset="24">
  </nl>
</File>
```

リスト 3　xpath の記述例

```
File[//Function/ident[text() = "main" ]]
```

式を書く部分である．軸に関しては省略して記述することができ，リスト 3 で示した XPath は省略形で記述している．

XPath ルール作成方法

XPathViewer における基本的なルール記述の作成を説明する．ここではリスト 1 に示すソースコードにおいて「a」というローカル変数名を検索する場合を例とする．

まず，XML 文書中に識別子を表す ident 要素がどこに存在しているのかを考える．CX-model ではソースプログラムのファイルを表す File 要素から始まる．そして ident 要素は File 要素の子ノードである Function 要素の下の Local 要素の下に存在することがわかる．Function 要素は関数定義を表す要素であり，Local 要素は局所変数宣言を表す要素である．

XPath では，XML の階層構造に対応したロケーションステップを「/」で区切って記述する．この例の場合，ident 要素は File 要素の下の Function 要素の下の Local 要素の直下に存在しているためこれを XPath で記述すると「/File/Function/Local/ident」となる．ここで指定したいものは ident 要素の a というテキスト値である．したがって述部にその条件を記述する．「/File/Function/Local/ident[text() = "a"]」．

XPath の記述には同じ意味でも様々な表現方法が存在するが，どのような表現方法においても例にも示したように単純なルールに対しては非常に少ない記述量でルールを記述することができ有用な方法である．

XPath でルールを記述するためには，CX-model に依存した要素の構造を理解している必要がある．CX-model の構造を理解しない状態で記述すると，検出漏れがあるような不十分な XPath ルールを記述してしまう恐れがある．さらに，XML 文書を繰り返し見ながら記述しなければならず無駄な時間を費やしてしまう．また，CX-model はどのような検出対象プログラムにおいても共通なため，ルール記述の際に毎回似たことを書いている特徴もある．

そこで補完機能を追加することによりある程度 XML 文書を見なくとも記述することが可能となり，さらに入力の手間を省くとともに入力ミスも防ぐことができる．また，補完候補の要素を見ることにより CX-model の構造の理解を促す副次的な効果も期待できる．

3. クエリ言語に対する補完機能

記述途中の言語を入力と対象言語の構文に基づいて解析することにより補完を実現することができる．補完までの全体の流れを 4 に示す．

Vol.2010-SE-167 No.28
2010/3/19

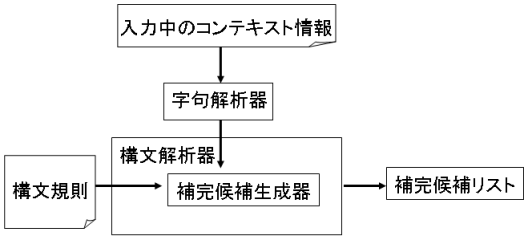

図 3 補完機能の全体図
図 4 Overall view of code completion method

3.1 字句解析

まず記述途中の文字列を意味のあるトークン列に区切る必要がある．1 文字先読みの字句解析を行いトークン列に区切っていく．

あらかじめ対象言語で使われるトークンすべてをリストとして準備しておく．そして最長一致探索によりトークンを区切り，属性などの情報とともにトークンリストに追加していく．通常の字句解析とは違い，補完の場合の字句解析では末尾のトークンが完全ではないトークンの可能性がある．そこで本字句解析器は不完全なトークンもリストに追加し，構文解析器に与えている．

3.2 構文解析

字句解析によってトークン化されたリストと構文規則の情報に基づいて構文解析を行う．構文規則に表される記号の前後を状態として持ち，トークンの情報を基に状態が遷移する．そして最後のトークンまで状態を遷移させた後，次に遷移し得るトークンを補完候補としてリストアップする．

3.3 候補リストの限定

図 4 で示した通りに実装すれば補完機能を実現することができるが，構文規則からは補完対象のトークンの種類を推定することが可能であるが，そのトークンが英数字列で定義されている場合などでは補完される候補を特定することができず，候補数が増大する．

- 検索対象のメタ情報を字句解析器と補完候補生成器に与える
- 検索対象文書例の情報を補完候補生成器のあとのフィルタ処理に加える

この 2 つの情報を加えることで，補完候補リストの数を限定することが可能となり有用な補完機能を実現することができる (図 5)．

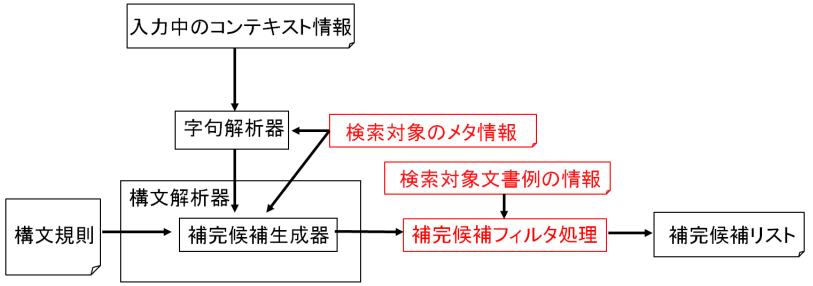

図 5 提案する補完機能の全体図
Fig. 5 Overall view of proposal code completion method

4. XPath への補完機能の適用

4.1 XPath の補完機能

第 3 章で述べた補完機能の手法を XPath へ適用する．クエリ言語は XPath であり，検索対象のメタ情報を CX-model となる．また，検索対象文書例の情報は XML 文書である．そのようにとらえると以下の 3 つの情報を基に補完機能を実現することがいえる．

- XPath の構文規則
- CX-model の DTD
- 対象 XML 文書例

図 6 に補完の様子を示す．右下の一覧が XPath が「`/File/Type[`」のときに補完キー (Ctrl+Space) が押されたときの補完候補である．左下のスペースに記述されていることは選択している補完候補における属性などの説明と候補の使用方法を記述している．要素の補完候補において候補の右側に XML と DTD をそれぞれ表示することによってどちらを情報源としているかを区別した．これは CX-model にしたがう XML 文書から導出された要素であるのか DTD から導出された要素であるのかを区別するためのものである．

作成途中の未完成 XPath からの次の入力の補完は構文規則によって決められる．未完成 XPath からコンテキスト情報を取得し，構文規則をたどることで次に現れる入力候補を取得することができる．しかしながらこれだけでは補完機能は実現できない．構文規則からだけでは補完に適切な XML の要素を取得することはできないからである．この補完に適切な要素を取得するために DTD と対象 XML 文書の情報を用いる．DTD の情報を用いることによりコンテキスト情報以下の要素を取得することができる．また未完成 XPath のコ

Vol.2010-SE-167 No.28
2010/3/19

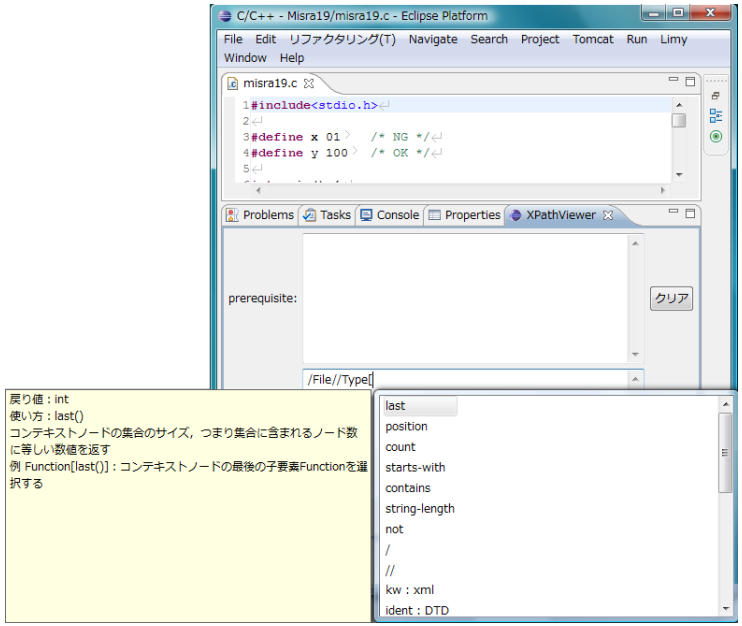

図 6 補完の動作例
Fig. 6 Example of a code completion behavior

ンテキスト要素以下を取り出すために未完成 XPath を変換し，XPath をサポートしているツールである Saxon[9] に読み込ませることで，XML 文書から適切な要素のみを取得する．次項で詳細を述べる．

5. XPath における補完機能の実装

5.1 字句解析部の実装

入力された XPath の字句解析を行う．入力された XPath を先頭から読み込んでいく．そして 1 文字先読みの字句解析によってトークンに区切る (**図 7**)．図に示すトークン集合は XPath で使われるすべてのトークンの集合である．このトークン集合は XPath の構文規則でトークンとして与えられているものと対象 XML 文書の DTD で与えられているものの 2 種類を合わせたものをトークン集合としている．

入力コンテキスト情報の 1 から順を追って字句解析を行う．例のような XPath が入力さ

読み込み位置	入力バッファ	トークン候補	字句解析結果	
			トークン	トークン種別
1	/	/, //		
2	//	//		
3	//k	なし	//	オペレーター
3	k	kw		
4	kw	kw		
5	kw/	なし	kw	要素
5	/	/,//		
6	/a	なし	/	オペレーター
6	a	ancestor ancestor-or-self		
7	an	ancestor ancestor-or-self		
8	anc	ancestor ancestor-or-self		
12	なし	ancestor ancestor-or-self	anc	不明

トークン集合
/
//
ancestor
ancestor-or-self
::
File
Function
kw
⋮

図 7 字句解析例
Fig. 7 Example of a lexical analysis

れたとき先頭から解析を行うため,「/」を解析対象として取り出す.「/」を解析した結果，一致するトークン候補は「/」と「//」である．一致している候補が複数あるために次の文字列を取り出し連結する．次に「//」をトークン候補と比較する．読み込み位置 3 のようにその次の文字列を連結させ，トークン候補がなくなったときにトークンとして区切られる．そしてトークンに区切られた文字列の次の文字からトークン集合と比較していく．これを XPath の文字列の末尾まで行うことで字句解析を行う．補完機能における字句解析は末尾のトークン列が完全ではないことがある．この不完全なトークン列もトークン種別を不明に分類し，構文解析器に与える．

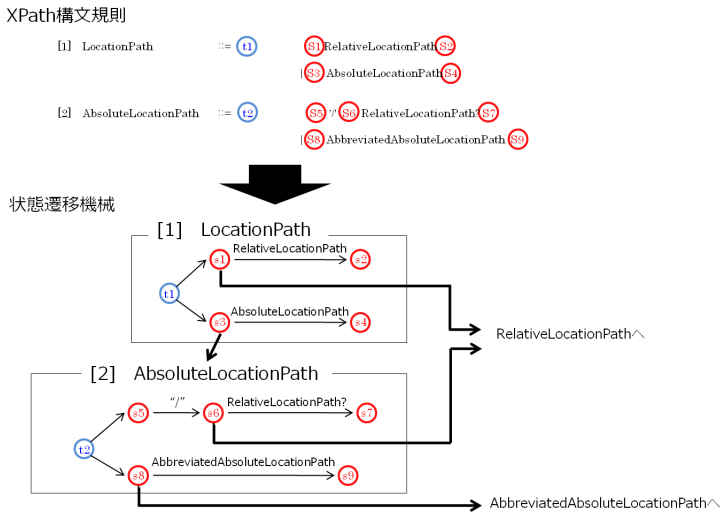

図 8 状態遷移機械の生成
Fig. 8 Generating automaton

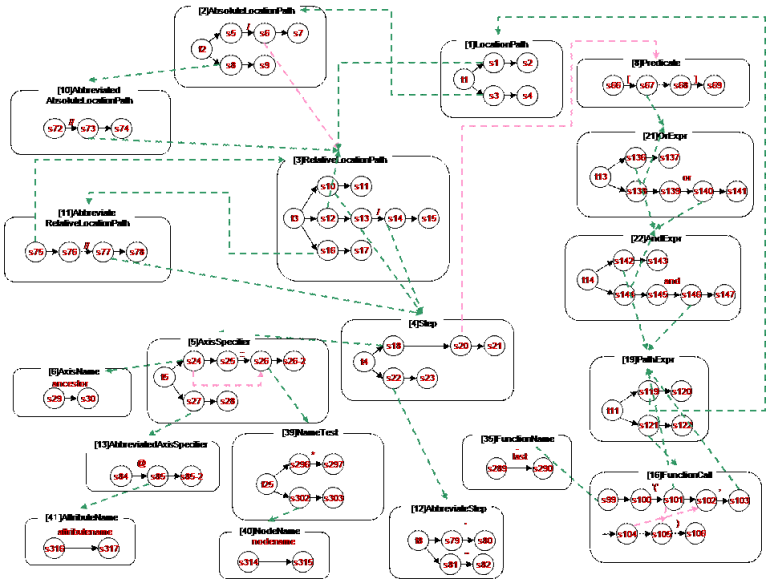

図 9 状態遷移機械
Fig. 9 Automaton

5.2 構文解析部の実装

区切られたトークンをパーサが取得して状態遷移機械にしたがって解析する (**図 9**). 構文解析はプッシュダウンオートマトンを採用している.

図 8 は BNF によって記述された構文規則と状態遷移機械との対応を示している. BNF による構文規則の右辺は, 記号列 (連接) と記号列の選択 (|) によって構成される. EBNF についてもシンタックスシュガーであるため同様に展開可能である.

構文規則における記号列は, 状態遷移機械においては各記号の前後が状態に対応する. 終端記号についてはその状態間を遷移として表し, トークンを遷移の属性として保持する. 非終端記号については, その記号を左辺に持つ構文規則の初期状態への遷移として扱う.

パーサは構文規則と字句解析によって区切られたトークンの情報を入力として遷移し, 字句解析結果の最後のトークンの次に遷移する可能性のある状態を補完候補としてリストアップする. しかし, 最後の字句解析結果の最後のトークン種別が不明であるときは入力途中のトークンである可能性がある. それに対応するために字句解析結果の最後のトークンの 1 つ前の状態における補完候補を記憶しておく. そして入力途中のトークンとトークン集合で一致したトークン候補を補完候補としてリストアップする.

図 7 の入力を例として考えた場合, トークンリストの最後は「anc」であり, これはトークン集合中に存在しないトークンであるため, 要素として認識して次状態のトークン候補を表示することはせず, 要素の途中として認識し, トークン集合中に「anc」で始まる要素を補完候補としてリストする.

図 9 の状態遷移機械は XPath ルールで利用する可能性のある構文規則のみを表したものであるためすべての構文規則に対応していない. また, W3C の構文規則には存在しない [40]NodeName と [41]AttributeName が存在している. これは構文規則上ではこれらを区別する必要がないためであるが, 補完機能においてはノード要素なのか属性要素なのかを厳密に区別する必要があるので図 9 のような状態遷移機械となる. なお, この状態遷移機械で示した構文規則は W3C が勧告している構文規則をより制約を厳しくしたものとなっている.

5.3 DTD に基づいた補完

構文規則からだけでは適当な要素を補完候補としてリストアップすることができないため，DTD の情報を用いて適当な要素の補完を行う．DTD の情報には XML 文書に変換するためのスキーマが書かれている．これを利用することで適当な要素をリストアップすることができる．

例えば XPath 式が「/File/」で補完を行うことを考える．DTD の情報からわかることは現状態の要素の子要素もしくは子孫のすべての要素である．「/File/」の場合，File の直下の要素をリストとして表示すればよいことがわかる．それを正しく表示するためにはまず File が要素であることを認識していなければならない．この情報は字句解析のタイミングでトークン種別を参照することにより知ることができる．次に「/」の後に要素が候補として挙がることを認識していなければならない．これは構文規則から次に要素が候補としてくることを知ることができる．そして字句解析から最後のトークンは「/」であることがわかるため DTD の情報から File の直下の要素を補完候補としてリストアップする．また XPath 式が「/File//」の場合は最後のトークンが「//」であるため，DTD の情報から File 要素の子孫全ての要素を補完候補としてリストアップする．

DTD に基づいた補完が活かされる例としてマクロを使用してはならないというルールを考える (/File[//Define])．

DTD に基づいた補完によりこの XPath ルールを記述することができる．「/File[//」のタイミングで補完キーが押されたときに File の直下に来る全ての要素を DTD の情報を基に取得する．その補完候補の中からマクロを表すタグである Define 要素を選択することにより例のようなルールを容易に記述することができる．

5.4 対象 XML 文書に基づいた補完

XPath でルールを記述するときに利用する要素は，主に CX-model にしたがった XML 文書に存在する要素を使用して記述することが多くあるため，適切な要素を補完候補としてリストアップするためには DTD の情報からだけでは十分ではない．そこで XML 文書に存在する適切な要素を取得する．

取得するためにルール記述途中の XPath 式を変換し，Saxon に読み込ませることで XML 文書に存在する適切な要素のみをリストアップする．

ここで Saxon を利用するためには XPath 式を受理状態の完全な XPath 式に変換する必要がある．また変換する過程で間違った要素を抜き出してしまうような XPath 式にしてはいけない．

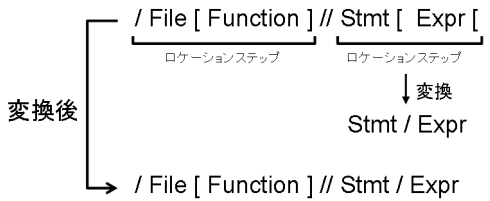

図 10 XPath 式の変換
Fig. 10 Converting XPath

表 1 MISRA-C に対する適用ルール
Table 1 MISRA-C rule of Applicable case

	MISRA-C のルール数
全ルール数	127 ルール
XPath で記述可能なルール数	47 ルール
補完機能適用数	45 ルール

XPath 式の変換方法は**図 10** に示す．まず，ロケーションステップごとに XPath を切り出す．基本的にロケーションステップは「/」や「//」で区切られている．しかしながら述部中に「/」や「//」で区切られたものに関してはロケーションステップとして区切ってはいけない．そこで XPath をロケーションステップごとに区切るためには述部の中ではない「/」と「//」に注目し，区切ることによって簡単にロケーションステップを切り出すことができる．

次に記述途中のロケーションステップのみを抜き出す要素に違いが生じないように変換を行う．変換を行い，ロケーションステップを連結し Saxon が読み込める構文エラーのない XPath 式から要素をリストアップする．

6. 評　　価

6.1 補完機能の適用範囲

CX-Checker ではこれまで MISRA-C のルールを対象として評価を行ってきた．補完機能に対しても MISRA-C のチェックルールに対して補完機能の適用範囲の評価を行った (**表 1**)．

6.2 補完機能の実装における制約

3 通りの方法で補完機能を実現したが，本機能には以下の制約がある．

- 限られた構文規則のみ補完機能が有効であること

表 2 補完ができない MISRA-C ルール
Table 2 MISRA-C Rules of inapplicable case

補完が使えなかった MISRA-C ルール	要因
ルール 11	不等号を使用しているため
ルール 19	不等号を使用しているため

- 述部の階層が 3 階層以内であること

現段階では適用している構文規則が限定されている．MISRA-C を対象として限定されたことによる影響を調査した．調査した結果，補完できない構文規則を用いて記述しているルールは XPath で記述できる 48 ルール中 2 ルールであった．その原因は**表 2** の通りである．

2 つ目の制約はメモリ領域を超えてしまうために起こってしまった制約である．述部の階層が 3 階層以内のルールでないと補完機能が利用できないという制約に対して MISRA-C で調査したところ 4 階層以上必要とするルールは存在しなかった．こちらの制約に対しては現在のところ影響はないと考えられる．

上記 2 点の制約から補完機能が適用できないルールは 2 ルール存在することがわかった．この 2 ルールに対しても補完機能を利用するためには状態と遷移の情報をデータとして登録するのみの修正でよい．しかしながらそのデータを追加することによりさらにメモリを消費するために 2 つ目の制約に影響を及ぼす可能性があるため慎重に検討を進める必要がある．

7. おわりに

7.1 まとめ

本報告では，XPath を適用例としてクエリ言語に対する補完機能を実現する手法を提案した．クエリ言語が構造化文書の構造に依存していることを利用し，クエリ言語の構文規則に加え，検索対象の構文規則と入力中の検索対象例となる文書を組み合わせることで補完機能を実現した．

7.2 今後の課題

現在の補完機能は構文規則が限られている．XPath 式の述部において特定の演算子を利用したときに補完をすることができず，それ以降は補完を行うことができない．XPath を用いたルール記述に演算子を使用することは少なくなく，より複雑なルールを記述するためには全ての構文規則を網羅した補完ができることが望ましい．

述部の階層が 4 階層になったときにメモリ領域を超えてしまうために補完機能が利用できない．この要因は今回の実装においてプッシュダウンオートマトンをそのまま実装したためにメモリを大量に消費してしまったことにある．yacc などで用いられる LALR 法などを適用することでより効率的なアルゴリズムで実現する必要がある．

本研究により構文規則から補完機能を実現できることを示せたので，今後は構文規則をデータとして入力するだけで補完機能が実現できる補完機能フレームワークを実現し，様々な言語に適用できることを目指す．

謝辞 本研究を行うにあたって，文部科学省「先導的 IT スペシャリスト育成推進プログラム」による助成を受けたことを記して感謝する．

参考文献

1) 文部科学省 先導的 IT スペシャリスト育成推進プログラム．
http://www.ocean.is.nagoya-u.ac.jp/
2) 大須賀俊憲, 小林隆志, 間瀬順一, 渥美紀寿, 山本晋一郎, 鈴村延保, 阿草清滋．
CX-Checker: C 言語プログラムのためのカスタマイズ可能なコーディングチェッカ SES 2009 pp.119–126, 近代科学社 2009.
3) Reid Holmes and Gail C.Murphy．Using Structural Context to Recommend Source Code Examples．Department of Computer Science University of British Columbia, Proceedings of International Conference on Software Engineering 2005, pp.117–125, May 2005.
4) 福安直樹，山本晋一郎，阿草清滋．細粒度ソフトウェア・リポジトリに基づいた CASE ツール・プラットフォーム Sapid．情報処理学会論文誌，Vol.39, No.6, pp.1990–1998, 1998.
5) W3C Recommendation. XML Path Language(XPath) Version 1.0
http://www.w3.org/TR/xpath/
6) 渥美紀寿，山本晋一郎，阿草清滋．XML 記述によるソフトウェアリポジトリを用いたコード検索．情報処理学会研究報告, 2005-SE-149, pp.57–64, 2005.
7) Eclipse.
http://www.eclipse.org/
8) MISRA-C 研究会．組込み開発者におくる MISRA-C-組込みプログラミングの高信頼化ガイド．日本規格協会，2004.
9) Saxon.
http://saxon.sourceforge.net/
10) World Wide Web Consortium.
http://www.w3.org