Motivating Application

```plaintext
procedure Concata(a: Node; b: Node)
returns (res: Node)
requires laeq(a, null) & a+ laeq(b, null)
enforces lea(eq(a, null))
{ 
  if (a = null) { 
    return b; 
  } 
  size i 
  var cur := a; 
  while (cur.next != null) { 
    cur := cur.next; 
  } 
  return a; 
}
```

Output Representation

**logical description** of the instantiated data structures,
(i.e., formal description of a set of allowed heaps)

Inductive predicates: Binary tree of “panhandle lists”

```plaintext
e.g. ls(x, y, f), tree(x, y)
```

Given a predicate, we allow nested subformulas

```plaintext
\( \exists\exists!ls(i_2, t, i_5, i_6, i_7, i_8 \rightarrow T) \)
```

Problem Formalization

To automatically predict a separation logic formula from a given heap \( H \)

### Separation logic formulas

```plaintext
Formula → Heaplets | \( \exists \) Vari | Var, Var, Heaplets | ...
Heaplets → T | Heaplet + Heaplets
Heaplet → i = expr | var, var, var, var, var → Formula
\( \text{tree} \) (expr, var, var, var, var → Formula)
```

```
\( \exists\exists!\text{ls}(i_2, t, i_5, i_6, i_7, i_8 \rightarrow T) \)
```

Parse tree

```
F = (A, g(\cdot)).
```

Partial tree \( T_{\text{const}} \): Parse tree restricted to nodes \{1, \ldots, n\}

Problem Statement

Sequentially predict the next node, conditional upon everything that has been predicted so far:

```
P(T) = \prod_{a \in \mathcal{A}(\mathcal{N})} p(ch(a) \mid H, T < a).
```

Experimental Results

**Features**

- **Heap features**
  - Heaplet frequency
  - depth of the current nonterminal node
  - if variable \( y \) is reached

- **Partial tree features**
  - (int) \( 1 \text{-gram} \)
  - depth of the current nonterminal node
  - if variable \( y \) is accounted for

- **Joint features**
  - (bool) graphic is cyclic
  - (string) predecessor token
  - (string) relation to defined predicates

**Prediction model:** Random Forests, Multiclass Logistic Regression, Multiclass Neural Network

**Predicates used:** lists, cyclic lists, panhandle lists and trees.

<table>
<thead>
<tr>
<th>Nesting level</th>
<th>Number of free variables</th>
<th>Number of Formulas</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>127</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>33254</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>3615</td>
</tr>
</tbody>
</table>

We sample 1757 formulas from the above table (last row)
500 heap graphs are generated per formula.
878,500 formula/heap graph combinations.

Example input graph from dataset which is corrected predicted.

```
(\text{noted level} 0, \text{number of free variables} 4)
```

Training / validation / testing sets: 6:2:2

<table>
<thead>
<tr>
<th>Method</th>
<th>Top 1 Acc.</th>
<th>Top 10 Acc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Concatenation features</td>
<td>0.05%</td>
<td>0.07%</td>
</tr>
<tr>
<td>Joint features</td>
<td>91.5%</td>
<td>91.6%</td>
</tr>
</tbody>
</table>

One can then use static program verification tools to determine whether the description is accurate and whether the program satisfies memory safe.