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:

$$\begin{array}{lcl}
\langle \textit{case-expression} \rangle & ::= & (\texttt{case}\ \langle \textit{expr} \rangle\ (\{\ \langle \textit{case-branch} \rangle\ \})) \\[1ex]
\langle \textit{case-branch} \rangle & ::= & (\langle \textit{pattern} \rangle\ \langle \textit{expr} \rangle\ [\text{side-condition}]) \\[1ex]
\langle \textit{pattern} \rangle & ::= & \langle \textit{value-variable-name} \rangle \\
 & | & \langle \textit{value-constructor-name} \rangle \\
 & | & \langle \textit{value-constructor-name –pattern''} \rangle \\
 & | & (\texttt{oneof}\ \langle \textit{pattern} \rangle\ \{\langle \textit{pattern} \rangle\}) \\
 & | & \_ \\[1ex]
\langle \textit{side-condition} \rangle & ::= & (\texttt{when}\ \langle \textit{expr} \rangle)
\end{array}$$

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 rhs-exp (when condition)])
```

Example:

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

A question about types:

I had a blurb like this:

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.

As I was writing this, I realized uML *does*, obviously, have a type system to do exactly this. At the same time, I remember you saying we won't have static types in our languages- which do you want to go off of?

2 Or-patterns with `oneof`

This again raises the question of the type system.

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])
```

A question about or-patterns and types:
The ocaml description of or-patterns is as follows:

The pattern $pattern_1 \mid pattern_2$ represents the logical "or" of the two patterns $pattern_1$ and $pattern_2$. A value matches $pattern_1 \mid pattern_2$ if it matches $pattern_1$ or $pattern_2$. The two sub-patterns $pattern_1$ and $pattern_2$ must bind exactly the same identifiers to values having the same types. Matching is performed from left to right. More precisely, in case some value v matches $pattern_1 \mid pattern_2$, the bindings performed are those of $pattern_1$ when v matches $pattern_1$. Otherwise, value v matches $pattern_2$ whose bindings are performed.

This is a restriction at the level of the type system. Again, do we want strict static types in P?