関係代数を基礎とするプログラムに現れる名前解析システム

大堀 淳　上野 雄大

プログラムの理解や保守のためには、プログラムに含まれる種々の名前とその相互関係を解析し理解する必要がある。プログラム開発時に必要とされる名前解析機能は、プログラムポイントで有効な名前集合の取得など、類型的かつ一様であるが、大規模プログラムの理解や保守に必要とされる名前解析機能はそれぞれに固有であり、既存のシステムでは不十分の場合が多い。本研究の目的は、種々な要求に柔軟に対応できるプログラムの名前解析機能の実現方式とその実装方法を開発することである。鍵となる洞察は、種々の固有な名前に関する情報は、コンパイラ利用する名前の属性を表現する関係集合から、関係代数操作を通じて、新たな関係として抽出できるはずである、というものである。我々は、この洞察の下、SML#のデータベース連携機能を用いて、名前に関する種々の属性を関係データベースの関係として保存する機構をコンパイラに組込み、名前解析システムを実現した。本発表では、その概要を報告する。

1　はじめに

大規模なシステムの構築には多大な労力が必要とされるが、そのシステムの保守や拡張にも同様、あるいは、それ以上の労力が必要とされる場合が多い。その際、システムの内部構造を記述したドキュメントは参考になるはずであるが、デバッグや拡張のためにコードを理解するための信頼できるドキュメントの構築と維持も、保守や拡張の問題と同程度に困難な問題である。すべての動作はコードとして書かれており、副作用の使用が限定される ML 系言語の場合そのコードの可読性は高く、その詳細なドキュメントの必要性は高くない、との一般的な議論もあるが、数万あるいは数十万行のコードに対しては、抽象的な議論と言わざるを得ない。これらは、我々が 17 年にも及ぶ長期間に SML#を開発を継続して実施して来た経験に基づく切実な実感でもある。

本研究の一般的な目的は、このような大規模システムのコードの正確な記述という困難な問題の解決に向けた手法の確立である。大規模システムの正確な理解は、ソフトウェア工学の分野で広く認識され多数の研究が行われている重要な研究テーマであるが、未だに体系的な解決法が見出されているとは言いがたい。我々SML#プロジェクトでも、過去にこの問題に取り組み、変数の参照関係を用いたコード・リーディング支援ツールの試作を行った [2]。しかしながら、手法の限界やツールの完成度の問題などから、実際に SML#システムの理解のための活用に供するには至っていなかった。

本研究では、以前試みたアプローチをより系統的な手法へと発展させ、汎用性が高くかつ、種々のニーズに展開可能な手法の確立と、それに基づくコード理解システム構築の基盤の構築を目的とする。この目的を達成する上での我々の洞察は、必要な情報を網羅する基本データ集合を関係データベースの関係集合として表現すれば、それぞれの場面で必要となる多種多様な情報は、関係代数の問い合わせ式を通じて取得できるがずである、というものである。適当に正規化されたテーブルの集合から幅広い情報の取得が可能であることは、関係データベースを基盤とした種々の幅広い情報検索システムの普及に言及するまでもなく、明らかであり、この洞察を基にしたコード理解スステム構築の可能性は大きいと期待される。さらに、データベースシステムとの親和性の高いプログラミング

Atsushi Ohori, Katsuhiro Ueno , 東北大学, Tohoku University.

言語を基盤とすれば、この洞察に基づく汎用性の高いコード理解ススステムの構築を効率よく実現できると期待される。その分析対象として設定した SML#コンパイラは、関係代数 SQL をシームレスに統合しており、記述システムとしても、この要求を満たしている。そこで、本研究では、SML#コンパイラを開発基盤として、コード理解システム構築の基盤技術を開発し、それを基に、SML#コンパイラのソースコードの解析システムを例に、その可能性を検証することを具体的な目標とする。本発表では、システム構築手法の概要、必要とする実装技術、実装した名前解析システムの概要を報告する。

2 名前解析システムの基盤技術

本節では、我々が文献 [2] で提案した既存技術の概要を振り返り、そのアプローチを体系的な名前解析システムの基盤技術して発展させるための技術の概要を提示する。

2.1 コンパイラが解決する名前の参照関係の取得

コード理解の基本は、コードに現れる種々の名前の参照関係の理解である。この洞察の下、我々は、文献 [2] で、名前の参照関係を理解するシステム構築技術とそれに基づくプロトタイプシステムを開発した。その戦略は、コンパイラが決定する名前の参照関係を、そのまま関係として出力し、その関係を Emacs の TAG ファイルや Graphviz ツールの入力ファイルへと編集し、Emacs エディタ上での変数の定義位置や参照位置の表示や、ファイルの依存関係の可視化などを行う、というものである。SML#コンパイラは、種々の複雑な名前の参照関係をすべて解決する名前解決フェーズ `NameEvaluation` を独立のフェーズとして備えている。文献 [2] の研究では、このフェイーズ実行の副作用として、名前の参照関係をファイルに出力する手法を提案している。

2.2 参照関係の基底関係の構築

以上の着想は、単純かつ実効性の高いものであり、本研究でもそれを踏襲することとした。ただ、文献 [2] の研究では、名前の参照関係の用途である Emacs の TAG ファイルの生成等、使用目的に合わせた出力を行っており、取得された参照関係は、これら想定された用途に限定されており、そこから種々の必要に応じた情報の柔軟な取得は困難である。例えば、ファイルの依存関係データの取得などの単純な関連の計算にも、得られたデータの探索プログラムを書く必要があった。そこで本研究では、種々の情報の基底となる基本情報の集合をコンパイラから取得し、それらを関係データベースのテーブルに格納する戦略をとった。分析の結果明らかとなった、幅広いニーズの情報の基底を構成すると思われる関係の集合は、以下の通りである。

- システム情報：システム名、起点となるファルパス、バージョン等の記述
- ソースファイル情報：ソースファイルの絶対パス、ソースファイル ID、ソースファイル種別。
- ファイル対応情報：ソースファイルとインタフェイスファイルの対応関係。
- ファイル依存情報：
  インタフェイスファイルの参照（_require）情報。
- 名前定義情報：名前の定義位置、名前定義の範囲、名前の属性情報。
- 名前参照情報：参照名シンボル、参照出現の先頭位置、参照属性、参照名の定義位置。

これら関係情報を、文献 [2] の手法を拡張し、それぞれ別の関係データベースのテーブルとして格納すれば、種々の場面で必要となる情報を、SQL 式を書くことによって取得できる。SML#は、SQL 式がシームレスに統合されており、この機能により、種々のテーブルを結合し必要な関連を表現する関係代数の SELECT-FROM-WHERE 式を、SML#の式として記述できる。さらに、SML#の再帰関数の中で SQL 式を利用することにより、推移的閉包を含む種々の関連を構築するプログラムを書くことができる。

3 SML#の SQL 統合機能の拡張

本システムの実現のキーとなる技術は、SML#で書かれた SML#言語コンパイラから、上述した種々の参照情報を内部で構築し、それらを関係データベースに登録する機能である。この機能は、SML#の利

点として主張される SQL のシームレスな統合によって、スムーズに実現可能と期待されてたものであったが、本研究で、コンパイラが生成する種々の多量のデータを登録するコードを書く上で不便さに直面した。この不便さは、多相的な静的型付き言語である ML とメタ情報の動的な指定を必要とする SQL 言語との違いに起因するもの言える。

SML#の SQL 統合は、一般に多相型を持つ SQL 式を、SML#の多相レコード機能を用いて、ほぼそのまま SML#の式として使用可能とする技術である。この技術ば、問い合わせを行う SELECT-FROM-WHERE 式において、その威力を発揮することが、本研究でも確認されている。しかしながら、データの生成を行う SQL 式は、問い合わせ式とは異なり、静的な言語では型情報に属するメタ情報を明示的に要求する仕様となっている。この要求から、SML#の SQL 統合では、例えばテーブルの作成はそもそもサポートされておらず、データ生成の典型である `INSTET` 文では、

`insert into #`$db.table$ `(`$label$`,`$\cdots$`,`$label$`)`
`values` exp

のようにカラム名の集合 (`label`,`$\cdots$`,`label`) をプログラム中に列挙することが要求される。この制約のため、種々のデータ生成を行うプログラムの記述が極めて煩雑となり、さらに、汎用的性ある関数の定義が困難となっていた。

この問題は、SML#に最近導入された動的型付け機構 [1] を使うことにより、系統的な解決が可能である。この洞察に基づき本研究では、テーブルの追加とテーブルへのデータのインサートを行う

```
val createTables
  : ['a#reify.conn -> ('a table) -> unit]
val insert
  : ['a#reify. conn -> 'a -> unit]
```

の型を持つ関数を SQL 統合機構に追加した。これによって、例えば、

```
type refTable = {refTable: refTuple list}
val _ = createTable (con, nil:refTable)
...
val L = ... :  {refTable: refTuple list}
val _ = insert conn L
```

のような操作によって、データベースに ML のレコード型を用いてテーブルを生成し、ML で作成したレコード型のリストデータをそのテーブルに追加することが可能となっている。この機能により、種々のテーブルを生成するプログラムも、ML の関数の中で容易に記述可能となった。

4 実装の概要

以上の戦略と拡張された SQL サポートを持つ SML#を基礎に、名前の名前解析のためのデータベースを構築する機能を、SML#コンパイラ自身に実装し、さらに、得られた関係データベースを用いた Emacs ユーテリティを含むプログラムを開発し、テストを行った。本節では、その概要を報告する。

4.1 SML#コンパイラの拡張

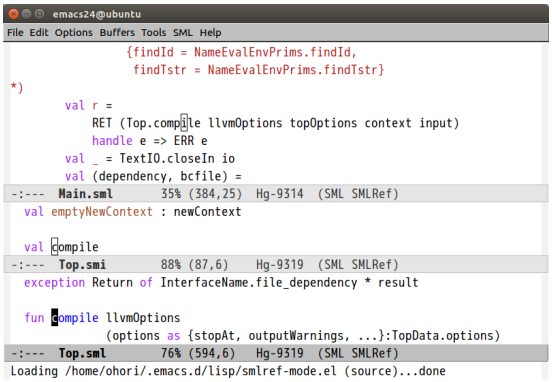
SML#コンパイラは、対話型モード以外に、ソースファイルのコンパイル、分割コンパイルされたオブジェクトファイル集合のリンクを行うリンクモード等を含む複数の起動モードがある。本拡張では、これら起動モードに、名前解析モードを追加することによって実現した。SML#コマンドを`-A` スイッチとともに、

`smlsharp -A`$dbparam$ $file$`.smi`

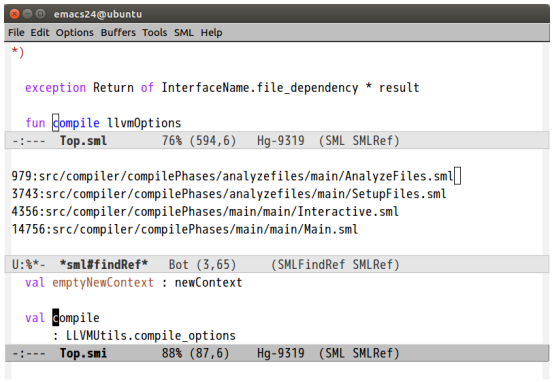
のように起動すると、$file$`.smi` をトップレベルのインターフェイスと見なし、すべてのソースファイル、インタフェイスファイルを分析し、結果を $dbparam$ で指定された関係データベースに格納する。

拡張は、SML#コンパイラに以下の処理を行う `AnalyzeFiles` コンポーネントを追加することのよっておこなった。

1. メインモジュール
   `-A` スイッチで起動され、リンクモードと同様に、インターフェイスを評価し、インターフェイスファイルとソースファイルの一覧を取得し、各ファイルに ID を付与し、データベースに登録した後、それぞれのソースファイルとインタフェイスファイルを、名前分析機能をオンにして、コンパイル関数によって静的解析を行う。
2. データベースモジュール



C-D により定義位置の探索



C-R により参照位置の探索

図 1　smlref-mode.el の利用例

データベースのスキームを型宣言として定義し、あらたに拡張した SQL 統合機能を使用して、データベースに名前分析機能が使用するテールを作成するとともに、テーブルへのレコードの追加機能を提供する。

3. 名前の分析・登録関数
   名前の参照と定義の際呼び出されるデータベースへの名前情報の登録関数を実装し、これら関数を用いて、名前の参照と登録を行うライブラリ関数を改良した。これら名前の参照と登録を行うライブラリ関数は、名前分析機能がオンの場合、これら登録関数を呼び出す。この構造により、SML#のコンパイル関数に殆ど変更を行うことなく、名前の参照と定義情報の登録が可能となる。

以下は、この機能を使用し SML#コンパイラ自身の名前解析を実行してえられる関係データベースの各ユーザテーブルの大きさを `psql` コマンドで表示した結果である。

```
        relname   | reltuples | mbyte
------------------+-----------+--------
 refTable         |    443401 |     51
 defTable         |    106475 |     10
 fileDependTable  |      2593 |      0
 sourceTable      |      1075 |      0
 fileMapTable     |       332 |      0
```

ちなみに、このデータから、SML#コンパイラの概要を以下のよに推測できる。

- 1075 のファイルで構成され、
- ソースファイル、オブジェクトファイル、インタフェイスファイルがそれぞれ 332 個あり、
- 一つのインタフェイスファイルは、平均約 7 (≈ 2593/332) 個のファイルを `_require` し、
- それらファイルの中で約 10 万の名前が定義され、約 44 万の名前の参照がある。

4.2　名前参照システムの試作

構築した関係データベースに対して、関係代数の SELECT-JOIN-PROJECT 問い合わせのより、名前の定義属性（定義位置、属性、定義の範囲）、参照属性（位置、参照モード）などを組み合わせた任意の関係を取得できる。この機能を使用した種々のツール開発は、現在進行中の研究課題である。本研究では、ごく基本的な SQL 式を評価する関数を幾つか定義してテストを行った。

変数参照からその定義を求める関数 `findDef` と、変数定義からその参照リストを求める関数 `findRef` を SQL 式を用いて定義し、さらに、これら関数名と変数位置を標準入力から JSON 形式のデータとして受け取り、関数の結果を標準出力に JSON 形式データとして出力するコマンドとして作成し、そのコマンドを Emacs の sml-mode から呼び出すマイナーモード `smlref-mode.el` を試作した。

このマイナーモード使用の様子を図 1 に示す。

このマイナーモードは、カーソル位置の変数名とその位置（ファイル名とファイル内の位置）を取得し、その変数を変数参照とみなし `findDef` 関数呼び出し要求をする `smlref-find-def` 関数とその変数を変数定義とみなし `findRef` 関数呼び出し要求をする `smlref-find-ref` 関数を定義し、`C-D` と `C-R` キーにバインドしている。

最初のスクリーンショットは、`Top.comple` の位置で `C-D` を押下すると、この関数を定義するインターフェイスファイル `Top.smi` の `compile` 関数が探索され、さらのその位置で再度 `C-D` を押下すると、対応するソースファルの関数定義が探索される様子を示している。２番めのスクリーンショットは、ソースファイル `Top.sml` の `compile` 関数の定義位置で `C-R` を押下すると、インタフェイスファイル `Top.sml` の `compile` 変数の宣言が探索され、さらにそこで再度 `C-R` を押下すると、`compile` 関数を参照している４つのファイル位置のリストが表示される様子を示している。これらファイル位置のリストの各行でリターンキーを押下すると、それぞれの `compile` 関数参照が表示される。このように、本試作により、全ての名前の正確な関連を取得できるとこが、確認された。

その他にも、ファイルを直接的・間接的に参照するファルの参照ツリーを構築する関数や、実際の参照関係系から、正確なインタフェイスファイルの利用関係を求める関数などを定義し、テストを行った。それらは、いずれも期待された動作をし、55 万件に及ぶ名前の定義と参照関係から、種々の意図する関係を導き出せることが確認できた。

5 まとめ

既存のプログラムの理解や保守のために必要とされるプログラムで使用される名前に関する多種多様な関連情報は、すべて、コンパイラが管理する名前に関する情報から構築可能なはずである。そこで、コンパイラが管理する名前に関する情報を、基本的な関係の集合として表現すれば、名前の関連情報の殆どは、それら関係上の関係代数操作を通じて構築可能と期待される。この基本的な洞察に基づき、本研究では、SML#のデータベース連携機能を用いて、名前に関する種々の基本属性を関係データベースの関係の集合として保存する機構をコンパイラに組込み、それら関係集合に対して SQL クエリを発行し、多様な名前の関連情報を構築する基盤を開発した。Emacs 上での簡単な名前検索ツールの試作などを通じ、このアプローチの有効性が確認できた。今後は、この名前解析基盤を活用し、プログラム理解のための種々のツールの開発と提供、さらに、本研究の切っ掛けであったSML#のソースコードを理解するためのドキュメント作成などを行っていく予定である。本研究で開発した名前解析基盤を含むこれらツールは、近い将来、SML#とともにリリース予定である。

謝辞 本研究の一部は JSPS 科研費 18K11233 および 19K11893 の助成を受けたものです.

参 考 文 献

[1] 大堀 淳 and 上野 雄大. Sml#の動的型付け機構. In **日本ソフトウェア科学会第** *36* **回大会論文集**, 2019.

[2] 遠藤 誠典, 百足 勇人, 森畑 明昌, 上野 雄大, and 大堀 淳. 変数参照関係を用いた関数型プログラムのコードリーディング支援. **コンピュータ ソフトウェア**, 32(1):1_194–1_212, 2015.