A Syntax of Or-patterns and side conditions in P^+

rab

October 21, 2023

We extend an example grammar of patterns within uML with or-patterns and side conditions:

⟨*case-expression*⟩ ::= `(case` ⟨*expr*⟩ { ⟨*case-branch*⟩ } `)` ⟨*case-expression*⟩ ::= `(case` ⟨*expr*⟩ { ⟨*case-branch*⟩ } `)`

⟨*case-branch*⟩ ::= `(`⟨*pattern*⟩ ⟨*expr*⟩`)`

⟨*pattern*⟩ ::= ⟨*value-variable-name*⟩
| ⟨*value-constructor-name*⟩
| ⟨*value-constructor-name*⟩ {⟨*pattern*⟩}
| `(`⟨*pattern*⟩ `when` ⟨*expr*⟩ `)`
| `(oneof` {⟨*pattern*⟩}`)`
| _

1 Side conditions with `when`

The `when` keyword may optionally appear on the rightmost side of a `case` branch in P, within a set of parentheses also containing an expression. If the scrutinee matches the pattern, the expression is evaluated. If it evaluates to produce a truthy value, the match succeeds and the right-hand side expression is evaluated with the new ρ' produced by the pattern.

General concrete syntax of `when`:

```
(case scrutinee
    [pattern (when condition) rhs-exp])
```

Example:

```
(case v
    ['() 0]
    [(cons x xs) (when (= 0 (mod 2 x))) (+ 1 (count-evens xs)) ])
```

Note: the `exp` in a `when` is not limited to be a boolean expression, and there is no static type system to assert that it will evaluate to a boolean. As in the rest of P, when an expression evaluates to `#f`, it is considered falsey; otherwise, it is considered truthy.

2 Or-patterns with `oneof`

The `oneof` keyword may optionally appear on the leftmost side of a `case` branch in P, within a set of parentheses also containing the set of patterns for that branch. The set of patterns S is defined as such: if S contains a pattern p and the scrutinee matches p, that branch is evaluated if the pattern-matching algorithm reaches it. When the match succeeds and the right-hand side expression is evaluated with the new ρ' produced by a pattern, only that pattern's fresh variables are introduced into ρ'.

General concrete syntax of `oneof`:

```
(case scrutinee
    [(oneof pattern-1 pattern-2 ... pattern-k) rhs-exp])
```

Example:

```
(case light
    [RED 'stop]
    [(oneof GREEN YELLOW) 'keep-on-goin])
```

Typed languages with or-patterns, like OCaml, often have the restriction that logical variables introduced within a section of an or-pattern must represesent values of the same type within all parts of the or-pattern. Because P has no static type system, we don't make this assertion: whichever pattern in the or-pattern matches will introduce its variables and bindings into the ρ' with which the right-hand side is evaluated.
In addition, a fresh variable on the right-hand side of the or-pattern must appear in ALL branches of the or-pattern.
In V-, you can have defaults to your 'or-patterns' in a way you can't so much in P+, i.e. x can be a literal at the end of a list of unmatched patterns (or we can fail).
Example:

```
(case (list2 1 #f)
    ['() 0]
    [(oneof (cons 4 x) (cons x 3) (cons x #f)) x]) ;; returns #f
```