Roller language specification draft

mrZalli

November 7, 2015

This is a draft of the language for the interactive parser Roller. No fully functional parser exists yet. I'm sorry if this looks hard to understand since this is mainly meant for me to gather my thoughts.

1 Values

The program can operate on a limited set of built-in types. These types can be put into lists.

1.1 Single values

There are three different types of values: numerals, strings and errors. Numerals can further be split into integers and reals (floating point). Single values can be implicitly converted to single value lists.

- Numeral
  - An integer or real value.
  - If in a mathematical operation any of the operands are real numbers, the result is a real.
  - If in a mathematical operation both of the operands are integers, then the result is an integer, except in division where the result is a real.
- String
  - From mathematical operations, only addition operation is allowed for strings.
    - Addition concatenates two strings, or a string and a numeral into a larger string.
- Error
  - A signal about erroneous execution in a command.
  - Contains the error type and an error message about the nature of error.

- Can only be created with the $Raise$ keyword and can be catched with the Try and $Catch$ keywords.
  - No other operations can be done to the error other than returning or printing it
- Evaluating an error usually returns from the expression or statement unless catched.

1.2 Lists

Lists are values inside curly brackets and separated by commas. For example: $\{a, b, c, d, e\}$. A list can have lists as members, called child-lists.

1.2.1 List operations

Operations meant for single values are applied recursively on every list member separately when used on lists. Lists can be implicitly converted to single values, which calls the lists sum.

1.2.2 Ranges

Ranges are flat numeral lists that start from the start value and end on the end value, with each consecutive value incrementing or decrementing by the step value until the end value is reached.

Ranges are defined with $start, next..end$ where $start$ is the initial value, end is the last value, or a greater value than the last value, and $next$ is the initial value plus the step value. Depending on the step value, the range might not contain the end value, but it will never contain any value with a greater absolute value than end.

If the $next$ parameter is omitted, eg. $start..end$, a step value of +1 or -1 is assumed depending whether $start$ is lesser than end or not. If the step value has the wrong sign (eg. $0, 4..(-12)$) an error is raised.

1.3 Variables

A variable can hold any value. Variable identifier is recognized by the regular expression $([a-zA-Z]|\_)$, so they can contain only letters and underscores.

Variables are stored in the environment and to change them you need to use statements.

1.4 Functions

Functions are repeatable expressions with a zero or more parameters. Functions always return a value, or cause an error.

Parameter is the function's local variable, while argument is the value given to the parameter on function call. Variables and function names must be unique.

1.4.1 Parameters

By default all variables are in the global namespace, but functions can define parameter variables that are in the function's namespace.

If there is a function parameter with the same name as a global variable, the parameter hides the global variable. No type checking is done to the arguments put into parameters.

2 Commands

The Roller application takes commands as an input. There are two types of commands: expressions and statements.

Expressions return a value and don't cause side-effects, while statements don't return a value and cause side-effects. If the command is an expression, it's return value is printed.

2.1 Expressions

All expressions must return a value except if the expression encounters an error.

In that case, the execution of the command the expression was in, must be stopped and the user should be informed.

2.1.1 Expression types

- Value expression
  - This constructs a value
  - Value can be a single value or a list.
  - The value types are defined in the part 1
- Dice expressions
  - An expression that represents dice throw with multiple similar die.
  - Uses the dice notation, so $N\mathrm{d}M$ means throw N amount of M sided die, if both N and M are integers. If the parameters are not present the defaults are used.
    - By default the amount is 1 and the sides are 6, so d equals to $1\mathrm{d}6$.
  - If the sides parameter is a list, then each of the dice face is one of the list members.
    - Example: $1\mathrm{d}\,\{"heads", "tails"\}$ either gives the string "$heads$" or the string "$tails$" as the result.
  - If the amount parameter is a list of integers, then the expression returns a list where each list member is a list of that many dice throws as the corresponding parameter integer.

    - Example: $\{2, 4, 3\}$ d6 is equivalent to $\{2\text{d}6, 4\text{d}6, 3\text{d}6\}$.

- Mathemathical operators
  - All mathematical operators are infix, except the negation operator.
  - All operators behave normally between single values (scalars)
  - If one of the operands is a list and the another is a single value, then the operation is applied to the every element on the list recurisvely
    - Example: $a + \{b, c\} \rightarrow \{a + b, a + c\}$
  - If both of the operands are lists then the operator is applied between each element. If another list is longer than another, then the remaining elements are ignored.
    - Example: $\{a, b, c\} * \{d, e\} \rightarrow \{a * d, b * e\}$

- Keyword expressions
  - All keywords start with a capital letter followed by non-capital or capital letters
  - All keyword expressions consist of a keyword and the parameters. The keyword is always before the parameters
  - Unlike functions, keyword expression's arguments are not surrounded by parentheses.
  - The different keywords and their functions are defined in 2.1.2

- If expression
  - Syntax: $If\, a\, Then\, b\, Else\, c$
  - The else part is required since this is an expression and has to return a value
  - Returns b if the condition expression a is not an empty list ($\{\}$), otherwise returns expression c.
  - Use with the filtering expressions
  - Example: $If\ foo[> 3]\ Then\ 1\text{d}20 + 2\ Else\ 3\text{d}6$
  - Example: $If\ a[= 1]\ Then$"a is one" $Else\ If\ a[= 2]\ Then$"a is two" $Else$"a is not one or two"

- Switch/Case expression
  - Syntax: $Switch\, a\, Case\, value\ :\ expression \ldots Default\ :\ c$
  - Compares the value of the expression a against all of the $value$ parts of the case-parts and the $expression$ of the first matching case part is returned.
    - If nothing matches the default part is returned.

- Namespace calls
  - To use the value of a variable write it's name. If there is no variable with that name an error is given.
    - Example: $foo + 1$
  - To call a function write it's name, followed by it's arguments, if any, in parenthesis.
    - Example: $foo(1, 6)$
- List filtering expression
  - The lists can be filtered with the filtering expression.
  - The filtering expression consists of the filtered expression and the predicate in square brackets.
    - Example: $Count\, 8\mathrm{d}6[>= 5]$ counts all the dice throws that were five or six.
  - Returns a list of the members that fit the for each of the predicates in the list.
  - Filtering considers the list to be flat and the filtering is not done to the members of the child lists.
  - The different predicates are defined in the part 3

2.1.2 Different operator and keyword expressions

- Mathemathical operators (precedence is as expected)
  - Addition/String concatenation $(a + b)$
  - Substraction $(a - b)$
  - Multiplication $(a * b)$
  - Division (a/b)
    - Result is a real numeral.
  - Exponent $(a\hat{}b)$
  - Negation $(-a)$
    - Note: $-a\hat{}b$ means $-(a\hat{}b)$
- Keyword expressions
  - $Count\, a$
    - Gives the length of a list.
  - $Sum\, a$
    - Gives the sum of a list

    * If any of the members are a string, the sum is all of the values converted to strings and concatenated with spaces between.
- $Mean\, a$
    * Only defined on numeral values
    * Returns the mean value of a numeral list.
- $Sqrt\, a$
    * Only defined on numeral values
    * Returns the square root of a recursively
- $Root\, n\, a$
    * Only defined on numeral values
    * Returns the n:th root of a recursively
    * If n is a list return the roots for all it's lists recursively
- $Repeat\, n\, a$
    * Repeats the a expression n times and returns a list of the b's value for each repetition.
    * The first parameter must be an integer or an integer list.
    * If the first parameter is an integer list then the repeat is done for each of it's members recursively.
        · Example: $Repeat\, \{3,\, 2\}$ 1d20 is equivalent to $\{Repeat\, 3$ 1d20, $Repeat\, 2$ 1d20$\}$. It is also equivalent to $\{3,\, 2\}$ d20
- $Acc\; function\; sequence$
    * Accumulates the $function$ over the $sequence$
    * Example: $Accadd\, \{1, 2, 4, 3\}$, where $add(a,\, b) = a+b$, is equivalent to $Sum\ \{1, 2, 4, 3\}$
- $Flatten\, a$
    * Converts all the child lists of a into single values
    * Example: $Flatten\, \{a,\, \{b,\, \{c,\, d\}\},\, \{e\}\} \rightarrow \{a,\, b + c + d,\, e\}$
- $ToFlat\, a$
    * Flattens the a list so that it contains no child lists
    * Example: $Flatten\, \{a,\, \{b,\, \{c,\, d\}\},\, \{e\}\} \rightarrow \{a,\, b,\, c,\, d,\, e\}$
- $ToString\, a$
    * Returns a string representation of a
- $ToNumeral\, a$
    * If a is a string this returns it's numeral representation. If the string doesn't contain a numeral (integer or floating point) this generates an error
    * If a is a numeral list this returns it's sum

    - Otherwise generates an error
  - $ToList\, a$
    - Defined for strings
    - Converts the string a to a list of letters
    - The letters are single character strings
  - $Raise\, a$
    - Creates an error with a as it's error message.
    - Evaluating this effectively terminates the expression
  - $Try\, a\, Catch\, :\, b$
    - Evaluates a and if it returns an error returns b, otherwise returns the value of a.

2.2 Statements

Statements have side-effects, which means that they change the state of the environment, more specifically the namespace. The statements can raise errors.

2.2.1 Variable statements

- Variable assignment, $name = expression$
  - Creates or replaces a variable with the of $name$ with the evaluated value of $expression$
- Assignment and operation, $+=$, $-=$, $*=$, $/=$
  - Does the operation with the variable and the expression as operands and places the result into the variable.
    - Example: $foo+=4$ is equivalent to $foo = foo + 4$
  - If the variable doesn't exist this raises a $UndeclaredVariable$ error
- Delete statement, $Delete\, name$
  - Deletes the variable or function $name$.
  - If the operand doesn't exist this raises a $Undeclared\, Name$ error
- Try and catch statement, $Try\, a\, Catch\, :\, cmd$
  - If the statement a raises any error, then the command cmd is executed, which can be an expression or statement.
  - Works like the try and catch expression, except on statements.
  - Doesn't return a value, but if the cmd is an expression then it's value is printed as normal.

2.2.2 Function definition

Functions are defined like: $name() = expression$ or $name(parameterList) = expression$.

The $parameterList$ is a comma separated list of parameters, which are variables local to the function and assigned values on the function call.

The $expression$ is evaluated when the function is called.

Function and variable names should be unique and defining a function or variable with the same name as any existing function or variable raises an error.

3 Predicates

Predicates define which values are passed through when filtering a list.

If there are multiple predicates, separated by commas, the filtering branches and it is done for all of the predicates and returned in a list.

List of the precedence levels different predicates have:

Precedence level	Predicates
Level 1, lowest precedence	Logical connectives, except negation
Level 2	Comparison predicates
Level 3, highest precedence	Type predicates, indexing predicates, negation

3.1 Logical connectives

Logical connectives connect different predicates. Available connectives are:

- Conjunction/and, $a \,\&\, b$
  - Defined true iff (if and only if) both of the operand predicates (a, b) are true.
- Disjunction/or, $a \mid b$
  - Defined true iff any of the operands are true.
- Exclusive disjunction/xor, $a \hat{} b$
  - Defined true iff one of the operands are true.
- Negation/not, $!a$
  - Defined true iff the operand is false.
  - Has high precedence

3.2 Comparison predicates

- Equality, $= a$
  - Inequality can be achieved by combining the equality and negation predicates.
- Comparison, $< a$, $> a$, $<= a$, $>= a$

3.3 Type predicates

These predicates hold true only for certain type of values

- Integer predicate, $\#$
- Real predicate, $\%$
- String predicate, $
- List predicate, $\{\}$, $\{Predicate\}$
  - Matches child lists
  - The empty list predicate $\{\}$ matches any list.
  - The $Predicate$ is almost any predicate that tells what kind of list we match. It must hold true for all of the members of the child list.
    - The $Predicate$ can't contain single values because that would be mixed with list indexing predicate (3.4)
  - Examples:
    - $\{\#\}$ matches integer lists
    - $\{\#|\%\}$ matches lists with integer and/or real values (numerals)
    - $\{!\{\}\}$ matches lists that don't have any list members
    - $\{< 5\}$ matches lists with only numeral variables that are smaller than five

3.4 Indexing predicates

These predicates hold true only for a certain index or range of any list. They are normal expressions.

- $Value$
  - $Value$ is not a list.
  - Holds true only for the list member that is at the index $Value$.
  - For normal lists the indexing starts from one. This is because dice ranges start from one as well.
  - Example: $\{4,2,3\}[1] \rightarrow \{4\}$

- $List$
  - Holds true for the values whose indexes are in the $List$.
  - Example: $\{4,2,3\}[\{1,2\}] \rightarrow \{4,2\}$, whereas $\{4,2,3\}[1,2] \rightarrow \{\{4\},\{2\}\}$

4 Errors

An error is raised when the program runs into an erroneous situation and can't continue. If an error happens during a statetement, the program state shouldn't be changed by that statement.

Information of the type of the error with more info and suggestions should be given to the user. The error messages should be given in a human-readable form.

Suggested error message format is: "([Error number]) [Error name]: [Error message], at "[Error location]" ", for example "(11) Arithmetic error: division by zero, at "1/x""

4.1 Parse error

If the written input doesn't follow the semantic syntax of the language an parse error should be raised. Additional information could be what position the error was encountered.

Various possible parse errors:

- Invalid token error
  - When the lexer finds an invalid token
  - Example: "Invalid token error: letters in an integer value, at "41a3"
  - Example: "Invalid token error: invalid character, at "böö"
- Semantic error
  - When the semantic analyzer finds an invalid structure
  - Example: "Semantic error: unclosed parenthesis, at "(1+2"