完全遅延評価による関数プログラムの部分計算

武市正人・金子敬一（計数工学科）

Partial Computation of Functional Programs by Fully Lazy Evaluation

Masato TAKEICHI and Keiichi KANEKO
Department of Mathematical Engineering and Information Physics

We show that a fully lazy evaluator implements partial computation effectively by demonstrating that an interpreter of functional programs can derive a Knuth-Morris-Pratt algorithm from a simple pattern matching algorithm. This research is a very contrast to that of Consel and Danvy in which they have succeeded in derivation of an KMP algorithm by using a conventional partial evaluator. As a consequence of application of several optimization techniques of functional programming, we conclude that a lazy evaluator works as a partial computer by using program transformation based on binding-time analysis.

1. 部分計算と完全遅延評価

二村[2]によれば，部分計算とは稼働環境に関する情報を利用して汎用のプログラムを，より効率の良いプログラムに特殊化することである。部分計算系は一般に2変数関数 Π と定義できる。Π は，第一引数として $n(\geq 1)$ 変数関数 f をとり，第二引数として既知データ k をとり，f の第一引数を k について特殊化した関数を返す。部分計算系 Π に対して理想的な三つの要求がある：

1. 健全性．Π は次の方程式を満足する：
   $\Pi f k x_2 \cdots x_n = f k x_2 \cdots x_n$.
2. 完全性．Π は既知データのみに依存する部分をすべて計算する。
3. 安全性．任意の f, k, および $e_2, \cdots, e_n$ に対して，$f k e_2 \cdots e_n$ が停止するなら $\Pi f k$ も停止する。

しかし，部分計算系はこれらの要求を同時に満足できない。単純な部分計算系は，通常，健全性と完全性の要求を満たすが安全性は満たさない。

一方，完全遅延評価は以下のように定義される：

- 各部分式は，その中のすべての変数が束縛された後は，高々一度しか評価されない。

この特徴から，完全遅延評価系がある意味で部分計算を実行すると結論づけることができる。このような評価過程を完全遅延部分計算と呼ぶこととする。

必須呼び出しを行う評価系を遅延評価系とすると，変数の束縛時解析に基づくプログラム変換により，遅延評価系上で完全遅延評価が実現できる。ラムダ巻き上げ[4]は，プログラムを完全遅延標準形に変換する。この結果，各極大自由式は可能な限り外側に巻き上げられ，その中の全変数が束縛されたとき直ちにその極大自由式も束縛される。よって，変換後のプログラムを遅延評価すれば，もとのプログラムを完全遅延評価することになる。

完全遅延評価系は健全性と安全性を満たし，完全性もある程度まで実現する。しかし，単純な部分計算系が計算を進める簡単な場合でも，失敗することがある。たとえば部分計算系が式 `\x->condTrue (x+1) (x+2)` に対して，`\x->x+1` を生成することを期待するが，完全遅延評価系は，変数 `x` が束縛される度に本体を簡約化してしまう。次節では，この問題に対する救済法を示す。

2. 完全遅延部分計算系

完全遅延評価によって得た残余プログラムの後処理のため変換系を導入する。これは残余プログラムのすべての単純な簡約基を簡約化して，より効率的なものに変換する。適用範囲を制限し，この変換の停止性を確認する。

定義 簡約基 `((\x1->...\xn->e) e1 ... en)` が単純であるとは，以下の3条件を満たすことである：

1. すべての `i(i=1,..,n)` に対して，`xi` は `e` 中に高々一度しか出現しない。
2. `e` 中に局所定義やλ式がない。
3. `e` 中の作用ノードの数が n 未満である。 ■

条件1は線形性を要求し，引数 `ei` の再計算を避ける。条件2は簡約化の結果，プログラムが完全遅延標準形から外れる可能性を除去する。条件3は変換の停止性を保証する。後処理系は，さらに幾つかの最適化をする。

既存の遅延評価系では大域的な計算結果の保持機構 (persistency) を持つものは少なく，部分計算の結果利用は困難である。本実現では，残余プログラムを関数閉包として大域変数に保持し，遅延評価系と後処理系の間の開放系が内部表現をソース言語に変換する（図1参照）。

3. KMP 算法の導出

本節では，部分計算以外の領域で提案されてきた最適化技法について論ずる。これらが我々の目的にどの程度まで有効かを実験結果により示す。最後に完全遅延部分計算によって効率的な残余プログラムを得る技法を示す。

3.1 Consel と Danvy のアプローチ

Consel ら[1]は，かなり単純な算法に従来の部分計算系を適用して KMP 算法[3]を導きうることを，Scheme で書いたプログラム `kmp` と，これをパタン `[1,2,1,2,3]` について特殊化した `kmp_0` を与えることで示した。

以下の実験では，β 簡約化の段数を測定する。

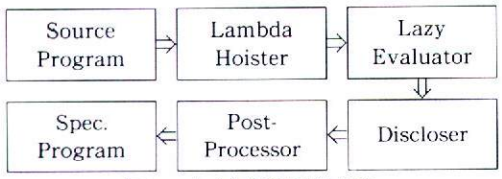

図1　完全遅延部分計算系

実験　関数 **kmpp = kmp [1,2,1,2,3]**を定義し，

(a) 次式を評価して，部分計算を促す：
kmpp [1,2,1,3,1].

(b) これを再評価し部分計算の結果を推定する。

(c) **kmpp** を後処理し，先の式をもう一度評価する。

(d) 最後に次式を評価し，特殊化の効果を推定する：
kmpp [2,1,2,1,3,1].

断わらない限り，各実験は以上のやり方に従う。 ■

Consel らの関数の評価結果を表 1A に示す。この場合，関数 **kmpp** の代わりに **kmp_0** を使用している。

3.2　単純な評価

関数 **kmp** に対し我々の完全遅延部分計算系を使った結果を表 1B に示す。段階 (c) では，後処理した残余プログラムは表 1A と同じ結果を達成しているが，段階 (d) は段階 (a) とほぼ同じ簡約化段数を必要とする。これは不照合時にパタンのシフト量を計算する **static_kmp** の結果を再利用しないからである。この再計算の回避は KMP 算法の本質であり，結果のプログラムはその能力を持たない。以下では，この問題を解決するための技法を示す。

3.3　遅延メモ化と表計算

遅延メモ化の技法では，メモ化関数を評価すると特殊な関数閉包となる。この閉包は引数と結果の対を保持する。評価系は，この閉包の作用に出会うと引数を評価し，閉包が持つ対の中からその引数を捜す。もしあれば結果を再利用し，なければ通常の閉包と同様に評価を進める。

static_kmp の結果の蓄積のため，これをメモ化した結果を表 1C に示す。開放操作は特殊閉包の能力を失わせるので，実験は，(a)→(b)→(c), (a)→(b)→(d) と二回行う。関数 **static_kmp** の再計算の除去により，段階 (d) の簡約化段数は表 1B より 83 段少ない。

遅延メモ化では，特殊閉包の引数を評価するので遅延性を失なう場合がある。そこで **static_kmp** の結果を蓄積するのに，表を作成する手法を試みる。表計算化したものの結果を表 1D に示す。表計算関数の導入により，段階 (d) の簡約化段数は表 1B よりも 47 段少ない。

3.4　不必要な仮引数渡しの除法

完全遅延評価に遅延メモ化や表計算を組み合わせ単純なプログラムから KMP を導出したが，段階 (d) の結果は Consel らのものに比べ不満足である。これは元の

表1　kmpp の簡約化段数

段階	(a)	(b)	(c)	(d)
A	85	81	81	95
B	400	90	81	396
C	397	90	81	313
D	400	90	81	349
E	373	90	81	95

Scheme プログラムが大域関数から成り，全情報を引数で渡し完全遅延評価には見通しの悪い構造であるためである。**kmp** から不必要な仮引数渡しを除去したものの実験結果を表 1E に示す。ラムダ巻き上げが先の表計算関数に相当する部分式を巻き上げるため，段階 (d) の結果は表 1A と等しくなっている。改良プログラムに完全遅延部分計算系を適用しただけで，効率的な KMP 算法を得た。完全遅延性を隠す冗長な引数の除去は簡単な作業である。

4.　結　論

評価の前後でソースレベルの変換を行うことで遅延評価系が部分計算を実現することを示した。この変換はプログラムの静的構造のみに基づき，意味論的情報を必要としない。また KMP 算法の導出の際，幾つかの最適化技法を調べ，不必要な仮引数の除去が重要であると認識した。プログラムの評価系に基づく部分計算系は，残余プログラムを内部表現の形で残すことに問題があるとされてきたが，我々の場合は内部表現を等価なソース言語に変換する開放系があるのであてはまらない。

我々はさらに部分計算の結果を享受するには，persistency を組み込むことが重要であることも学んだ。この機構によって計算を繰り返すたびにプログラムを徐々に改善することも可能となる。

参　考　文　献

1) Consel, C. and O. Danvy: Partial Evaluation of Pattern Matching in Strings, *Inf. Process. Lett.*, **30**(2), 79–86 (1989).

2) Futamura, Y.: Partial Computation of Programs, In *RIMS Symp. on Software Sci. and Eng.*, LNCS 147, Springer-Verlag, 1–35 (1983).

3) Knuth, D. *et al.*: Fast Pattern Matching in Strings, *SIAM J. Comput.*, **6**(2), 323–350 (1977).

4) Takeichi, M.: Lambda-Hoisting: A Transformation Technique for Fully Lazy Evaluation of Functional Programs, *New Generation Computing*, **5**, 377–391 (1988).