1. INTRODUCTION AND MOTIVATION

Setup defines a syntax for operating on finite sets. Setup provides intuitive notation for quickly and clearly defining sets, as well as performing rudimentary set operations on user-defined sets. Setup also defines a notation for for functions which take literals and sets as parameters.

Setup provides a level of abstraction to the user which makes set manipulation more intuitive. We anticipate users will solve simple set-oriented problems like schedule, rudimentary databases, and probability problems.

2. LANGUAGE FEATURES

2.1 DATA TYPES

Literals / Atoms
- Integers -- [0-9]+
- Float -- Integer.[Integer]+ uses the 32 bit IEEE range
- Character -- A - z, no punctuation or white space
- Strings -- [Character]+
- Symbols -- Globally unique names that may be members of Sets or Tuples

Sets : homogeneous, all elements of the same type, unique values
Tuples or Lists : ordered lists, heterogeneous, can be of mixed type, duplicate values permitted

2.2 KEYWORDS AND OPERATORS

Setup allows for the usual four operations {+,-,*,/} on integer and float types, as well as the following operators for set types:

<table>
<thead>
<tr>
<th>Setup Operator</th>
<th>Mathematical Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>intersect</td>
<td>∩</td>
<td>computes the intersection of the sets to the left (lhs) and right (rhs) of it</td>
</tr>
<tr>
<td>union</td>
<td>∪</td>
<td>computes the union of lhs and rhs</td>
</tr>
<tr>
<td>minus</td>
<td>-</td>
<td>returns lhs with any members of rhs removed</td>
</tr>
<tr>
<td>cross</td>
<td>×</td>
<td>returns the cartesian product of lhs and rhs</td>
</tr>
<tr>
<td>in</td>
<td>∈</td>
<td>iterates over members of rhs</td>
</tr>
<tr>
<td>not</td>
<td>~</td>
<td>returns complement</td>
</tr>
<tr>
<td>#</td>
<td></td>
<td></td>
</tr>
<tr>
<td>:=</td>
<td>:=</td>
<td>assignment</td>
</tr>
<tr>
<td>sum</td>
<td>operates only on numeric sets and returns sum of elements (done coordinate-wise) in the set</td>
<td></td>
</tr>
<tr>
<td>and</td>
<td>arranges cross product pairings from sets on the left and right</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>range operator applies to integers and characters</td>
<td></td>
</tr>
<tr>
<td>{}</td>
<td>denotes a set of elements</td>
<td></td>
</tr>
<tr>
<td>()</td>
<td>denotes an ordered list, or tuple. the cross product of two sets is a set of tuples.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>where, as in SQL, in a Setup clause, the expression to the left of of</td>
<td>declares variable names and their structural relationships, while the expression on the right binds variables to values</td>
</tr>
<tr>
<td>--</td>
<td>begins a comment. comments begin with -- and end with a new line</td>
<td></td>
</tr>
<tr>
<td>.</td>
<td>converts lhs and rhs to string representation and returns their concatenation. (all types have a string representation)</td>
<td></td>
</tr>
<tr>
<td>*</td>
<td>wildcard is a placeholder that accepts any value without binding it to a variable name or checking its type</td>
<td></td>
</tr>
<tr>
<td>;</td>
<td>statement terminator</td>
<td></td>
</tr>
<tr>
<td>[]</td>
<td>in function declarations, groups input arguments and statements in function body</td>
<td></td>
</tr>
</tbody>
</table>

3. Functions
We anticipate functions having no side effects on their arguments. Functions accept as arguments literals and their containers (i.e., sets, sets of sets).

3.1 Function Syntax

3.1.1 Definition

```plaintext
function FuncName [set x, int c] returns set
|
  statement;
  statement;
  return ret;
|
```

3.1.2 Invocation

```
FuncName [Week, 7];
```

4. Sample Code
4.1 **SET INITIALIZATION**

4.1.1 **Initialization using literals and tokens:**

\[
\begin{align*}
\text{Hours} & := \{1 \ldots 24\}; \\
\text{Weekdays} & := \{\text{Mo Tu We Th Fr}\}; \\
\text{Weekend} & := \{\text{Sat Sun}\};
\end{align*}
\]

4.1.2 **Initialization Built-in Operators:**

\[
\begin{align*}
\text{FullWeek} & := \text{Weekdays union Weekend}; \\
& \quad -- \{\text{Mo Tu We Th Fr Sat Sun}\}
\end{align*}
\]

\[
\begin{align*}
\text{WeekdayHrs} & := \text{Weekdays cross Hours}; \\
& \quad -- \{(Mo 1) \ (Mo 2) \ldots (Fr 24)\}
\end{align*}
\]

4.1.3 **Initialization Using Relations:**

\[
\begin{align*}
\text{WeekdayHrs} & := \{(x \ y) \mid x \in \text{Weekdays} \text{ and } y \in \text{Hours}\}; \\
& \quad -- \{(Mo 1) \ (Mo 2) \ldots (Fr 24)\}
\end{align*}
\]

\[
\begin{align*}
\text{TokenWeekdayHrs} & := \{\text{"day". str(x) . "-hr". str(y)} \mid x \in \text{Weekdays} \text{ and } y \in \text{Hours}\}; \\
& \quad -- \{\text{dayMo-hr1 \ldots dayFr-hr24}\}
\end{align*}
\]

\[
\begin{align*}
\text{MondayHrs} & := (\text{Mo } *) \text{ in WeekdayHrs}; \\
& \quad -- \{(Mo 1) \ (Mo 2) \ldots (Mo 24)\}
\end{align*}
\]

\[
\begin{align*}
\text{Hours} & := \{x \mid (\ast \ x) \text{ in WeekdayHrs}\}; \\
& \quad -- \{1 \ldots 24\}
\end{align*}
\]

\[
\begin{align*}
\text{TreeWeek} & := \{ \ (d \{h}\) \mid d \text{ in Weekdays and } h \text{ in Hours} \}
\end{align*}
\]

\[
\begin{align*}
& \quad -- \{(Mo \{1 \ldots 24\}) \ldots (Fr \{1 \ldots 24\})\}
\end{align*}
\]

4.2 **SAMPLE PROGRAM**

Users may want to use *Setup* to solve problems related to probability. The following program computes the expected value of a roll of a fair dice. It can be extended simply to solve harder problems relating to conditional probability and random walks.

<table>
<thead>
<tr>
<th>Program</th>
</tr>
</thead>
<tbody>
<tr>
<td>function ExpVal [ set S ]</td>
</tr>
<tr>
<td>[]</td>
</tr>
<tr>
<td>Temp := {x*y</td>
</tr>
<tr>
<td>return sum Temp;</td>
</tr>
<tr>
<td>]</td>
</tr>
<tr>
<td>Pips := {1 2 3 4 5 6};</td>
</tr>
<tr>
<td>Prob := { 1/6 };</td>
</tr>
<tr>
<td>Dice := Pips cross Prob; -- {(1/6) ... (6 1/6)}</td>
</tr>
<tr>
<td>print ExpVal [ Dice ]; -- 3.5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.5</td>
</tr>
</tbody>
</table>