함수형 언어의 패턴 매칭 기능과 제약에 관한 연구

권 기 항† · 주 예 찬†† · 신 현 삼†††

요 약

최근에 설계된 함수형 언어들은 어느 정도의 패턴 매칭 기능을 가지고 있으나, 그 기능은 언어마다 조금씩 다르고 일관성이 결여되어 있기 때문에 사용자가 배우고, 이해하기 어렵다. 본 논문에서는 이와 같은 문제를 해결하기 위해서 패턴 매칭의 정확한 정의를 내리고 이를 이용한 여러 예제들을 제시하여 체계적으로 함수형 언어에 패턴 매칭 기능을 도입하는 방법에 대해서 설명한다. 그리고 기존의 함수형 언어들에 있는 패턴 매칭의 제약점을 분석하고, 패턴 매칭을 확장하는 방법에 대해서 설명한다.

Pattern Matching and Its Restrictions in Functional Languages

Kee-Hang Kwon† · Yae-Chan Ju†† · Hyun-Sam Shin†††

ABSTRACT

Modern functional languages provide some forms of pattern matching capability in them. However, these forms are on an ad-hoc basis and vary from languages to languages, making the user hard to understand the feature. To overcome this problem, we present a systematic approach to adding pattern matching to functional languages. We extend to the core functional language with pattern matching capability and illustrate several examples of the language. We also discuss how to extend the pattern matching capability to higher-order terms.

1. 서 론

전통적으로 함수형 언어는 다양한 자료 구조(트리(Tree), 그래프(Graph), 튜플(Tuple))를 쉽게 표현할 수 있는 데이터 생성자(data constructor)를 제공하지 못했다. 예를 들어, Lisp 언어는 리스트 생성자만을 제공하므로 트리를 생성하기 위해서는 리스트를 사용하여 트리를 인코딩(encoding) 해야 했다.

※ 본 연구는 한국과학재단 1997년도 핵심전문연구비 지원(과제번호 : 971-0901-008-2)에 의해 수행되었음.

† 정 회 원 : 동아대학교 컴퓨터공학과 교수
†† 준 회 원 : 블루엣(Bluette) 근무
††† 준 회 원 : 동아대학교 대학원 컴퓨터공학과

논문접수 : 1999년 2월 2일, 심사완료 : 1999년 3월 10일

이와 같은 함수형 언어의 인코딩 방법은 표현 방법이 자연스럽지 못할 뿐만 아니라 타입 검사를 어렵게 하는 등 여러 가지 문제를 내포하고 있다. 예를 들어, 리스트를 사용하여 <Name, Height>란 페어(pair) 구조를 표현 할 때, Name과 Height는 서로 다른 데이터 형을 가지지만 이를 정확히 표현할 수 있는 방법이 없다. 이와 같은 문제를 해결하기 위하여 ML[2,4]과 같은 함수형 언어에서는 사용자가 임의의 데이터 생성자들을 정의할 수 있도록 한다.

그러나 프로그래밍 언어에서 어떤 새로운 생성자를 사용하려면 생성자에 대응하는 소멸자(destructor)를 같

이 제공해야 하지만 소멸자는 사전에 예측할 수 없기 때문에 소멸자를 적절하게 제공할 수 없다. 이와 같은 문제점을 해결하기 위해서 최근의 함수형 언어들에서는 패턴 매칭(pattern matching) 기능을 사용하고 있다. 패턴 매칭은 사용자 마음대로 데이터 생성자를 분할 할 수 있도록 하는 기능을 가지고 있기 때문에 별도의 소멸자를 필요로 하지 않으므로 매우 유용하다. 예를 들어, 1::2::3::4::nil 이라는 리스트가 있을 때 리스트의 head와 tail을 car, cdr 함수를 사용하지 않고 다음과 같은 방법으로 구현할 수 있다 : 1::2::3::4::nil과 X::Y를 같게 만드는 X와 Y의 값을 구할 경우에 X <- 1, Y <- 2::3::4::nil을 대입하면 조건을 만족하는 값을 구할 수 있다. 더 복잡한 예로서 1::1::2::4::nil 리스트와 X::X::Y를 같게 만드는 X, Y의 값은 X <- 1, Y <- 2::4::nil 으로서 쉽게 값을 구할 수 있다.

이와 같이 패턴 매칭은 매우 유용한 기능이지만, 각 언어마다 도입한 패턴 매칭의 기능은 다소 주먹구구식이며, 몇몇은 적합하지 않은 제약 조건들을 가지고 있다. 본 논문에서는 패턴 매칭의 명확한 개념을 정의하고, 이를 함수형 언어에 체계적으로 도입하고, 그 유용성을 여러 예제를 통하여 보이고자 한다.

본 논문에서는 다음과 같은 패턴 매칭 구문을 도입한다.

$$let\ val\ t = Exp_1\ in\ Exp_2\ endlet$$

이 의미는 다음과 같다 : 먼저 식(expression) Exp_1의 값을 구하고 이를 Exp_1' 이라고 한다. 그리고 어떤 항(term) t와 Exp_1'를 같게 만드는 치환(substitution) θ를 찾은 다음, θ를 Exp_2에 대입한 값을 구한다. 예를 들어, *let val X::Y = 1::2::3::4::nil in Exp*으로 표시되는 구문은 *X <- 1, Y <- 2::3::4::nil*의 결과를 얻을 수 있다.

본 논문의 순서는 다음과 같다. 2장에서는 패턴 매칭의 개념과 이를 도입한 함수형 언어를 정의하고자 한다. 3장에서는 여러 예제를 통하여 패턴 매칭의 유용성을 보이고자 한다. 마지막으로 결론에서는 기존 언어에서의 개념을 비교 분석하고 앞으로의 연구 방향을 소개한다.

2. 기본 함수형 언어의 패턴 매칭 도입.

본 장에서는 기본 함수형 언어에 패턴 매칭을 도입하여 그 언어를 확장하고자 한다. 이 확장된 언어는 두 개의 변수 D와 E를 사용하여 다음과 같이 정의된다.

정의 1. 본 논문에서 제시하는 언어는 정의 방정식(defining equation)인 D와 식 E를 사용하여 다음과 같이 정의된다.

D:: = (*val T = E*)*::nil*
| (*val T = E*)*::D*

E::= *V*
| *C*
| *(E₁ E₂)*
| *λx.E*
| *if E then E else E*
| *let D in E*

여기서 V는 변수, C는 상수, T는 단차 항(first-order term)[1]을 나타낸다.

함수형 언어의 전통적인 계산과정은 주어진 프로그램을 이용하여 주어진 식 *E*의 값을 구하는 것(evaluation)으로 정의된다. 여기서 프로그램은 환경(environment)을 의미하고 <*V, T*>의 리스트의 형태를 가진다. *V*는 변수, *T*는 *λ-term*을 의미한다. 그리고 자세한 의미는 다음과 같이 식 *E* 와 환경 *Env*가 주어졌을 때 *eval(E, Env)*를 정의함으로써 명확해진다. 그러나 위 언어에서는 "*let val T = E₁ in E₂ endlet*"라는 패턴 매칭 구문을 가지고 있기 때문에, 이 경우 계산 과정은 값을 구하는 것과 단일화(unification)의 조합으로 이루어진다.

정의 2. 식 *E*와 환경 *Env*가 주어졌을 때 함수 *eval(E, Env)*의 정의는 다음과 같다.

가) *E*가 어떤 변수 *v* 이고, <*v, t*> *Env*일 때
eval(E, Env) = t

나) *E*가 상수(constant) *c* 일 경우
eval(E, Env) = c

다) E가 (E_1, E_2) 일 경우

$eval(E, Env) = eval([R/x]S, Env)$

여기서 $\lambda x.S = eval(E_1, Env)$, $R = eval(E_2, Env)$, $[R/x]E$는 E에서 x 대신 R을 대입한 값을 의미한다.

라) E 가 $\lambda x.E_1$ 일 경우

$eval(E, Env) = \lambda x.eval(E_1, Env)$

마) E가 *if* E_1 *then* E_2 *else* E_3 일 경우

if $eval(E_1, Env) = true$

$eval(E, Env) = eval(E_2, Env)$

if $eval(E_1, Env) = false$

$eval(E, Env) = eval(E_3, Env)$

바) E가 *let* D *in* E 일 때

$eval(E, Env) = eval(E', Env_1 + Env)$

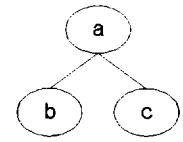
여기서 $Env_1 + Env$는 $Env_1 + Env$의 합집합을 의미한다. 다만 Env_1에 $<x, t_1>$이 있고, Env에 $<x, t_2>$가 있을 때, Env_1의 $<x, t_1>$이 우선 순위를 가진다. D는 $val\ t_1 = E_1$, $val\ t_2 = E_2$, .. $val\ t_n = E_n$ 이라고 하고, $eval(E_i, Env) = P_i$ 이라 할 경우 $Env_1 = unify(\{<t_i, P_i> | 1 <= i <= n\})$를 의미한다. 여기서 unify는 논리 프로그래밍에서 널리 사용되는 단일화 알고리즘을 의미한다[1].

지금까지 패턴 매칭을 도입한 함수형 언어의 구문들과 각 구문의 의미에 대해서 정의하였다. 잘 알려진 바와 같이 단차 단일화(first-order unification)는 결정성이 있고(decidable), occurs-check을 포함할 경우에도 O(nlogn)에 수행할 수 있기 때문에[1], 본 논문에서 확장된 언어는 속도 측면에서 기본 언어에 비해 크게 뒤지지 않을 것으로 생각된다.

다음 장에서는 우리가 정의한 언어를 이용하여 여러 가지 예제들을 살펴본다.

3. 패턴 매칭의 응용

이 장에서는 2장에서 정의한 언어를 이용하여 패턴 매칭의 여러 예들을 살펴 보도록 한다. 그 첫 번째 예로서 이진 트리(binary tree)의 높이(height)를 구하는 프로그램을 생각해 본다. 먼저 트리 형의 데이터를 생성하기 위해서 트리 생성자로는 bintree를 사용하고 empty 트리는 e로 나타낸다.



(그림 1) 첫번째 예를 위한 이진 트리

예를 들어, (그림 1)과 같은 이진 트리는 bintree(a, bintree(b,e,e), bintree(c,e,e))로 간단하게 나타낼 수 있다. 그리고 이진 트리에 대한 높이(height)를 구하는 프로그램은 다음과 같이 패턴 매칭을 사용하여 정의할 수 있다.

```
height(T) =
   if T = empty then 0
   else
      let val bintree(Root, Lsubtree, Rsubtree) = T
      in 1 + max(height(Lsubtree)), height(Rsubtree))
      endlet
```

두 번째 예로서 패턴 매칭을 사용한 퀵 정렬(quick sort)을 설명한다. 먼저 그 정의는 다음과 같다.

```
qsort(L) =
   if L = nil then nil
   else
      let val X::L1 = L
      in qsort(fst(split(X, L1)) @ (X::nil) @ (qsort
      (snd(split(X, L1)))
      endlet
```

여기서 @는 append 함수를 나타내고 infix 연산자로 사용된다. 여기서 주의할 점은 하부 식인 split(X, L_1)는 값을 중복해서 구해야 한다는 점이다. 이와 같은 문제점을 피하기 위해서 우리는 다음과 같이 패턴 매칭 구문을 사용한다.

```
qsort(L) =
   if L = nil then nil
   else
      let val X::L1 = L
```

```
    in let val (Low, High) = split(X, L1)
      in qsort(Low) @ (X::nil) @ qsort(High)
      endlet
    endlet
```

여기서 Low, High들 각각 X 보다 작은 원소들의 리스트, X 보다 크거나 같은 원소들의 리스트를 갖게 된다.

세 번째 예로서 피보나치(fibonacci) 수열을 구하는 함수는 일반적인 정의와는 달리 새로운 방식으로 정의할 수 있다.

```
if n = 1 then 1
else if n = 2 then 1
else fib(n) =
   let val X = fib(n-1) and val Y = fib(n-2)
   in X + Y
   endlet
```

위 정의의 경우 fib(n)의 정의가 논리 프로그래밍(logic programming) 스타일로 작성되었음을 알 수 있다. 이런 관점에서 본 연구에서 제시한 언어는 함수형 언어(functional language)와 논리 언어(logic language)의 결합의 한 분야라고 할 수 있다.

4. 결 론

본 논문에서는 기본 함수형 언어에 패턴 매칭 기능을 도입하고 이를 확장하였다. 이를 위해 다음과 같은 구문을 정의하였다.

let val t_1 = E_1 *and ... and val* t_n = E_n *in Exp endlet.*

이 구문은 여러 가지 방법으로 사용할 수 있으며, 3장에서 여러 예제를 통해서 함수들의 지역 정의와 패턴 매칭의 기능을 동시에 얻을 수 있음을 보였다. 사실, 이 구문은 ML, Hope의 let/where 구문과 유사하지만 다음과 같은 점에서 차이가 있다.

가) ML의 경우 패턴 매칭 구문에서 동일한 변수가 두 번 이상 나올 수 없다.(page89 in [5]) 예로서 *let val X::X::Y = 1::1::2::nil*과 같은 구문은 사용할 수 없다.

나) ML의 경우 *let val X = λx.λy.(+ x y)*과 같은 구문은 사용할 수 있다. 그러나 *val* t_1 = t_2에서 t_2가 λ-*term* 이면 t_1은 하나의 변수만 사용할 수 있기 때문에 *let + X Y = λx.λy.(+ x y)*과 같은 구문은 사용할 수 없다.

나)의 경우에서 볼 때 ML의 패턴 매칭은 λ-*term*을 사용하는데 있어서 여러 가지 제약 조건이 있음을 알 수 있다.

앞으로 본 논문에서 정의한 패턴 매칭 기능을 더욱 확장하여 고차(higher-order) 패턴 매칭[3]을 도입하고자 한다. 이 경우 우리는 다음과 같은 구문을 사용할 것이다.

let val t = Exp1 in Exp2 endlet

이 때 t는 임의의 λterm 이 허용된다. 고차 패턴 매칭의 예로서 다음을 살펴보자.

```
let val (forall P or Q) = forall ( λx.(g x) or λx (r x))
in (P a)
endlet
```

여기서 P는 $\lambda x.(g\ x)$, Q는 $\lambda x.(r\ x)$의 값을 가지게 되고, $(P\ a)$는 $(g\ a)$의 값을 가지게 된다.

고차 패턴 매칭은 메타 언어로서의 함수형 언어 기능을 향상시키고, theorem prover[6] 작성시 매우 유용하게 사용된다. 하지만 고차 패턴 매칭의 경우 항상 결정적이지 않고, 가장 일반적인 unifier가 존재하지 않는다[3]. 따라서 이런 문제점들을 신중히 고려해야 하므로 매우 흥미 있는 문제다.

참 고 문 헌

[1] J. Corbin and M.Bidoit, *Rehabilitation of Robinsons unification algorithm*, In R.Pavon, editor, Proc 1983 IFIP Congress, pages 909-914, North-Holland, 1983.

[2] R.Harper, R.Milner and M.Tofte, *The definition of Standard ML(version 2)*, MIT Press, 1989.

[3] G. Huet, A unification algorithm for typed-calculus, *Theoretical Computer Science, 1:27-57, 1975.*

[4] R.Milner and M.Tofte, *Commentary on Standard ML*. MIT Press, 1991.

[5] Chris Reade, *Elements of functional programming*, Addison Wesley, 1989.

[6] M. Vaninwegen and E.Gunter, Hol-ML. Higer-order Logic Theorem Proving and its Applications, *Springer LNCS 780, 1994.*

권 기 항

e-mail : khkwon@daunet.donga.ac.kr

1983년 서울대 컴퓨터공학과 졸업(학사)

1985년 미국 Georgia Tech 컴퓨터과학과 졸업(석사)

1986년~1988년 (주)상운

1995년 미국 Duke 대학 컴퓨터과학과 졸업(박사)

1995년~현재 동아대학교 컴퓨터공학과 조교수

관심분야 : 소프트웨어공학, 프로그래밍 언어

주 예 찬

e-mail : elephant@ce.donga.ac.kr

1997년 동아대학교 컴퓨터공학과 졸업(학사)

1999년 동아대학교 대학원 컴퓨터공학과(공학 석사)

1999년~현재 블루엣(Bluette)에 재직중

관심분야 : 소프트웨어공학, 검색엔진, 객체지향언어 등

신 현 삼

e-mail : sami32@ce.donga.ac.kr

1998년 경성대학교 정치외교학과 졸업(학사)

1999년~현재 동아대학교 대학원 컴퓨터공학과(석사과정 재학중)

관심분야 : 소프트웨어공학, 프로그래밍 언어 등