

completing the parser for
Core Language

In Part 1 of the project we did not consider the two productions:

$\text{expr} \rightarrow \text{expr aexpr}$

and

$\text{expr} \rightarrow \text{expr1 binop expr2}$

the problem with the first production is that it is left-recursive and a naive parser following it would cause an infinite recursion

we transform the production into:

$\text{expr} \rightarrow \text{aexpr1} \dots \text{aexprn}$, for $n \geq 1$ which can be easily mimicked by the parser using the function `some`

for the application of the binop's we need to have many productions that model the different precedence of the different binop's. The precedences are summarized in the following table:

Precedence	Associativity	Operator
6	Left	Application
5	Right	*
	None	/
4	Right	+
	None	-
3	None	== ~ = > >= < <=
2	Right	&
1	Right	

in order to model the associativity and precedence of the different binop's we follow the idea that we have studied in chapter 13.8 of the text
the new productions for expr are in the next slide

$expr$	\rightarrow	<code>let defs in expr</code>	
		<code>letrec defs in expr</code>	
		<code>case expr of alts</code>	
		<code>\ var₁ ... var_n . expr</code>	
		<code>expr1</code>	
$expr1$	\rightarrow	<code>expr2 expr1</code>	
		<code>expr2</code>	
$expr2$	\rightarrow	<code>expr3 & expr2</code>	
		<code>expr3</code>	
$expr3$	\rightarrow	<code>expr4 relop expr4</code>	
		<code>expr4</code>	
$expr4$	\rightarrow	<code>expr5 + expr4</code>	
		<code>expr5 - expr5</code>	
		<code>expr5</code>	
$expr5$	\rightarrow	<code>expr6 * expr5</code>	
		<code>expr6 / expr6</code>	
		<code>expr6</code>	
$expr6$	\rightarrow	<code>aexpr₁ ... aexpr_n</code>	$(n \geq 1)$

Figure 1.3: Grammar expressing operator precedence and associativity

important advice: since `aexpr` can be simple variables, and, following the last production in the previous table, the parser will search the input for `n aexpr`, $n > 0$, it is necessary that the function that recognizes variables, is able to distinguish variables from keywords of the Core Language, such as `in`, `of`, `let`, etc. Otherwise you risk that such keywords are mixed up with `aexpr` which may cause the failure of the program.