Próximo:Desestruturando com padrões pcase, Anterior:Estendendo pcase, Acima:

Condicional de correspondência de padrões [Conteúdo][Índice]

11.4.3 Padrões de Estilo Backquote

Esta subseção descreve *padrões de estilo backquote* , um conjunto de padrões internos que facilitam a correspondência estrutural. Para obter o plano de fundo, consulte Condicional de correspondência de padrões .

Padrões de estilo backquote são um poderoso conjunto de `pcase`extensões de padrão (criadas usando `pcase-defmacro`) que facilitam a correspondência de *expval* com especificações de sua *estrutura* .

Por exemplo, para corresponder a *expval* que deve ser uma lista de dois elementos cujo primeiro elemento é uma string específica e o segundo elemento é qualquer valor, você pode escrever um padrão principal:

```
(e (pred listp)
     ls
     (guarda (= 2 (comprimento ls)))
     (guard (string = "primeiro" (car ls)))
     (deixe o segundo elemento (cadr ls)))
```

ou você pode escrever o padrão de estilo backquote equivalente:

```
`("primeiro", segundo-elem)
```

O padrão de estilo backquote é mais conciso, lembra a estrutura de *expval* e evita vinculação `ls`.

Um padrão de estilo backquote tem a forma em que *qpat* pode ter as seguintes formas: `` `*qpat* ``

`(`*`qpat1 . qpat2`*`)`

Corresponde se *expval* for uma célula cons cujas `car` correspondências *qpat1* e cujas `cdr`correspondências *qpat2* . Isso se generaliza prontamente para listas como em . `(`*`qpat1 qpat2 …`*`)`

`[`*`qpat1 qpat2 … qpatm`*`]`

Corresponde se *expval* for um vetor de comprimento *m* cujos `0`.. th elementos correspondem a *qpat1* , *qpat2* … *qpatm* , respectivamente. `(`*`m`*`-1)`

`symbol`

`keyword`

`number`

`string`

Corresponde se o elemento correspondente de *expval* for `equal`para o objeto literal especificado.

`,pattern`

Corresponde se o elemento correspondente de *expval* corresponder a *pattern* . Observe que o *padrão* é qualquer tipo que `pcase`suporte. (No exemplo acima, `second-elem`é um padrão de núcleo de *símbolo* ; portanto, corresponde a qualquer coisa e let-binds `second-elem`.)

O *elemento correspondente* é a porção de *expval* que está na mesma posição estrutural que a posição estrutural de *qpat* no padrão de estilo backquote. (No exemplo acima, o elemento correspondente de `second-elem`é o segundo elemento de *expval* .)

Aqui está um exemplo de uso `pcase`para implementar um interpretador simples para uma pequena linguagem de expressão (observe que isso requer vinculação léxica para a expressão lambda na `fn`cláusula para capturar `body`e `arg`(consulte Lexical Binding):

```
(defun avalia (form env)
  (formulário de caixa
    (`(adicionar ,x ,y) (+ (avaliar x env)
                           (avaliar y env)))
    (`(call ,fun ,arg) (funcall (avalia fun env)
                                (avaliar arg env)))
    (`(fn ,arg ,body) (lambda (val)
                        (avaliar corpo (cons (cons arg val)
                                             env))))
    ((pred numberp) formulário)
    ((pred symbolp) (cdr (assq form env)))
    (_ (erro "erro de sintaxe: formulário %S"))))
```

As três primeiras cláusulas usam padrões de estilo backquote. `` `(add ,x ,y) ``é um padrão que verifica se `form` é uma lista de três elementos começando com o símbolo literal `add`, depois extrai o segundo e o terceiro elementos e os vincula a símbolos `x`e `y`, respectivamente. O corpo da cláusula avalia `x`e `y`adiciona os resultados. Da mesma forma, a `call`cláusula implementa uma chamada de função e a `fn`cláusula implementa uma definição de função anônima.

As cláusulas restantes usam padrões centrais. `(pred numberp)`corresponde se `form`for um número. Na partida, o corpo avalia. `(pred symbolp)`corresponde se `form`for um símbolo. Na partida, o corpo procura o símbolo `env`e retorna sua associação. Por fim, `_`é o padrão abrangente que corresponde a qualquer coisa, portanto, é adequado para relatar erros de sintaxe.

Aqui estão alguns programas de exemplo nesta pequena linguagem, incluindo seus resultados de avaliação:

```
(avaliar '(adicionar 1 2) nil) ⇒ 3
(avaliar '(adicionar xy) '((x . 1) (y . 2))) ⇒ 3
(avaliar '(chamar (fn x (adicionar 1 x)) 2) nil) ⇒ 3
(avaliar '(sub 1 2) nil) ⇒ erro
```

Próximo:Desestruturando com padrões pcase, Anterior:Estendendo pcase, Acima:

Condicional de correspondência de padrões [Conteúdo][Índice]