IIT CS440: Programming Languages and Translators

MiniML Language Spec

Prof. Stefan Muller

March 4, 2023

The grammar of MiniML is shown below.

op	→	`+` | `-` | `*` | `/` | $<$ | $\leq$ | $>$ | $\geq$ | `=` | `<>` | `&&` | `||` | `^`
τ	→	`int` | `string` | `bool` | `unit` | τ `list` | τ `->` τ | τ `*` τ
e	→	*var* | *num* | *string* | `true` | `false` | `()` | `[]` | e *op* e | `fun` *var* `->` e | `if` e `then` e `else` e
	|	`let` *var* $=$ e `in` e | `let` *var var* $=$ e `in` e
	|	`let rec` *var var* $=$ e `in` e | `let` $(var, var) = e$ `in` e | e e
	|	`match` e `with []` `->` e `|` *var*`::`*var* `->` e | e, e | e`::`e
decl	→	`let` *var* $=$ e`;;` | `let` *var var* $=$ *ekw*`;;` | `let rec` *var var* $=$ e`;;` | e`;;`
prog	→	*decl* | *decl prog*

You may notice that you can actually write a pretty large subset of OCaml in MiniML without making any changes. In particular, we've gotten rid of MicrOCaml's odd `app` e `to` e syntax and replaced it with normal OCaml application. One nice result of this is that, while you can't necessarily take any OCaml program and run it in MiniML, you *can* run any MiniML program through OCaml (`ocaml` or TryOCaml) to figure out what types it should have or what the result should be. A couple non-obvious restrictions present in MiniML:

1. Pattern matching is limited to using `let` to break apart pairs and using `match` to match on a list (note that we haven't defined `fst` or `snd`: you have to break apart pairs with pattern matching; you can define them yourself though).

2. Functions (both lambdas and let-defined functions) can only take one argument. You can get around this with currying (though that doesn't work well for recursive functions) or having functions take pairs (see `map` in `examples/rec.ml`).

3. As in OCaml, a program consists of one or more top-level declarations, where a declaration can be a `let` declaration or an expression by itself. Unlike in OCaml, these declarations **must** be followed by two semicolons.

4. There are no type annotations (e.g. `e : t`), on patterns or expressions.