Academia vs. Business

I just wrote the most beautiful code of my life.

They casually handed me an impossible problem. In 48 hours and 200 lines, I solved it.

Academia:
My god... this will mean a half-dozen papers, a thesis or two, and a paragraph in every textbook on queuing theory!

Business:
You got the program to stop jamming up? Great while you're fixing stuff, can you get Outlook to sync with our new phones?

https://xkcd.com/664/
Translation from string to concrete program piece

```plaintext
type concrete_program_piece =
  Number of int
  | Symbol of string
  | List of concrete_program_piece list
```

<table>
<thead>
<tr>
<th>string</th>
<th>concrete_program_piece</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;17&quot;</td>
<td>Number 17</td>
</tr>
<tr>
<td>&quot;sq&quot;</td>
<td>Symbol &quot;sq&quot;</td>
</tr>
<tr>
<td>&quot;(sq 17)&quot;</td>
<td>List [Symbol &quot;sq&quot;; Number 17]</td>
</tr>
<tr>
<td>&quot;(define x 1)&quot;</td>
<td>List [Symbol &quot;define&quot;; Symbol &quot;x&quot;; Number 1]</td>
</tr>
<tr>
<td>&quot;(quote hi)&quot;</td>
<td>List [Symbol &quot;quote&quot;; Symbol &quot;hi&quot;]</td>
</tr>
<tr>
<td>&quot;(quote (1 2))&quot;</td>
<td>List [Symbol &quot;quote&quot;; List [Number 1; Number 2]]</td>
</tr>
</tbody>
</table>
type raw_program = string

type concrete_program_piece =
  Number of int
  | Symbol of string
  | List of concrete_program_piece list

type concrete_program = concrete_program_piece list

read: raw_program -> concrete_program_piece

# read "17" ;;
- : concrete_program_piece = Number 17

# read "sq" ;;
- : concrete_program_piece = Symbol "sq"

# read "(sq 17)" ;;
- : concrete_program_piece = List [Symbol "sq"; Number 17]

# read "(quote (1 2))" ;;
# : concrete_program_piece
# read "17" ;;
- = List [Symbol "quote"; List [Number 1; Number 2]]
# read "sq" ;;
- : concrete_program_piece = Symbol "sq"
The string

"(define mylist (quote (1 2))) (+ 5 (car mylist))"

gets translated by read_all into the list

List
  [List [Symbol "define"; Symbol "mylist";
    List [Symbol "quote"; List [Number 1; Number 2]]];
  List [Symbol "+"; Number 5; List [Symbol "car"; Symbol "mylist"]]]
type abstract_program_piece =
    Definition of definition
    | Expression of expression

type definition = identifier * expression

parse_expression:
    concrete_program_piece -> expression

<table>
<thead>
<tr>
<th>concrete_program_piece</th>
<th>expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number 17</td>
<td>NumE 17</td>
</tr>
<tr>
<td>Symbol &quot;sq&quot;</td>
<td>IdentE (ID &quot;sq&quot;)</td>
</tr>
<tr>
<td>List [Symbol &quot;sq&quot;; Number 17]</td>
<td>ApplicationE [IdentE (ID &quot;sq&quot;); NumE 17]</td>
</tr>
<tr>
<td>List [Symbol &quot;quote&quot;; Symbol &quot;hi&quot;]</td>
<td>QuoteE (Symbol &quot;hi&quot;)</td>
</tr>
<tr>
<td>List [Symbol &quot;quote&quot;; List [Number 1; Number 2]]</td>
<td>QuoteE (List [Number 1; Number 2])</td>
</tr>
</tbody>
</table>
type expression =
  NumE of int
| IdentE of identifier
| AndE of expression * expression
| OrE of expression * expression
  ..... 
| CondE of (expression * expression) list
| QuoteE of concrete_program_piece
| LambdaE of identifier list * expression
| ApplicationE of expression list
type value =
  VNum of int
| VBool of bool
| VSymbol of string
| VList of value list
| VBuiltin of string * (value list -> value)
| VClosure of identifier list * expression * environment

<table>
<thead>
<tr>
<th>string</th>
<th>expression</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;5&quot;</td>
<td>NumE 5</td>
<td>VNum 5</td>
</tr>
</tbody>
</table>
| "(or true x)" | Or (IdentE (ID "true"),
  IdentE (ID "x"))       | VBool true |
| "(+ 1 2)"  | ApplicationE [IdentE (ID "+");
  NumE 1; NumE 2]              | VNum 3     |
symbols bound in initial top-level environment

+ - * / remainder = <

cons car cdr not