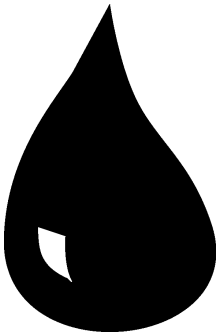
DruL Final Report
COMS W4115: Programming Language and Translators



Team Leader: Rob Stewart (rs2660)
Thierry Bertin-Mahieux (tb2332)
Benjamin Warfield (bbw2108)
Waseem Ilahi (wki2001)

December 19th, 2008

Contents

Chapter 1

Language White Paper

1.1 Introduction

DruL stands for "Drumming Language". It is a programming language designed for composing drum music. It is common these days for drum beat composers to create drum parts using computer software (e.g. FL Studio). Creating drums parts with these programs often involes of lot of tedious "pointing and clicking" (especially when making longer drum parts). DruL was designed to give the composer the ability to automate much of this tedium. There already exist other more general-purpose music programming languages (e.g. ChucK, SuperCollider, Nyquist, Haskore). These languages are complicated by note pitches, durations, and audio effects. DruL is unconcerned with these things and focuses soley on allowing the drum composer to define and manipulate beat patterns.

DruL meets the needs of algorithmic drum-composers with the beat, pattern, and clip domain-specific data-types. A pattern is essentially an object that holds binary, discrete, time-series data. At each discrete-time step, which will henceforth refer to as a beat, there is either a note or a rest. For the non-musically inclined, a note represents sound produced by the striking of a drum (or similar instrument) and a rest represents the absence of any such sound. Patterns are immutable. When a pattern is manipulated, the target pattern remains intact and a new copy is created. An instrument is one of a pre-defined set of sounds (e.g. drum notes) that can occupy a single beat. A clip is a mapping of patterns to instruments. Clips are processed in sequence as the program runs to produce output which may be audio, sheet-music notation, or a MIDI file.

DruL is mainly an imperative programming language, however it borrows ideas (map) from the functional paradigm. DruL is strictly and dynamically typed. DruL programs do not contain any loops or user-defined functions. Rather, DruL uses *map* and *mapper* defined below.

A composer typically starts a DruL program by defining some initial patterns. These patterns can then be individually processed by built-in DruL functions to produce new patterns. Alternately, the composer may define and use new functions called mappers. Composers then apply their mappers to patterns, iterating over the beats of one or more patterns at a time, building up a new pattern along the way. Once the composer has a set of patterns with which they are happy, they define their desired set of instruments (e.g. hi-hat, snare, bass drum, cowbell, etc.). With the instruments defined, the composer uses the clip constuct to assign a pattern to each instrument. Finally, the clip is output to a MIDI file, playable by many multimedia players.

1.2 Language specification

There are 3 main data types in DruL: **int**, **pattern**, and **clip**.

Keywords are white space delimited. Indentation is not significant. Function arguments are enclosed in parentheses and comma-separated.

Anything remaining on a line after // is a comment will be ignored by the compiler.

A *map* takes one or many patterns, and iterates over beats on all of them at the same time, from the first beat to the last beat of the longest sequence.
A *map* returns a pattern (that can be empty). Inside the map, per each beat, a may pattern returned which is then appended to an accumulated pattern. This accumulated pattern is then returned by the map after the final iteration.

return	rand	clip	mapper
if	pattern	instrument	print
elseif	concat	map	
else	true	false	

Scopes: There is a general scope, and one scope per *map*. Variables in the general scope can be seen from within a *map*, but not written to. Variables defined in a *map* are garbage collected at the end of the *map*.

1.3 Quick tutorial

In this section we give examples of what DruL code will look like, in the form of a tutorial.

1.3.1 Integers

Integers are part of our language.

```
a = 3;
b = a + 2;
c = b * 12;
```

1.3.2 Pattern

Patterns are the data type the programmer will likely spend most of their time dealing with. For convenience, the programmer can supply a string constant made up of 1s and 0s, which will be translated into a pattern: if the character is a 1, there is a note on the corresponding beat; if 0, a rest.

```
p1 = pattern("101010");
```

Patterns can be concatenated to form new patterns:

```
pcat = concat(p1, pattern("111000"), pattern("1"));
```

pcat will be equal to 1010101110001.

There is also a shortcut to concatenate the same pattern many times:

```
pcat2 = concat(p1, p1, p1);
pcat3 = pattern("101010").repeat(3);
pcat4 = p1.repeat(3);
```

pcat2, *pcat3*, and *pcat4* are all equivalent.

1.3.3 Map

Of course, we will not hardcode every pattern we want to create. We use map to create meaningful new patterns from existing ones:

```
p2 = map(p1)
{
    if ($1.note()) { return pattern("11"); }
    else           { return pattern("0");  }
};
```

This will create the following pattern: 110110110. The goal of a map is to easily iterate over a pattern. *p1.note* returns *true* if there is a note on the current beat, *false* otherwise. If you call map on multiple patterns that are not of the same length, the shorter patterns will be padded with *NULL* beats.

1.3.4 Mapper

For ease of use, you can define a *mapper* that contains the behavior used by *map*. We create *p3*, which is the same as *p2*:

```
mapper myMapper(p)
{
    if (p.note) { return pattern("11"); }
    else        { return pattern("0");  }
}

p3 = map(p1) myMapper;
```

mapper will be very important when building a standard library for the language.

1.3.5 More complex examples

Now that we have a proper syntax, let's get to more complicated examples. We introduce 2 new features that can be used inside a *map*: *prev* and *next*. They give

you access to earlier and later beats in a pattern, using the syntax *p.prev(n)* and *p.next(n)*. Also, for a pattern *p*, *p.rest()* is true if and only if we did not reach the end of this pattern.

reduction: accelerate by cutting one beat out of two

```
downbeats = pattern("1000100010001000");
alternate_beats = pattern("10").repeat(8);
downbeat_diminution = map(downbeats, alternate_beats)
{
    if     ($2.rest()) { return pattern("");  } // pattern of length 0
    elseif ($1.note()) { return pattern("1"); }
    else               { return pattern("0"); }
}
```

output is: 10101010.

improved reduction: putting a rest (0) only if the 2 original beats were rest

```
// this will map "1001100110011001" to "11111111", rather than "10101010"
one_and_four    = pattern("1001100110011001");
alternate_beats = pattern("10").repeat(8);
improved_diminution = map(one_and_four, alternate_beats)
{
    if     ($2.rest())         { return pattern("");  }
    elseif ($1.note())         { return pattern("1"); }
    elseif ($1.next(1).note()) { return pattern("1"); }
    else                       { return pattern("0"); }
};
```

1.3.6 Instruments and Clips

Now that we have a large and varied collection of patterns, we can show how to combine those patterns into clips.

Before we define any clips, we must tell the compiler what instruments they will use. This can only be done once per program, and uses the *instruments* function:

```
instruments("hihat", "bassdrum", "crash", "snare");
```

Once the instruments are defined, we can create a clip from our existing patterns, using an associative-array notation:

```
clip1 = clip
(
    "bassdrum" <- downbeats,
    "hihat"    <- alternate_beats
);
```

The same result can be achieved by simply listing the patterns for each instrument in the order they are defined in the *instruments* declaration:

```
clip2 = clip
(
    alternate_beats,
    downbeats
    // remaining instruments have an empty beat-pattern
);
```

Chapter 2

Tutorial

2.1 Introduction

In this section we present some quick tutorials that explain the main features of DruL language. We start with the basics: "Hello World" is in Section 2.2.1, integers and if/else are shown in 2.2.2 and finally patterns are introduced in 2.2.3. That being done, we learn how to combine patterns (2.3), manipulate patterns (2.4, create a named mapper (2.5), create a clip (2.6), and finally how to bring all this together and create a sheet of music (2.7).

2.2 The Very Basics

The `drul` command can take DruL code either from the standard input or as a file specified on the command line (`drul mysource.drul`). Examples in this section should work equally well if passed in either way.

2.2.1 Say hello!

Because it is traditional, albeit almost completely irrelevant to this language, here is our first DruL program:

```
print("hello, world!");
```

This will print the string "hello, world!" to the standard output, on a line by itself. Note that unlike many languages, DruL does not require you to place a newline character at the end of a string to have it print on a single line (conversely, it gives you no method to print *without* a newline at the end).

2.2.2 Fundamentals

Variables in DruL must have names that begin with a letter or underscore, and contain only letters, numbers, and underscores thereafter. Variables are dynamically typed and scoped, so to create one, you need only assign a value to it:

```
a = 350;
b = 300;
print(a + b);
```

This should print out the number "650", again on a line by itself.

Now, with some variables defined, we can proceed quickly through the rest of the features that you might guess are present from the above:

```
a = 350;
b = 300;

c = b - a;

d = a % b;

e = 60 / d;

if (e > 1) {
        print("this is what you might expect to have happen");
} else {
        print("but this is what actually prints");
}
```

Why does the second line print, and not the first? DruL's types do not include floating-point numbers, so all arithmetic is done using integers, and non-integral results are truncated (as is done in C and other related languages) to their integer parts.

2.2.3 One more variable type: patterns

We will now introduce the first data type that distinguishes DruL from most other languages: the pattern. A pattern is a sequence of true/false values, telling the drummer (or MIDI sequencer) whether or not to play on a particular beat. They are created using the `pattern` function:

```
p1 = pattern("");          // empty pattern (length 0)
p2 = pattern("0");         // pattern with only one 'rest' in it.
p3 = pattern("1");         // pattern with only one 'note' in it.
p4 = pattern("100100100");
```

Each time a "1" appears in the string you pass to `pattern`, the resulting pattern carries the instruction to play on that beat; when a "0" appears, the pattern contains a rest. You may notice that we also have explanatory comments in this code: comments in DruL begin with "//" and continue to the end of the current line (there are no multi-line comments).

To see the contents of a pattern you have created, you can always just print it out:

```
p = pattern("100100100");
print(p);
```

2.3 Combining Patterns

Once you have a pattern or two, DruL gives you several ways to build new ones. Using the `concat` function, you can combine them end-to-end:

```
catenated = concat(
        p1,
        pattern("11110000"),
        pattern("00011")
);
```

Notice that we have broken up the arguments to `concat` onto multiple lines for ease of reading–since DruL is a free-form language, any amount of whitespace can appear any place that any whitespace is allowed.

Pattern objects also have a `repeat` method, which produces a new pattern containing all the beats and rests of the original, repeated however many times the method is

given as its argument. (In fact, you can give it an argument of 0 to return an empty pattern, though there are less obscure ways to do that.)

```
p_custom = concat(
        p2,
        p3.repeat(2),
        p4.repeat(3),
        p3.repeat(2),
        p4.repeat(4)
);
```

2.4 Manipulating Patterns

Patterns also have methods that allow you to produce new patterns that are not simply combinations of old ones laid end to end.

Using the `reverse` method, you can turn a pattern back to front; using the `slice` method, you can extract just the portion of it you want:

```
bassackwards = catenated.reverse();

p_new = bassackwards.slice(4,10);
```

The arguments to `slice` tell DruL which is the first beat of the pattern that you're interested in, and how many beats (including that one) you would like. So to the call above will produce a pattern 10 beats long, that starts with the 4th beat of `bassackwards`. If you ask for more beats than the pattern has, then `slice` will return a pattern that starts on the beat you specify and continues until the end of the original pattern.

Since all of these methods return patterns, and are methods of patterns, you can also stack your method calls into one statement:

```
p_new = catenated.reverse().slice(4,10);
```

But the most powerful mechanism for creating new and different patterns is the `map` function. This is how to take a pattern and create its complement: a pattern that has a rest everywhere that the original has a note, and vise-versa:

```
reversed = map (p_new) {
        if ($1.rest()) { return pattern("1"); }
        else           { return pattern("0"); }
};
```

The `map` function moves from beat to beat of the pattern it is passed, setting the variable `$1` to the point to the current beat of the first (and in this case, the only) pattern in its argument list. After each step, it stores the value that is returned, and in the end, it concatenates all these patterns together to form the new pattern that is created by this map.

You might be wondering, at this point, if it is legal to return a pattern that is longer or shorter than one beat. The answer, happily, is "yes!" To produce, for example, a pattern that has an extra rest inserted after every note, we could do this:

```
new_pattern = map(old_pattern) {
        if ( $1.note() ) { return pattern("10"); }
        return $1;
};
```

You may notice that we didn't bother to create a pattern for the second case: if we simply want to return a single-beat pattern with the same value (beat or rest) as the current beat of one of our input patterns, we can simply return that beat, and DruL will interpret it correctly.

Finally, we can pass more than one pattern to a mapper, and use variables `$2`, `$3` and so forth. This mapper takes two patterns as its arguments, and produces a new pattern that contains the portions of the first pattern that occur in parallel with notes (not rests) in the second pattern:

```
old_pattern = pattern("10101100");
filtered_pattern = map(old_pattern, pattern("1110011110000") ) {
        if ( $2.note() ) { return $1; }
};
print(old_pattern);
```

In this case, the printed value will be "101100" (if no return statement is found, `map` assumes that you meant to return an empty pattern). The same result could be achieved using a series of calls to `slice` and `concat`, but this is a much more flexible method.

If one of the patterns passed to `map` is longer than the others, `map` will continue until it reaches the end of the longest pattern–the beats of the patterns that have already ended are considered to be neither notes nor rests (and will return false if either of those methods is called).

We've mentioned beats as if they were objects once or twice, and in fact they are–you can't create them directly, but `map` does it for you. You've seen two of the methods you can call on beats (`note` and `rest`, but there are two more that make `map` even more powerful: `prev` and `next`. Calling `prev` with an integer argument returns the beat in the same pattern from that many beats ago in the pattern (and you can probably guess how `next` works). These beats may be from before the beginning or after the end of the pattern, in which case they behave just as described in the previous paragraph: both the `note` and `rest` methods will return `false`.

```
new_pattern = map (old_pattern) {
        if ( $1.note() && $1.prev(1).note() ) { return $1; }
        else { return pattern(""); }
};
```

2.5 Named mappers

In all of the examples so far, we have simply supplied the `map` function with a block of statements to run for this particular set of patterns. This block is what we refer to as an "anonymous mapper." In reality, of course, it is likely you would want perform the same type of transformation on more than one pattern (or set of patterns). To do this without annoying repetition of code, you can define a named mapper, then use the name instead of the code block. The named mapper can also have named parameters, which may be easier to keep track of than the shell-like variables used in anonymous mappers. This allows us to re-write the previous example in a somewhat more readable way:

```
mapper filter_map ( input_pattern, filter_pattern ) {
        if ( filter_pattern.note() ) { return input_pattern; }
};

filtered_pattern =
        map (old_pattern, pattern("1110011110000") ) filter_map;
```

2.6 Assembling clips

Now that we have a bunch of patterns, the next thing to do with them is assemble them into pieces of music, where each instrument has a (presumably) different pattern to play. Before we do that, however, we have to define what instruments we are using. This is done using the `instruments` function (which isn't actually a function at all, but we're going to ignore that for now–see section 3.7.1 in the Language Reference Manual if you want the grizzly details).

```
instruments("hihat","bassdrum","crash","snare");
```

If you want to use the default instruments, you can simply call `instruments` with no arguments, but you have to call it once (and exactly once) before you get to the next step: using the `clip` function to bring all of your patterns into one place.

```
my_first_clip = clip(
        "hihat" <- pattern("00100010"),
        "bassdrum" <- pattern("10001000"),
        "crash" <- pattern("10000000"),
        "snare" <- pattern("01110111")
);
```

Of course, this is a little verbose–if you're specifying the patterns in the order they appear in the instrument definition list, you can just pass the patterns you want as arguments, instead of using the fancy syntax above:

```
my_first_clip = clip(
        pattern("00100010"),
        pattern("10001000"),
        pattern("10000000"),
        pattern("01110111")
);
```

2.7 The Big Payoff

Now that we know how to assemble patterns into a song, all that's left is to see what our song looks like when we bring it back to the outside world. There are three easy ways to do this (other, of course, than a simple `print` call). First, you can call the `outputText` method to print your song to a text file:

```
instruments();  // default instruments: hihat, snare, kick-drum and cowbell

song = clip(
        pattern("00100010"),
        pattern("01110111"),
        pattern("10001000"),
        pattern("10000000")
);
song.outputText("my_song.txt");
```

If you have the `midge` program[1] installed, you can also convert it directly into a MIDI file you can play using many music players:

```
tempo = 120;
song.outputMidi("my_song.mid", 120);
```

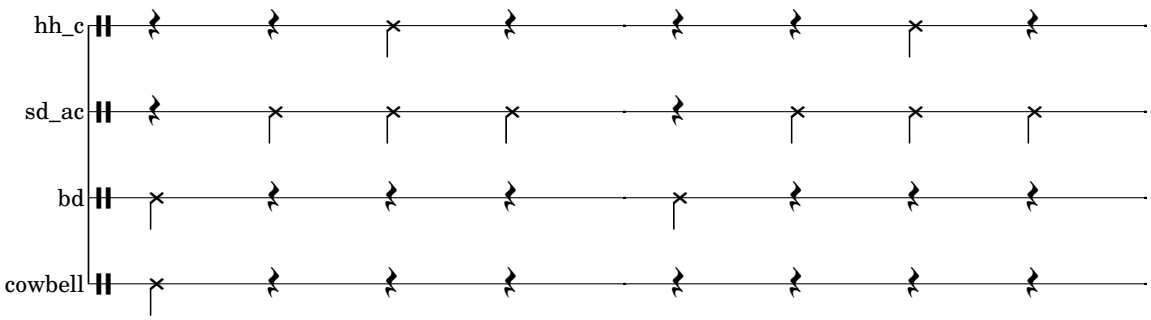
And finally, you can output to the format used by the typesetting package Lilypond[2] to produce beautifully typeset sheet music:

```
song.outputLilypond("my_song.ly", "Title of the Song");
```

Assuming you have Lilypond installed, this allows you to produce PDF sheet music that looks roughly like this:

[1] `http://www.undef.org.uk/code/midge/`
[2] `http://www.lilypond.org/`

Title of the Song



Congratulations! You're done with the tutorial–have fun with DruL!

Chapter 3

Language Reference Manual

3.1 Introduction

DruL is mainly an imperative programming language, however it borrows ideas (map and filter) from the functional paradigm. In addition to integers, DruL's main datatypes are pattern and clip. Instruments are defined as constants.

DruL programs do not contain any loops or user-defined functions. All pattern and clip creation and manipulation is done using the map construct described below.

3.2 Lexical Conventions

3.2.1 Comments

Comments in DruL start with the token "//" and continue until the end of the current line. DruL has no multi-line comment syntax.

3.2.2 Whitespace

Space, tab, end of line, and return are all considered the same and their only use is to seperate tokens.

3.2.3 Characters

DruL uses the ASCII character set.

3.2.4 Identifiers

An identifier consists of any uppercase or lowercase character or an underscore, followed by any sequence of uppercase or lowercase characters, underscores, and digits (0 through 9). The maximum length of an identifier is 64 characters.

In addition, within the context of a mapper, special variables $1 through $$n$ (where n is the number of patterns passed to the mapper) are defined as read-only aliases (see section 3.6.2 for more details on this feature).

All identifiers in a given scope, be they mapper names, variables, or built-in functions, belong to a single namespace.

3.2.5 Keywords

return	rand	clip	mapper
if	pattern	instrument	print
elseif	concat	map	
else	true	false	

3.3 Types

There are 3 basic types in DruL: **integer**s, **pattern**s, and **clip**s. In addition, 'string' constants may be used in DruL source code, but there is no variable type to which they can be directly assigned. Likewise, boolean expressions exist, but cannot be assigned to variables. Values in DruL are strongly typed, but the type of a variable is determined dynamically.

3.3.1 integer

All integers are base 10, and may optionally be preceded by a sign (`+ -`). Any sequence of digits (0 through 9) is valid. Leading 0s are ignored, so a sequence such as 0000123 is interpreted as 123. Integers are mutable.

rand is a function that returns a non-negative number. It either accepts a positive integer argument, in which case it returns a random number between 0 (inclusive) the argument (exclusive), or no argument in which case **rand** returns either 0 or 1.

```
r = rand();
s = rand(19);
```

3.3.2 pattern

A pattern is essentially an object that holds binary, discrete, time-series data. At each discrete-time step, which will henceforth be referred to as a beat, there is either a note or a rest. For the non-musically inclined, a note represents sound produced by the striking of a drum (or similar instrument) and a rest represents the absence of any such sound. Patterns are immutable. When a pattern is manipulated, the target pattern remains intact and a new copy is created. The length of a patten can be any non-negative integer.

3.3.3 beat

A beat is a lightweight object that cannot be created directly by the user: it exists only within a mapper (for more discussion of which, see section 3.6.2 below). It gives direct access to information about a single beat of a **pattern** object (including the beats surrounding it).

3.3.4 clip

An instrument is one of a pre-defined set of sounds (e.g. drum notes) that can occupy a single beat. A clip is a mapping of patterns to instruments. Clips are processed in sequence as the program runs to produce output which may be plain-text or a MIDI file. Clips are immutable.

3.3.5 string

A string constant begins with an ASCII double-quote character, continues with an arbitrary sequence of ASCII characters other than \, ″, and the ASCII newline character, and concludes with another ″ character. If a \ or ″ character is desired, it can be escaped using the \ character.

3.4 Statements

In the most common case, a statement consists of a single expression followed by a semicolon (";"). Importantly, unlike many languages with similar syntax, DruL does *not* consider a block to be equivalent to a statement. Instead, statements in DruL take one of the six forms below.

3.4.1 Expression Statements

The basic form of statement, as in most C-like languages, is the expression statement: *expression-statement*: *expression*;

The precedence table for operators in DruL is given here:

Operators	Notes	Associativity
.	Method call	left to right
− !	Unary minus and logical negation	right to left
∗ / %	Standard C meanings	left to right
+ −	Addition/subtraction	left to right
< <= > >=	Standard C meanings	left to right
! = ==	Standard C meanings	left to right
&&	Standard C meaning	left to right
||	Standard C meaning	left to right

The sections that follow use the model of the C Language Reference Manual to indicate the various types of expression. As in that example. the highest-precedence forms of expression are listed first. Since much of the material below is extremely straightforward, plain-English descriptions are supplemented by grammatical descriptions only when necessary.

Primary Expressions A primary expression consists of a constant (integer or string), an identifier, or a parenthesized expression.

Function, Method, and Mapper calls Arguments to functions, methods and mappers are evaluated in applicative order, left to right within a given list. (Arguments are also passed by value, not by reference.) Depending on the function or method in question, functions and methods may return values of any type, including boolean values (which cannot be assigned to variables); mappers, by their nature, always return patterns.

arglist: **(** *expression* **)** | **(** *expression* **,** *arglist* **)**

method-call : *identifier* **.** *identifier arglist*

function-call : *identifier arglist*

mapper-call: **map** *arglist mapper*

mapper : *identifier* | *block*

block : **{** *statement-list* **}**

statement-list: *statement* | *statement statement-list*

Unary operations The unary operations in DruL are arithmetic and logical negation (unary − and !). Since DruL is strictly typed, arithmetic negation can only be applied to integer values, and logical negation to boolean values.

Standard arithmetic operations Expressions may use the standard binary arithmetic operators (+, −, ∗ and /), with their standard precedence. It is required that both of the operands in such an expression be integer values.

Comparison operations As in most C-family languages (and as shown in the precedence table above), relational operators have precedence over equality tests. These operators return boolean values, which can be used in **if** statements but cannot be assigned to variables.

Relational tests may be used on integer values only; equality tests can be used on variables of any type, but in the case of patterns and clips, they will only report

whether the two variables being tested are aliases of the same object, not any deeper notion of equivalence.

Logical combination operations Here again we follow the conventions of C, and give && precedence over ||. These operators require their operands to be boolean values, and return boolean values.

3.4.2 Assignment Statements

Assignment in DruL is not a simple operator to be placed in the middle of an expression. Rather, it is a separate type of statement, which may appear anywhere another statement may appear.

assignment-statement: *identifier* = *expression-statement*

Assignment is polymorphic: the same syntax is used to assign variables to integers, patterns and clips. Furthermore, due to DruL's dynamic typing, a variable may be reassigned to a different type.

3.4.3 Selection Statements

Selection statements in DruL take the following form: the string **if**, followed by an expression that returns a boolean result, enclosed in parentheses, followed by an open-brace ("{"), one or more statements, and a close-brace ("}"). This may optionally be followed by one or more **elseif**s, which are also followed by parentheses and a block of statements, and one (optional) **else**, which omits the test expression but is also followed by a block of statements.

selection-statement : **if (** *boolean-expression* **)** *block if-tail*

if-tail : ϵ | *if-middle* | *if-middle* **else { ** *statement-list* **}**

if-middle : ϵ | **elseif (** *boolean-expression* **) { ** *statement-list* **}** *if-middle*

3.4.4 Mapper Definition Statements

A mapper definition consists of the word **Mapper**, followed by an identifier, followed by a parenthesized list of comma-separated identifiers, followed by a block.

mapper-definition : **mapper** *identifier namelist block*

namelist: (*identifier*) | (*identifier* , *namelist*)

3.4.5 Return statements

A return statement can only appear inside the statement block of a named or anonymous mapper:

return : **return** *expression*

If this statement is reached, the value of *expression* will be the output of this iteration of the mapper block. Accordingly, this expression must evaluate to either a pattern or a beat value.

3.4.6 Instrument definition

This is a special statement that closely resembles a function call:

instrument-definition: **instruments** (*arglist*)

The arguments to this pseudo-function must all evaluate to strings. See section 3.7.1 for a detailed discussion.

3.5 Blocks, namespace and scoping

3.5.1 Blocks

DruL has a limited block structure: only in the context of an **if**/**elseif**/**else** sequence or a Mapper Definition statement is a new block needed or allowed. In these cases, curly braces ("{}") are used to delimit the statement-sequence that falls within the block, and they must contain one or more statements.

Mapper definitions define a new closed scope (one from within which externally defined variables are not visible); **if** blocks do not define a new scope, so all variables used within them are visible to the enclosing block, and vise-versa.

3.5.2 Namespace

DruL has one namespace shared by variables, built-in functions and mapper names. Additionally, each type has an associated namespace for methods. Technically speaking, mappers are values like any other, and their names can be re-used, but this is strongly discouraged.

3.5.3 Scoping

Variables in DruL are dynamically scoped. DruL has one top level scope, and one scope for each mapper that the program enters (named or anonymous). Mappers may call each other (or themselves) recursively, and may be defined within other mappers, so a hierarchy of substantial depth can (in principle) be achieved. Within each scope, a program has read-only access to all variables and mapper names defined in the scopes above it in this hierarchy: attempts to assign to a variable from an outer scope will produce a new variable in the inner scope, which masks the original variable.

3.6 Patterns and pattern operations

3.6.1 Patterns

A pattern is a sequence of beats. Each beat can be a note or a rest. To define a pattern, DruL uses '0' for rests and '1' for notes. A pattern can be created in the following way:

```
p1 = pattern("101010");
```

which represents the sequence note, rest, note, rest, note, rest. Its length is 6.

There are built-in functions and methods on patterns included in DruL.

Patterns can be concatenated to form new patterns. The **concat** function can take any positive number of pattern arguments. Patterns are concatenated from left to right.

```
pcat = concat(p1 pattern("111000") pattern("1"));
```

pcat will be equal to 1010101110001.

The **repeat** method is a shortcut to concatenate the same pattern many times:

```
pcat2 = concat(p1, p1, p1);
pcat3 = pattern("101010").repeat(3);
pcat4 = p1.repeat(3);
```

Note that *pcat2*, *pcat3*, and *pcat4* are all equivalent.

The **length** method gives the length of a pattern.

```
len = p1.length();
```

The value of *len* is 6.

The **slice** method gives you a subpattern from a pattern. It takes two arguments: first is index (starting at 1) and second is length of the desired subpattern. Requesting a subpattern out of range will raise an error. Example:

```
psub = pattern("101010").slice(2, 3);
```

psub is "010".

The **reverse** method returns the reverse of a pattern. It doesn't take any arguments.

```
preverse = pattern("111000").reverse();
```

preverse is "000111".

Finally, you can have an empty pattern of length 0:

```
p8 = pattern("");
```

3.6.2 Map

The **map** construct is used to create new patterns from existing ones. **map** performs an operation iteratively on a set of patterns. The beats in the patterns are iterated over from left to right. The output of a map is a new pattern. For example:

```
p9  = pattern("101");
p10 = map(p9)
{
    if ($1.note()) { return pattern("11"); }
    else           { return pattern("0");  }
};
```

p10 is "11011".

map takes a sequence of pattern arguments and followed by a mapper function. In the above example the mapper function is defined anonymously within curly braces.

Within a mapper function, the current beat of each pattern argument is aliased to the special mapper variables $1, $2, $3... and so on. This notation is mandatory in anonymous mapper functions such as the example above. If you use $N while there is fewer than N arguments, DruL will raise an error.

DruL uses the **beat** methods **note**, **rest** and **null** to check whether the current beat is a note, a rest, or null. *$1.note* returns **true** if there is a note on the current beat in the first pattern argument, and **false** otherwise.

One can use the **beat** method **asPattern** to convert as beat to a pattern. This way, one can then make use of functions and methods of patterns. For example:

```
p11 = map( pattern("1111") )
{
    return concat($1.asPattern(), pattern("0"));
};
```

p11 is "10101010".

DruL uses the **beat** methods **prev** and **next** to access the previous and following beats of the pattern to which a given beat belongs. These methods can be passed a single argument which specifies how far forward or back in the pattern to go. For example:

```
p12 = map( pattern("1101") )
{
    if ($1.note() && $1.next(1).note()) { return pattern("1"); }
    else                                { return pattern("0"); }
};
```

p12 is "1000".

next may return a NULL beat as it does when called in the last iteration of the above example. When used with a NULL beat, both the **note** and **rest** methods will return *false*.

If you call map on multiple patterns that are not of the same length, the shorter patterns will be padded with NULL beats.

By default, an empty pattern is returned for each iteration.

Each new pattern constructed by map begins as an empty string. As the pattern arugments are iterated over, the return values of the mapper function (which are also patterns) are concatenated onto the end of the new pattern.

Variables defined in a mapper function are garbage collected at the end of the map.

3.6.3 Mapper

Mapper functions may also be defined with a name, to be used elsewhere in the program.

For example, the above example could have been written in the following way:

```
mapper myMapper(p)
{
    if (p.note() && p.next(1).note()) { return pattern("1"); }
    else                              { return pattern("0"); }
};
p12 = map(pattern("1101")) myMapper;
```

Recall from section 3.4.4 that a Mapper definition includes a name for the mapper and a *namelist* of formal arguments. When a named mapper is used in a **map** call,

each pattern that is passed to the **map** is associated with the corresponding name in the *namelist* in the mapper's definition. Then, within the body of the mapper, the current beat of each pattern is aliased to that name, as well as to "$\$n$".

A mapper function must be defined before it is used.

3.7 Clips

3.7.1 Instruments

Before we define any clips, we must tell the compiler what instruments they will use. This can only be done once per program, and uses the instruments function. (Technically speaking, this is not a function but a special statement type that uses a function-like syntax. The distinction is largely academic, however.) This function can take a variable number of arguments. Each argument is the name of an instrument to be defined. In the example below, four instruments are defined:

```
instruments(‘‘hihat’’,‘‘bassdrum’’,‘‘crash’’,‘‘snare’’);
```

Instruments must be defined before any clips have been defined. This function can only be called once. Also, it cannot be called from inside a mapper.

3.7.2 Clips

A clip represents a collection of patterns to be played in parallel, where each pattern is played on a single instrument.

Once the instruments are defined, we can create a clip from our existing patterns, using an associative-array notation:

```
clip1 = clip
(
    "bassdrum" <- downbeats,
    "hihat"    <- alternate_beats
);
```

The same result can be achieved by simply listing the patterns for each instrument in the order they are defined using the **instruments** function:

```
clip2 = clip
(
    alternate_beats,
    downbeats
    // remaining instruments have an empty beat-pattern
);
```

The patterns passed into clips are passed by value, not by reference.

Clips also have a small selection of output methods, discussed in the section below.

3.8 Outputs

DruL has two kinds of outputs: any data structure can be printed to standard output for debugging purposes, and clips may be output into files as text or using some more complex representation, such as MIDI or Lilypond (for PDF conversion).

3.8.1 Standard output

The **print** statement displays any type to the standard output, including strings. For example:

```
print ("DruL");
print (pattern("01"));
```

The representation of a string is the string itself. The representation of a pattern is the string that would have been used to initialize the pattern. For example, if we have a pattern

```
p = pattern("01").repeat(2);
print(p);
```

The output is "0101";

The **print** function always include a platform-appropriate line ending.

3.8.2 Text file

Using the same format as is used by **print**, DruL can print a text representation of a clip to a file using the **outputText** method of the clip:

```
myClip.outputText("myfile.txt");
```

The file being written to is truncated if it exists, and created if it does not exist.

3.8.3 MIDI file

The method **outputMidi** works similarly, but in addition to the filename, it requires a tempo for the MIDI file to be produced (in beats per minute–this must be a positive integer).

```
myClip.outputMidi(‘‘myfile.mid’’,120);
```

The transformation from clip to MIDI may rely on external libraries like MIDGE[1]. There is no guarantee on which of the three existing MIDI formats is used. DruL tries to match its instrument definition with MIDI instruments definitions using the names. If no match can be found, DruL will use a default MIDI instrument (first one is cow bell).

3.8.4 Lilypond file

The clip method **outputLilypond** operates similarly to the above, but takes a title (to be printed at the top of the page of typeset music) as an optional second argument:

[1] `http://www.undef.org.uk/code/midge/`

```
myClip.outputLilypond(‘‘myfile.ly’’, ‘‘My New Drum Loop’’);
```

For best results, the resulting Lilypond file will need to be typeset using an external program (Lilypond, one presumes).

Chapter 4

Project Plan

4.1 Processes

Almost all planning and decision making was done as group. The team leader resolved only very few disagreements of less importance, on which spending the time to come to a consensus was not warranted. As explained in more detail in section 4, we made liberal use of paired programming and most coding was done in group sessions. Documentation, including the Reference Manual and this report, was mostly done individually; each team member soley responsible for specific sections. Our testing process made use of an automated test-suite. After each change to the code base, the regression tests were all run, making sure that the number of tests passed always increased. This was somewhat done in the spirit of test driven development, as we made many test cases for language features before they were implemented, and used the test cases as a "ToDo" list. Our development process was also in the spirit of Agile or Extreme Programming. We started with a minimal, yet functioning language, and incrementally added features to it, all the while maintaining a working system.

Our plan was to complete tasks in the following order, working on tasks in paralell where possible:

1. Design DruL
    - Specify syntax
    - Specify semantics

2. Write the LRM
3. Implement Basic Building Blocks
    - AST
    - Scanner
    - Parser
    - Test Suite Driver
    - Initial test cases
    - Basic Interpreter (with evaluate, execute, and built-in print for string literals)
4. Implement Generic Language Features
    - Integer arithmetic
    - Boolean operations
    - Assignment statments and symbol table
    - Selection statements
5. Implement Core DruL Language Features
    - Pattern creation and printing
    - Pattern built-in functions and methods
    - Map, mappers, and beats
    - Named mappers
    - Beat built-in methods
    - Instrument definition
    - Clip creation and printing
6. Implement Advanced DruL Language Features
    - Clip text output
    - Clip MIDI output
    - Clip Lilypond output
    - Clip built-ins concat and repeat (not implemented)
7. Implement Fit and Finish
    - Interpreter command-line arguments

- Detailed error messages
- Error message line numbers
- Trapped parse errors
- Static semantic checks (not implemented)

We initially used our own SVN server for source and documentation management. However, we soon moved to Google Code to make used of its issues list for keeping track of bugs and the ToDo's correpsonding to the above features.

4.2 Style Guide

Due to the fact that all of our team members started as novice OCaml programmers and OCaml's syntax is unlike any languages our team was already familiar with, we lacked much intuition regarding good OCaml coding style. However, over the course of our coding, the following coding practice emerged:

- Use hard tabs for indentation
- Specify pattern matched arguments and use "match with" syntax instead of the "function" shorthand for pattern matching
- Encapsulate each "match with" clause in parentheses
- Put each match case on a new line
- Indent each match case and put a tab after the "—" separator
- Begin results of a match case on the same line as the "-¿"
- Indent subsequent lines of the results of a match case
- Horizontally align similar lines of code using extra whitespace

We never reached a conensus on whether to make our OCaml identifiers CamelCase or underscore_separated.

4.3 Timeline

Task	**Date(s)**
Specify syntax	Oct 8
Specify semantics	Oct 15
Write the LRM	Oct 22
AST	Nov 2
Scanner	Nov 2
Parser	Nov 3
Test Suite Driver	Nov 8
Initial test cases	Nov 8
Basic Interpreter	Nov 12
Integer arithmetic	Nov 19
Boolean operations	Nov 19
Assignment statments and symbol table	Nov 19
Selection statements	Nov 19
Pattern creation and printing	Nov 19
Pattern built-in functions and methods	Nov 26
Map, mappers, and beats	Nov 26
Named mappers	Dec 03
Beat built-in methods	Dec 03
DruL GCD	Dec 10
Instrument definition	Dec 10
Clip creation and printing	Dec 10
Clip text output	Dec 15
Clip MIDI output	Dec 16-17
Interpreter command-line argument	Dec 16
Detailed error messages	Dec 16
Error message line numbers	Dec 16
Clip Lilypond output	Dec 17
Trapped parse errors	Dec 17
DruL song	Dec 17
Presentation slides	Dec 17-18
Report	Dec 18-19

4.4 Roles and Responsibilities

Each team member volunteered for the completion of tasks. Tasks were not divied up amongst team members in advance. Rather, after team members finished their tasks they simply discussed what they should (or would like to) work on next with the rest of the team. Many tasks (especially the more difficult ones) were tackled in pairs. We found having an extra pair of eyes examining code and documentation as it was being written (i.e. paired programming) drastically cut down on the number of initial bugs and the amount of refactoring done later.

Below is a general description of the major tasks completed by each team member. Paired efforts are noted in parentheses. Note, that most work was with the group all in one room at one common table. This allowed and individual or pair to ask for help or advice from the rest of the team. This is done consistently and resulted with all members of the team being at least somewhat familiar with the impementation of almost all parts of the system.

- **Rob:**
    1. As team leader, resolved minor conflicts
    2. Proposed drumming language idea
    3. Setup intial SVN repository
    4. Wrote introduction for all documents
    5. Assisted with language design
    6. Wrote the Pattern, Map, and Mapper sections of the LRM
    7. Coded the AST (with Ben)
    8. Assisted with coding the parser
    9. Coded the initial "helloworld" interpreter (with Waseem)
    10. Coded pattern construct (with Ben)
    11. Coded DruL's built-in functions and methods and added corresponding test cases (with Waseem)
    12. Coded instrument and clip constructs (with Theirry)
    13. Performed built-ins code refactoring (with Ben)
    14. Performed code cleanup
    15. Refactored text output

16. Coded MIDI output
17. Wrote an example drum song using DruL
18. Edited all the presentation slides
19. Wrote the Project Plan section of the report

- **Ben:**

1. Decided on map/mapper idea
2. Assisted with writing proposal
3. Assisted with language design
4. Wrote the Types, Statments, Blocks and Scoping sections of the LRM
5. Coded the AST (with Rob)
6. Assisted with coding the parser
7. Coded the parse-tree dumper
8. Coded selecton and assignment statements
9. Coded pattern construct (with Rob)
10. Coded map and mapper (with Theirry)
11. Coded GCD implementation using DruL
12. Refactored built-ins (with Rob)
13. Refactored interpreter into smaller files
14. Refactored scanner, parser, and interpreter to inculde line numbers in error messages with (with Thierry)
15. Added tests for error messages
16. Coded Lilypond output
17. Refactored output code
18. Fixed corner-case symbol-table bugs
19. Wrote parts 1 and 2 of the presentation (except for the mapper animation)
20. Updated the LRM for the report (with Waseem). Wrote the tutorial section of the report (with Waseem)

- **Thierry:**

1. Assisted with writing proposal
2. Assisted with language design
3. Wrote the Instrument, Clip, and Output sections of the LRM

4. Setup replacement Google Code repository
5. Coded the test-suite driver
6. Manualy created initial suite of test input and corresponding output files
7. Assisted with coding the parser
8. Coded map and mapper (with Ben)
9. Coded named mapper
10. Coded instruments and clip constructs (with Rob)
11. Refactored scanner, parser, and interpreter to include line numbers in error messages (with Ben)
12. Added tests for error messages
13. Fixed corner-case symbol-table bugs
14. Wrote part 4 of the presentation
15. Made the mapper animation for the presentation
16. Wrote the testing section of the report

- **Waseem:**

1. Assisted with writing proposal
2. Assisted with language design
3. Wrote the Example Code section of the LRM
4. Coded the scanner
5. Assisted with coding the parser
6. Coded the initial "helloworld" interpreter (with Rob)
7. Coded DruL's built-in functions and methods and added corresponding test cases (with Rob)
8. Coded text file output
9. Wrote part 3 of the presentation
10. Updated the LRM for the report (with Ben)
11. Wrote the tutorial section of the report (with Ben)
12. Added the Divide by zero catch inside DruL.

4.5 Tools and Languages

All of our source code was written using OCaml with the exception of the special syntaxes used by ocamllex and ocamlyacc, and the test-suite driver which was written in Python.

4.5.1 Tools

- **Lexer:** We used ocamllex to compile our ocamllex code into an Ocaml lexer/-tokenizer, which given DruL source code, produces a token stream.
- **Parser:** We used ocamlyacc to compile our ocamlyacc code into an Ocaml parser, which given a token stream, produces a DruL abstract-syntax-tree.
- **MIDI Output:** MIDI (Musical Instrument Digital Interface) is a binary music protocal and file format that contains "event messags" for an audio device (e.g. a sound card or synthesizer). MIDI files are playable on many common multimedia players (e.g. Quicktime). DruL does not generate MIDI files directly. Rather it uses midge, which is yet another music composition language that compiles to MIDI. The language is not entirely different from DruL, however it allows for other instruments than drums and thus also has different note pitches and durations. However, midge doesn't not have contructs for algorithmic compositions comparable to the power of DruL's. In short, when DruL's outputMIDI method is called on a clip, the DruL interpreter produces midge code which is then piped to midge to produce the desired MIDI output file.
- **Lilypond:** Lilypond is a typesetting lanuage. DruL can produce Lilypond files. These output files can then be compiled into typeset PDF's (of sheetmusic) using Lilypond. DruL does not automate this however.

4.5.2 Code Editors

No one on our used the same code editor. None of us used an IDE. Team members used the following editors for all of their code (OCaml, latex, etc.):

- **Rob:** jEdit

- **Ben:** BBEdit
- **Thierry:** emacs
- **Waseem:** gedit

4.5.3 Documentation

All documentation was produced using LaTeX with the exception of the presentation, which was made using Microsoft PowerPoint.

4.5.4 Version Control

We used Google Code and Subversion (SVN) for our source version control and issue tracking.

4.6 Project Log

See the Appendix B.

Chapter 5

Architectural Design

5.1 Architecture Diagram

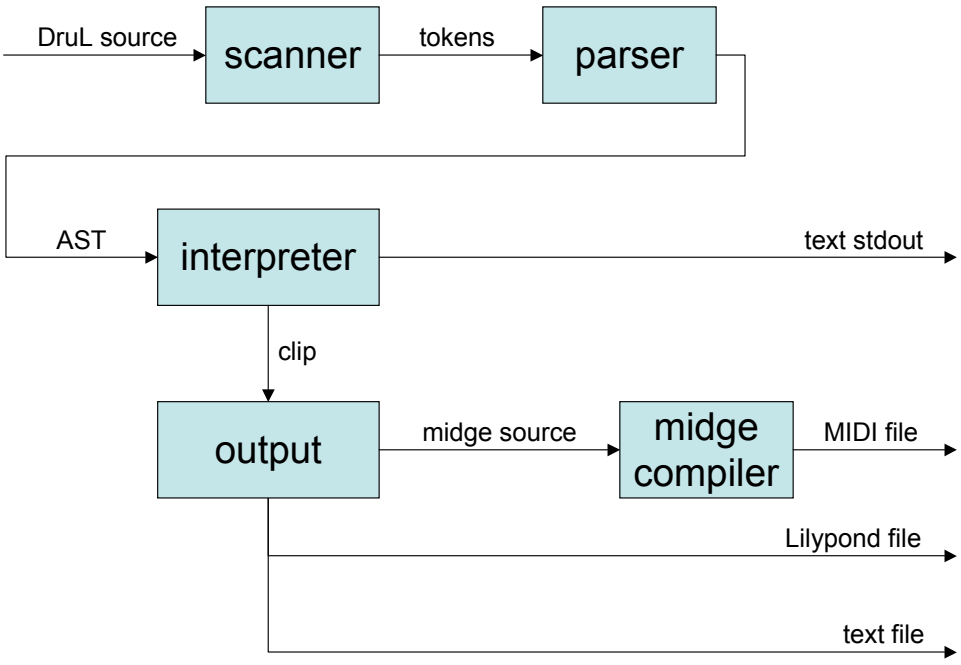


Figure 5.1: Arrows heads on edges show direction of data-flow.

5.2 Component Interfaces

The DruL interpreter is (as shown above) architecturally very simple.

1. The parser accepts a list of tokens from the scanner and builds a list of DruL statements (structured according to DruL's AST interface) to pass to the main unit of the interpreter. These components rely heavily on OCaml's Lexing and Parsing libraries.
2. The interpreter is monolithic: with the exception of the output module, all of its major sub-components are built into one set of mutually recursive functions (in the file `drul_main.ml`–see appendix C.1.2 on page 94). This monolith takes the statement list produced by the parser, evaluates it step by step (performing semantic checks on each statement only when program flow arrives at it), and passes the resulting structures to the output library when appropriate.
3. The output library is implemented as a set of simple utility functions: each takes a single DruL data structure and a small amount of extra data (the exact breakdown is unfortunately not well standardized, and varies from output type to output type), and returns a string formatted for the appropriate output style.

The monolithic design of the interpreter is necessitated by the single-pass approach taken to interpretation and by the dynamic typing of DruL variables: a re-implementation that included compilation and checking passes could also maintain cleaner separation of concerns.

5.3 Component Implemented By

File	**Author(s)**
drul_scanner.mll	Waseem
drul_parser.mly	all
drul_ast.mli	Ben and Rob
drul_interpreter.ml	all
drul_main.ml	all
drul_types.ml	all
drul_helpers.ml	all
drul_output.ml	all

Chapter 6

Test Suite

In the section we present the test suite we built and used for the DruL project. We start in Section 6.1 by showing the basic idea and limits for our testing program. In Section 6.2 we give details about the implementation. Finally, we give samples tests in Section 6.3 and explain what they test.

6.1 Overview

We built two different testing functions in order to debug DruL and help is maintainability: **LaunchTestParser** and **LaunchTest**. There usage is very similar.

LaunchTestParser's goal is to make sure every meaningful DruL code passes throught the scanner and parser without errors. We do not make sure that malformed DruL code is intercepted. The program passes a set of DruL code samples to the interpreter, and report whether a message error was produced. This sort of testing was very useful at the beginning of the project, but was later replaced by the more general **LaunchTest**.

LaunchTest takes a set of DruL code samples, pass them to the interpreter, and compares the output with some predefined output. Therefore, we can test both cases that fail (by catching the error message) or that correctly pass (by printing to the standard output).

6.2 Implementation

We implemented the two above testing programs in Python. This scripting language allows for rapid development and has an excellent packages for handling files. A test file has to have a certain extension (*.drultest*) and so does the desired output (*.drultestout*). The core of the testing programs, aside from finding the test files and passing them to the interpreter, is a simple "diff" function. This "diff" tells us if every line of two files are exactly the same or not. Everything is recorded in a LOG, whose name encodes the date and time of the test.

6.3 Sample tests

We present some typical tests for both the parser and the interpreter. In the second case, we also give the desired output.

6.3.1 Tests for DruL Parser

/TestSuite/ParserTests/logicalORAND.drultest

```
a = 1;
b = 2;
(false || true && false);
(true && false || true);
(a || b && 3 || false && true);
(true || false) && ((false && true || true) || true);
```

/TestSuite/ParserTests/print.drultest

```
print ("1");
print(        "allo");
print ("yo!3748473222937'1-232-/._(*&^%$#@");
print(pattern(""));
print( pattern ("010111001"));
a = pattern("11110");
print (a);
b = 3;
print( b );
c = clip(a);
print (c);
```

6.3.2 Tests for DruL

/TestSuite/Tests/pattern12.drultest

```
p11 = map(pattern("1111"))
{
 if ($1.note() && $1.next(1).note() && $1.next(2).note() ) { return pattern("1")
; }
 else { return pattern("0"); }
};
print(p11); // should return 1100
```

/TestSuite/Tests/pattern12.drultestout

```
1100
```

/TestSuite/Tests/clip2.drultest

```
instruments();
print(
        clip(
                pattern("1010")
        )
);
```

TestSuite/Tests/clip2.drultestout

```
[
        hh_c:   1010
        sd_ac:
        bd:
        cowbell:
]
```

/TestSuite/Tests/assign5.drultest

```
p = pattern("10");
mapper pattern (p) {}
print("bad");
```

/TestSuite/Tests/assign5.drultestout

```
Illegal assignment attempted on line 2: can't use keyword 'pattern' as a mapper name
```

6.4 Conclusion

The tests were designed by every team member, usually following the addition of a feature to DruL interpreter. We tried to keep the tests small and specific in order to better spot bugs. However, we also believe that "the more the better", thus we cannot say that the test were wisely chosen. Fortunately, there a smart-ass inside of everyone, and we do believe we tested most of the possible flaws.

Our test suite (programs and test files) adds up to 115 cases and about 1100 lines, almost as much as DruL itself. However, we felt it was time well spent for two major reasons:

- We did find bugs early in the coding process thanks to the test suite. One particular example is the precedance for member functions that we had forgotten.
- A complete test suite seems the only way to allow multiple programmers to modify a file without breaking code written by someone else

Thus we believe that a complete test suite is an essential part of a compiler's project and should be started before the actual language compiler.

Chapter 7

Lessons Learned

7.1 Introduction

In this chapter each team member tells about some lessons he learned from the project, and what he would do differently if we had start all over again.

7.2 Rob (team leader)

Coding standards are important, especially when using a new language that's unlike anything the team members have seen before. Unfortunately, this is when stanards are least likely to be used because no one knows of any relevant standards. Our team attemped to fit the square peg that is OCaml into the round holes that are the C and Java coding standards. This didn't work very well. In hindsight, we should have spent some time reading about suggested coding conventions for OCaml and researched how to organize a non-trivial OCaml code-base. We spent a long time agonizing over the monolithic spaghetti code that was our intereter before we finally got our heads straight and refactored it. However, we never reached a conensus on the proper way to format (e.g. indent) OCaml. I still find our code very hard to read. Also, domain specific conventions (assuming they exist) for writing a translator would have been useful. For exmaple, it got very confusing trying to keep track which of "int", "Int", and "CInt" were an OCaml type, a DruL type, or a DruL AST type. I had to look back to our type definitions almost everytime. In retrospect, prefixes such as "ast_int" and "drul_int" might have been less confusing.

7.3 Ben

I was surprised and impressed with how effective pair programming turned out to be. Leaving aside the technical issues, having second check on "the obvious way to do things" prevented me from getting into several potentially painful situations, when there was a much simpler solution available (this is especially relevant when working with a new language, of course).

Despite the amount of work done with pair programming, we still ran into some forms of communication trouble. In retrospect, a little more discussion up front about standards for code format and design (and for version control use) would have been helpful, at least in theory (it's hard to have a coding standard for a language that you don't actually know). Our error messages ended up somewhat inconsistent, and our log messages were sometimes uninformative (especially at first): better up-front coordination could have prevented those problems.

It is tempting to say that a more careful up-front design would have been well-advised, since it would definitely have been helpful–but since we were creating something we didn't really know how to create, using a language none of us was tremendously familiar with, it is unclear that spending more time on up-front design would actually have been productive in this case. Smaller-scale design issues, on the other hand, would have benefitted from a bit more forethought: we ended up with several somewhat inconsistent APIs for related helper functions, which could easily have been avoided by a little up-front communication or earlier and more aggressive refactoring. We did refactor often to retrofit better design onto the code we had written (made simpler by the easy-to-run regression test suite), but more aggressive refactoring of minor concerns would probably have sped things up toward the end, and would certainly have left us with a more maintainable final product. The type-checking and type-inference features of OCaml make this form of refactoring much safer than it is in many languages, and we should have taken more advantage.

More importantly,from the moment that we had working code, we should have made more active use of Subversion's branching capabilities, to avoid worries about breaking the main source tree while working on major features. We ended up re-inventing branching at least once, and leaving the entire tree in a non-working state for a couple of evenings, which could readily have been avoided.

7.4 Thierry

One part of the code I especially worked on is the test suite, but I still was surprised to see how important it turned out to be. In a new project, I would either build a more powerful testing program, or spend more time to find an appropriate package online. For instance, our current testing program does not have the ability to test an output file instead of the standard output. It would have become a problem if our language was designed for file operations.

Another lesson learned is the importance of helper functions designed early. At one point, every one of us had design is own method to lookup into the environment, and obviously we multiplied the number of bugs. For some functions, it is so obvious that they were going to be needed that we should have spent the time, as a team, to define them. Their documentation is also an important aspect when you work in a team of more than two programmers.

Following that idea, we probably did not use enough the "issue tracking" on Google code, the platform we used to host our project. Emails does not work as well...

7.5 Waseem

Most important lesson in while coding in OCaml is to modularize the code. Those match with clauses keep getting messier and also there is a lot of code repeatition while implementing similar functions or methods on the same language type object, e.g., pattern, clip, etc. Therefore, it is always good to have the helper functions, that can be used later on, in the code. This was my first group project of this level and believe it or not, my first time using version control:); Really makes your life easier. Of course, having those lexer and parser tools do most of the work for you is vry helpful. OCaml in itself is a rather powerful language. Syntax tends to get 'messy', however, its power is well to be noted. The code tends to be compact, especially when you factor out code that is repeated.

Working in pairs is definitely more helpful than working on one thing alone. In the former case you less likely tend to get stuck at a point, as compared to the later case.

Appendices

Appendix A

Number of Lines of Code

Main program and test suite.

40	drul_ast.mli
219	drul_helpers.ml
42	drul_interpreter.ml
471	drul_main.ml
87	drul_output.ml
119	drul_parser.mly
66	drul_printer.ml
106	drul_scanner.mll
59	drul_types.ml
61	Makefile
8	test.ml
5	treedump.ml
1283 total	

285	26 tests (parser)
422	79 test (drul)
399	2 'test' functions
1106 total	

Appendix B

Project Log (SVN Commit Log)

```
------------------------------------------------------------------------
r412 | waseemilahi | 2008-12-19 10:52:34 -0500 (Fri, 19 Dec 2008) | 1 line

Minor fix in the timeline
------------------------------------------------------------------------
r411 | waseemilahi | 2008-12-19 10:51:22 -0500 (Fri, 19 Dec 2008) | 1 line

Time line updated a bit; don't know whether to writer the date they were done
or the time period they were worked on
------------------------------------------------------------------------
r410 | benwarfield | 2008-12-19 05:15:45 -0500 (Fri, 19 Dec 2008) | 1 line

Made log a little less too wide.
------------------------------------------------------------------------
r409 | waseemilahi | 2008-12-19 05:11:04 -0500 (Fri, 19 Dec 2008) | 1 line

 Removed extra rand from table of keywords
------------------------------------------------------------------------
r408 | waseemilahi | 2008-12-19 05:01:25 -0500 (Fri, 19 Dec 2008) | 1 line

:)
------------------------------------------------------------------------
r407 | robstewart2 | 2008-12-19 04:56:25 -0500 (Fri, 19 Dec 2008) | 1 line

bunch of updates to report
------------------------------------------------------------------------
r406 | benwarfield | 2008-12-19 04:48:44 -0500 (Fri, 19 Dec 2008) | 1 line

Added code listings to appendices.  Some are kind of wide.
------------------------------------------------------------------------
```

r405 | robstewart2 | 2008-12-19 04:39:57 -0500 (Fri, 19 Dec 2008) | 1 line

put tutoiral section back in

---

r404 | benwarfield | 2008-12-19 04:38:22 -0500 (Fri, 19 Dec 2008) | 1 line

Tweaked colored-code sections.

---

r403 | benwarfield | 2008-12-19 04:25:29 -0500 (Fri, 19 Dec 2008) | 1 line

Stripped useless comments.

---

r402 | benwarfield | 2008-12-19 04:25:09 -0500 (Fri, 19 Dec 2008) | 1 line

Made the end of this file a little less... wide.

---

r401 | benwarfield | 2008-12-19 04:24:29 -0500 (Fri, 19 Dec 2008) | 1 line

Test for divide by zero.

---

r400 | benwarfield | 2008-12-19 04:21:05 -0500 (Fri, 19 Dec 2008) | 1 line

Added isnull method to beat.

---

r399 | waseemilahi | 2008-12-19 04:20:31 -0500 (Fri, 19 Dec 2008) | 1 line

 division by zero caught inside drul

---

r398 | waseemilahi | 2008-12-19 04:01:41 -0500 (Fri, 19 Dec 2008) | 1 line

 rand and reverse added in LRM

---

r397 | waseemilahi | 2008-12-19 03:59:10 -0500 (Fri, 19 Dec 2008) | 1 line

Lessons added

---

r396 | benwarfield | 2008-12-19 03:24:25 -0500 (Fri, 19 Dec 2008) | 1 line

Promoted a bunch of deserving subsections.

---

r395 | benwarfield | 2008-12-19 03:13:41 -0500 (Fri, 19 Dec 2008) | 1 line

Dumbed quotation marks in code sections back down.

---

r394 | benwarfield | 2008-12-19 03:09:55 -0500 (Fri, 19 Dec 2008) | 1 line

That should be \ref not \label...

---

r393 | benwarfield | 2008-12-19 03:00:53 -0500 (Fri, 19 Dec 2008) | 2 lines

```
Changed all the verbatims to lstlistings.  Also fixed the everything-is-red
problem.  Two things may not be actually connected.
------------------------------------------------------------------------
r392 | benwarfield | 2008-12-19 02:36:44 -0500 (Fri, 19 Dec 2008) | 2 lines

Added Tutorial section, and tweaked one footnote in the RefManual out of
general puckishness.
------------------------------------------------------------------------
r391 | robstewart2 | 2008-12-19 02:35:51 -0500 (Fri, 19 Dec 2008) | 1 line

changed intro to proposal
------------------------------------------------------------------------
r390 | robstewart2 | 2008-12-19 01:50:54 -0500 (Fri, 19 Dec 2008) | 1 line

added SvnLog.txt
------------------------------------------------------------------------
r389 | thierrybm@hotmail.com | 2008-12-19 01:40:24 -0500 (Fri, 19 Dec 2008) | 1 line

two minor typos fixed
------------------------------------------------------------------------
r388 | robstewart2 | 2008-12-19 01:38:29 -0500 (Fri, 19 Dec 2008) | 1 line

added an overview of drul to intro
------------------------------------------------------------------------
r387 | robstewart2 | 2008-12-19 01:22:16 -0500 (Fri, 19 Dec 2008) | 1 line

cleaned up ProjectPlan.tex
------------------------------------------------------------------------
r386 | robstewart2 | 2008-12-19 00:38:45 -0500 (Fri, 19 Dec 2008) | 1 line

fixed architecture table. added project plan to report.tex
------------------------------------------------------------------------
r385 | benwarfield | 2008-12-19 00:37:24 -0500 (Fri, 19 Dec 2008) | 1 line

Fixed up front page a bunch.
------------------------------------------------------------------------
r384 | robstewart2 | 2008-12-19 00:24:35 -0500 (Fri, 19 Dec 2008) | 1 line

attempting to fixed latex errors because of _ in Architecture.tex
------------------------------------------------------------------------
r383 | robstewart2 | 2008-12-19 00:20:50 -0500 (Fri, 19 Dec 2008) | 1 line

added arch diagram pdf.  cleaned up project plan in report
------------------------------------------------------------------------
r382 | robstewart2 | 2008-12-19 00:03:37 -0500 (Fri, 19 Dec 2008) | 1 line

added an architecture diagram
------------------------------------------------------------------------
```

r381 | robstewart2 | 2008-12-18 23:07:23 -0500 (Thu, 18 Dec 2008) | 1 line

mostly finished project plan

---

r380 | benwarfield | 2008-12-18 22:48:44 -0500 (Thu, 18 Dec 2008) | 1 line

Whoops.

---

r379 | benwarfield | 2008-12-18 21:35:21 -0500 (Thu, 18 Dec 2008) | 1 line

Adjustments to formatting (still unable to get the figure onto the cover page, though).

---

r378 | benwarfield | 2008-12-18 21:31:35 -0500 (Thu, 18 Dec 2008) | 1 line

Minor tweak to outputMidi paragraph.

---

r377 | benwarfield | 2008-12-18 21:29:24 -0500 (Thu, 18 Dec 2008) | 1 line

Fixes to code in (remaining) examples.

---

r376 | benwarfield | 2008-12-18 21:17:45 -0500 (Thu, 18 Dec 2008) | 1 line

Fixed a couple of examples, and updated output section.

---

r375 | benwarfield | 2008-12-18 20:53:05 -0500 (Thu, 18 Dec 2008) | 1 line

Typo in the first paragraph. Whoops!

---

r374 | benwarfield | 2008-12-18 20:49:04 -0500 (Thu, 18 Dec 2008) | 2 lines

Refmanual errata: return, instrument definition fixes; scoping explained,
expressions and string definitions cleaned up.

---

r373 | thierrybm@hotmail.com | 2008-12-18 20:44:08 -0500 (Thu, 18 Dec 2008) | 1 line

minor

---

r372 | thierrybm@hotmail.com | 2008-12-18 20:42:17 -0500 (Thu, 18 Dec 2008) | 1 line

architectural design section started

---

r371 | thierrybm@hotmail.com | 2008-12-18 20:40:03 -0500 (Thu, 18 Dec 2008) | 1 line

clip and instruments in RefMan seems OK

---

r370 | thierrybm@hotmail.com | 2008-12-18 20:31:59 -0500 (Thu, 18 Dec 2008) | 1 line

test sutie chapter updated

---

r369 | thierrybm@hotmail.com | 2008-12-18 20:26:24 -0500 (Thu, 18 Dec 2008) | 1 line

minor

---

r368 | thierrybm@hotmail.com | 2008-12-18 20:25:41 -0500 (Thu, 18 Dec 2008) | 1 line

tbm lessons learned is done

---

r367 | thierrybm@hotmail.com | 2008-12-18 20:13:19 -0500 (Thu, 18 Dec 2008) | 1 line

report has table of content

---

r366 | thierrybm@hotmail.com | 2008-12-18 20:08:43 -0500 (Thu, 18 Dec 2008) | 1 line

general layout of lessons learned chapter

---

r365 | thierrybm@hotmail.com | 2008-12-18 20:05:17 -0500 (Thu, 18 Dec 2008) | 1 line

test suite section kinda done... need approval by someone else

---

r364 | robstewart2 | 2008-12-18 19:49:17 -0500 (Thu, 18 Dec 2008) | 1 line

added empty Project Plan

---

r363 | robstewart2 | 2008-12-18 19:48:28 -0500 (Thu, 18 Dec 2008) | 1 line

changed intro section to chapter

---

r362 | benwarfield | 2008-12-18 19:47:56 -0500 (Thu, 18 Dec 2008) | 1 line

Use "report" format, which includes chapters; use chapters.

---

r361 | robstewart2 | 2008-12-18 19:45:27 -0500 (Thu, 18 Dec 2008) | 1 line

added intro to report

---

r360 | benwarfield | 2008-12-18 19:43:48 -0500 (Thu, 18 Dec 2008) | 1 line

Fixed cover page.

---

r359 | thierrybm@hotmail.com | 2008-12-18 19:43:44 -0500 (Thu, 18 Dec 2008) | 1 line

getting longer

---

r358 | thierrybm@hotmail.com | 2008-12-18 19:30:58 -0500 (Thu, 18 Dec 2008) | 1 line

using colors

---

r357 | thierrybm@hotmail.com | 2008-12-18 19:20:36 -0500 (Thu, 18 Dec 2008) | 1 line

```
test suite added
------------------------------------------------------------------------
r356 | thierrybm@hotmail.com | 2008-12-18 19:19:21 -0500 (Thu, 18 Dec 2008) | 1 line

beginning of the report
------------------------------------------------------------------------
r355 | robstewart2 | 2008-12-18 16:31:48 -0500 (Thu, 18 Dec 2008) | 1 line

cleaned up presentation
------------------------------------------------------------------------
r354 | robstewart2 | 2008-12-18 15:55:37 -0500 (Thu, 18 Dec 2008) | 1 line

added thierry's slides to parts 1-2
------------------------------------------------------------------------
r353 | robstewart2 | 2008-12-18 15:49:33 -0500 (Thu, 18 Dec 2008) | 1 line

added waseems slides to parts 1-2
------------------------------------------------------------------------
r352 | thierrybm@hotmail.com | 2008-12-18 15:48:32 -0500 (Thu, 18 Dec 2008) | 1 line

lines of code added
------------------------------------------------------------------------
r351 | thierrybm@hotmail.com | 2008-12-18 15:31:54 -0500 (Thu, 18 Dec 2008) | 1 line

updated list of reserved keywords
------------------------------------------------------------------------
r350 | thierrybm@hotmail.com | 2008-12-18 15:29:51 -0500 (Thu, 18 Dec 2008) | 1 line

we catch bad mapper naming
------------------------------------------------------------------------
r349 | thierrybm@hotmail.com | 2008-12-18 15:29:43 -0500 (Thu, 18 Dec 2008) | 1 line

we catch bad mapper naming
------------------------------------------------------------------------
r348 | thierrybm@hotmail.com | 2008-12-18 15:23:08 -0500 (Thu, 18 Dec 2008) | 1 line

instr def in mappers solved
------------------------------------------------------------------------
r347 | thierrybm@hotmail.com | 2008-12-18 15:13:24 -0500 (Thu, 18 Dec 2008) | 1 line

can't assing instruments inside mappers
------------------------------------------------------------------------
r346 | thierrybm@hotmail.com | 2008-12-18 14:53:29 -0500 (Thu, 18 Dec 2008) | 1 line

important test for weird assignments of mapper that should fail
------------------------------------------------------------------------
r345 | benwarfield | 2008-12-18 14:25:10 -0500 (Thu, 18 Dec 2008) | 1 line
```

```
Changed keyword check message (and keyword check).
------------------------------------------------------------------------
r344 | benwarfield | 2008-12-18 14:23:01 -0500 (Thu, 18 Dec 2008) | 1 line

Changed keyword check message (and keyword check).
------------------------------------------------------------------------
r343 | benwarfield | 2008-12-18 13:49:48 -0500 (Thu, 18 Dec 2008) | 1 line

Trapped an un-trapped internal error (mapper/variable name collision).
------------------------------------------------------------------------
r342 | waseemilahi | 2008-12-18 10:15:55 -0500 (Thu, 18 Dec 2008) | 1 line

part3.ppt update
------------------------------------------------------------------------
r341 | waseemilahi | 2008-12-18 09:46:22 -0500 (Thu, 18 Dec 2008) | 1 line

 yeah! another update:)
------------------------------------------------------------------------
r340 | waseemilahi | 2008-12-18 09:38:20 -0500 (Thu, 18 Dec 2008) | 1 line

 part3.ppt update
------------------------------------------------------------------------
r339 | waseemilahi | 2008-12-18 09:04:39 -0500 (Thu, 18 Dec 2008) | 1 line

presentation update
------------------------------------------------------------------------
r338 | waseemilahi | 2008-12-18 08:49:29 -0500 (Thu, 18 Dec 2008) | 1 line

slides for part three (in progress)
------------------------------------------------------------------------
r337 | benwarfield | 2008-12-18 03:19:27 -0500 (Thu, 18 Dec 2008) | 1 line

Added a very overlong draft of slides for the first 4 minutes or so.
------------------------------------------------------------------------
r336 | benwarfield | 2008-12-18 02:05:38 -0500 (Thu, 18 Dec 2008) | 1 line

Added typesetting to example song, and added example of typesetting to presentation.
------------------------------------------------------------------------
r335 | benwarfield | 2008-12-18 02:05:11 -0500 (Thu, 18 Dec 2008) | 1 line

Updated errata.
------------------------------------------------------------------------
r334 | benwarfield | 2008-12-18 01:39:58 -0500 (Thu, 18 Dec 2008) | 1 line

Moved all non-trivial string-production into drul_output.ml
------------------------------------------------------------------------
r333 | benwarfield | 2008-12-18 01:31:13 -0500 (Thu, 18 Dec 2008) | 1 line

Updated svn:ignore on Parser.
```

```
------------------------------------------------------------------------
r332 | benwarfield | 2008-12-18 01:30:50 -0500 (Thu, 18 Dec 2008) | 1 line

Made LilyPond output happen.
------------------------------------------------------------------------
r331 | benwarfield | 2008-12-18 01:30:04 -0500 (Thu, 18 Dec 2008) | 1 line

Removed slightly spurious (misplaced) comment.
------------------------------------------------------------------------
r330 | benwarfield | 2008-12-18 01:10:45 -0500 (Thu, 18 Dec 2008) | 1 line

Refactored midge output, and improved argument-checking on both output methods.
------------------------------------------------------------------------
r329 | benwarfield | 2008-12-18 00:34:52 -0500 (Thu, 18 Dec 2008) | 1 line

Refactored clip printing/text output.
------------------------------------------------------------------------
r328 | benwarfield | 2008-12-17 23:36:15 -0500 (Wed, 17 Dec 2008) | 1 line

Upgraded an error message slightly.
------------------------------------------------------------------------
r327 | thierrybm@hotmail.com | 2008-12-17 23:02:34 -0500 (Wed, 17 Dec 2008) | 1 line

text improving
------------------------------------------------------------------------
r326 | waseemilahi | 2008-12-17 22:53:37 -0500 (Wed, 17 Dec 2008) | 1 line

message changed from interpret to drul
------------------------------------------------------------------------
r325 | thierrybm@hotmail.com | 2008-12-17 22:52:31 -0500 (Wed, 17 Dec 2008) | 1 line

text improving
------------------------------------------------------------------------
r324 | thierrybm@hotmail.com | 2008-12-17 22:46:37 -0500 (Wed, 17 Dec 2008) | 1 line

text for part 4 that goes along with the slides
------------------------------------------------------------------------
r323 | thierrybm@hotmail.com | 2008-12-17 22:44:45 -0500 (Wed, 17 Dec 2008) | 1 line

slides for the part 4, very simple, text from the outline put in 2 slides...
------------------------------------------------------------------------
r322 | waseemilahi | 2008-12-17 22:33:33 -0500 (Wed, 17 Dec 2008) | 1 line

 changed interpret to drul in test suite
------------------------------------------------------------------------
r321 | benwarfield | 2008-12-17 22:32:53 -0500 (Wed, 17 Dec 2008) | 1 line

Added line-numbers to parse errors.
------------------------------------------------------------------------
```

r320 | waseemilahi | 2008-12-17 22:23:40 -0500 (Wed, 17 Dec 2008) | 1 line

Makefile updated, now makes drul instead of interpret

---

r319 | waseemilahi | 2008-12-17 22:19:51 -0500 (Wed, 17 Dec 2008) | 1 line

test for reverse method

---

r318 | waseemilahi | 2008-12-17 22:13:10 -0500 (Wed, 17 Dec 2008) | 1 line

pattern.reverse() added (is it suppose to be a method or a function:) i totally forgot

---

r317 | benwarfield | 2008-12-17 19:24:05 -0500 (Wed, 17 Dec 2008) | 1 line

Outline for presentation added, in msft word format (for my sins).

---

r316 | robstewart2 | 2008-12-17 19:19:28 -0500 (Wed, 17 Dec 2008) | 1 line

added song.drul to Examples/ and cleaned up gcd.drul

---

r315 | thierrybm@hotmail.com | 2008-12-17 19:15:58 -0500 (Wed, 17 Dec 2008) | 1 line

better presentation of $

---

r314 | thierrybm@hotmail.com | 2008-12-17 19:15:43 -0500 (Wed, 17 Dec 2008) | 1 line

better presentation of $

---

r313 | thierrybm@hotmail.com | 2008-12-17 19:07:52 -0500 (Wed, 17 Dec 2008) | 1 line

presentation in ppt

---

r312 | thierrybm@hotmail.com | 2008-12-17 19:05:45 -0500 (Wed, 17 Dec 2008) | 1 line

values of added

---

r311 | thierrybm@hotmail.com | 2008-12-17 19:00:54 -0500 (Wed, 17 Dec 2008) | 1 line

now has curr prev and next written

---

r310 | thierrybm@hotmail.com | 2008-12-17 18:52:27 -0500 (Wed, 17 Dec 2008) | 1 line

litlle presentation of the mapper iterator, in openoffice presentation

---

r309 | thierrybm@hotmail.com | 2008-12-17 18:43:03 -0500 (Wed, 17 Dec 2008) | 1 line

folder for the presentation with Edwards

---

r308 | benwarfield | 2008-12-17 18:19:12 -0500 (Wed, 17 Dec 2008) | 1 line

```
It would help if I checked these in, too...
------------------------------------------------------------------------
r307 | benwarfield | 2008-12-17 18:18:43 -0500 (Wed, 17 Dec 2008) | 1 line

One final bugfix in illegal-return (and tweaked the message).
------------------------------------------------------------------------
r306 | robstewart2 | 2008-12-17 18:14:55 -0500 (Wed, 17 Dec 2008) | 1 line

added a test for beat.asPattern()
------------------------------------------------------------------------
r305 | benwarfield | 2008-12-17 18:12:36 -0500 (Wed, 17 Dec 2008) | 1 line

Fixed exception handling in one_mapper_step.
------------------------------------------------------------------------
r304 | robstewart2 | 2008-12-17 18:04:38 -0500 (Wed, 17 Dec 2008) | 1 line

added beat.asPattern()
------------------------------------------------------------------------
r303 | thierrybm@hotmail.com | 2008-12-17 17:33:50 -0500 (Wed, 17 Dec 2008) | 1 line

checks the return of .prev(1)
------------------------------------------------------------------------
r302 | thierrybm@hotmail.com | 2008-12-17 17:32:56 -0500 (Wed, 17 Dec 2008) | 1 line

checks the return of .next(1)
------------------------------------------------------------------------
r301 | benwarfield | 2008-12-17 17:26:47 -0500 (Wed, 17 Dec 2008) | 1 line

Added return-this-beat capability to mappers.
------------------------------------------------------------------------
r300 | benwarfield | 2008-12-17 16:50:12 -0500 (Wed, 17 Dec 2008) | 1 line

Forbade assignment of mappers.
------------------------------------------------------------------------
r299 | thierrybm@hotmail.com | 2008-12-17 16:43:20 -0500 (Wed, 17 Dec 2008) | 1 line

test updated with new default instruments
------------------------------------------------------------------------
r298 | thierrybm@hotmail.com | 2008-12-17 16:31:37 -0500 (Wed, 17 Dec 2008) | 1 line

new default instruments
------------------------------------------------------------------------
r297 | thierrybm@hotmail.com | 2008-12-17 16:29:49 -0500 (Wed, 17 Dec 2008) | 1 line

new output is clip.outputText
------------------------------------------------------------------------
r296 | robstewart2 | 2008-12-17 15:53:20 -0500 (Wed, 17 Dec 2008) | 1 line
```

```
fixed the conflict with waseems commented out code
------------------------------------------------------------------------
r295 | thierrybm@hotmail.com | 2008-12-17 15:28:44 -0500 (Wed, 17 Dec 2008) | 1 line
test on assigning to unknown instruments when creating a clip
------------------------------------------------------------------------
r294 | thierrybm@hotmail.com | 2008-12-17 15:26:38 -0500 (Wed, 17 Dec 2008) | 1 line
better error when creating a clip and assigning something to an unknwon instrument
------------------------------------------------------------------------
r293 | thierrybm@hotmail.com | 2008-12-17 15:20:46 -0500 (Wed, 17 Dec 2008) | 1 line
better comments before some functions
------------------------------------------------------------------------
r292 | thierrybm@hotmail.com | 2008-12-17 15:15:56 -0500 (Wed, 17 Dec 2008) | 1 line
better comments before some functions
------------------------------------------------------------------------
r291 | thierrybm@hotmail.com | 2008-12-17 15:05:13 -0500 (Wed, 17 Dec 2008) | 1 line
better error messages
------------------------------------------------------------------------
r290 | thierrybm@hotmail.com | 2008-12-17 14:57:36 -0500 (Wed, 17 Dec 2008) | 1 line
better error messages
------------------------------------------------------------------------
r289 | thierrybm@hotmail.com | 2008-12-17 14:46:46 -0500 (Wed, 17 Dec 2008) | 1 line
tests on bad assignment with line numbers
------------------------------------------------------------------------
r288 | thierrybm@hotmail.com | 2008-12-17 14:38:12 -0500 (Wed, 17 Dec 2008) | 1 line
better error messages with line numbers
------------------------------------------------------------------------
r287 | thierrybm@hotmail.com | 2008-12-17 14:32:21 -0500 (Wed, 17 Dec 2008) | 1 line
better line numbering, in get_key_from_env and other functions
------------------------------------------------------------------------
r286 | robstewart2 | 2008-12-17 14:25:43 -0500 (Wed, 17 Dec 2008) | 1 line
cleaned up code formatting in all files and cleaned up file output
------------------------------------------------------------------------
r285 | benwarfield | 2008-12-17 02:17:42 -0500 (Wed, 17 Dec 2008) | 1 line
Made error messages look like they were written by a human being, and updated related tests.
------------------------------------------------------------------------
r284 | benwarfield | 2008-12-16 22:10:18 -0500 (Tue, 16 Dec 2008) | 1 line
Made interpreter take a command-line argument as a filename for input if one is provided.
```

```
------------------------------------------------------------------------
r283 | benwarfield | 2008-12-16 18:59:44 -0500 (Tue, 16 Dec 2008) | 1 line

Added line numbers to output-related exceptions.
------------------------------------------------------------------------
r282 | benwarfield | 2008-12-16 18:48:04 -0500 (Tue, 16 Dec 2008) | 1 line

Removed spurious comments.
------------------------------------------------------------------------
r281 | benwarfield | 2008-12-16 18:37:22 -0500 (Tue, 16 Dec 2008) | 1 line

Got line numbers passed down to "clip" and to mapper-related helpers.
------------------------------------------------------------------------
r280 | benwarfield | 2008-12-16 18:33:23 -0500 (Tue, 16 Dec 2008) | 1 line

Fixes to fixes on exception-raising.
------------------------------------------------------------------------
r279 | thierrybm@hotmail.com | 2008-12-16 18:32:40 -0500 (Tue, 16 Dec 2008) | 1 line

make_clip now takes a line number
------------------------------------------------------------------------
r278 | thierrybm@hotmail.com | 2008-12-16 18:19:37 -0500 (Tue, 16 Dec 2008) | 1 line

errors fixed
------------------------------------------------------------------------
r277 | thierrybm@hotmail.com | 2008-12-16 18:12:59 -0500 (Tue, 16 Dec 2008) | 1 line

failures should not have line number or -1, others errors do
------------------------------------------------------------------------
r276 | benwarfield | 2008-12-16 18:12:46 -0500 (Tue, 16 Dec 2008) | 1 line

Added line numbers to a lot of exceptions.
------------------------------------------------------------------------
r275 | thierrybm@hotmail.com | 2008-12-16 18:10:36 -0500 (Tue, 16 Dec 2008) | 1 line

failures should now have line number or -1
------------------------------------------------------------------------
r274 | thierrybm@hotmail.com | 2008-12-16 18:02:32 -0500 (Tue, 16 Dec 2008) | 1 line

some errors updated with line number
------------------------------------------------------------------------
r273 | thierrybm@hotmail.com | 2008-12-16 18:00:17 -0500 (Tue, 16 Dec 2008) | 1 line

some errors updated with line number
------------------------------------------------------------------------
r272 | thierrybm@hotmail.com | 2008-12-16 17:54:42 -0500 (Tue, 16 Dec 2008) | 1 line

exceptions takes also an int
------------------------------------------------------------------------
```

```
r271 | benwarfield | 2008-12-16 17:48:59 -0500 (Tue, 16 Dec 2008) | 2 lines

Made all uses of expr into tagged_expr.  Things somehow
still all work.
------------------------------------------------------------------------
r270 | benwarfield | 2008-12-16 17:28:46 -0500 (Tue, 16 Dec 2008) | 1 line

Reduced indentation a bit in output function.
------------------------------------------------------------------------
r269 | benwarfield | 2008-12-16 17:25:17 -0500 (Tue, 16 Dec 2008) | 1 line

Added line-number tagging to scanner and parser and AST.
------------------------------------------------------------------------
r268 | waseemilahi | 2008-12-16 16:33:50 -0500 (Tue, 16 Dec 2008) | 1 line

 Corrected the file permission problem for output to file
------------------------------------------------------------------------
r267 | thierrybm@hotmail.com | 2008-12-16 16:32:08 -0500 (Tue, 16 Dec 2008) | 1 line

in parser, tokens take at least one int, the line number, this upload BREAKS EVERYTHING
but we're fixing it
------------------------------------------------------------------------
r266 | benwarfield | 2008-12-16 15:40:38 -0500 (Tue, 16 Dec 2008) | 1 line

Corrected header of drul_types.ml
------------------------------------------------------------------------
r265 | benwarfield | 2008-12-16 15:36:07 -0500 (Tue, 16 Dec 2008) | 1 line

Minor cleanup in interpreter.
------------------------------------------------------------------------
r264 | benwarfield | 2008-12-16 15:23:05 -0500 (Tue, 16 Dec 2008) | 1 line

Rearranged code into multiple files, for ease of maintenance.
------------------------------------------------------------------------
r263 | waseemilahi | 2008-12-15 10:36:20 -0500 (Mon, 15 Dec 2008) | 2 lines

Comments Added at places.

------------------------------------------------------------------------
r262 | thierrybm@hotmail.com | 2008-12-14 22:23:44 -0500 (Sun, 14 Dec 2008) | 1 line

better error when assigning a clip without defining instruments
------------------------------------------------------------------------
r261 | thierrybm@hotmail.com | 2008-12-14 21:52:55 -0500 (Sun, 14 Dec 2008) | 1 line

clip assignment solved
------------------------------------------------------------------------
r260 | thierrybm@hotmail.com | 2008-12-14 21:51:25 -0500 (Sun, 14 Dec 2008) | 1 line
```

```
partly solve the problem of assignments, but we can still assign to clip...
------------------------------------------------------------------------
r259 | thierrybm@hotmail.com | 2008-12-14 21:19:56 -0500 (Sun, 14 Dec 2008) | 1 line

assign to pattern, fails for the moment
------------------------------------------------------------------------
r258 | thierrybm@hotmail.com | 2008-12-14 20:36:45 -0500 (Sun, 14 Dec 2008) | 1 line

makes sure we can't assign anything to true or false
------------------------------------------------------------------------
r257 | thierrybm@hotmail.com | 2008-12-14 20:34:27 -0500 (Sun, 14 Dec 2008) | 1 line

test assignment to 'rand', fails for the moment
------------------------------------------------------------------------
r256 | thierrybm@hotmail.com | 2008-12-14 20:32:23 -0500 (Sun, 14 Dec 2008) | 1 line

test updated, no problem with instruments, can't assign it
------------------------------------------------------------------------
r255 | thierrybm@hotmail.com | 2008-12-14 20:28:04 -0500 (Sun, 14 Dec 2008) | 1 line

instruments assignment test, fails for the moment
------------------------------------------------------------------------
r254 | thierrybm@hotmail.com | 2008-12-14 20:25:22 -0500 (Sun, 14 Dec 2008) | 1 line

clip assignment test, fails for the moment
------------------------------------------------------------------------
r253 | waseemilahi | 2008-12-14 12:02:42 -0500 (Sun, 14 Dec 2008) | 1 line

Removed .txt check. File name can be anything the user wants it to be, as far as
we are concerned.
------------------------------------------------------------------------
r252 | waseemilahi | 2008-12-14 11:33:44 -0500 (Sun, 14 Dec 2008) | 1 line

Just added file name check. I think as far as Linux is concerned we do not need
file name checks. But it looks better for a text file to have .txt extension.
------------------------------------------------------------------------
r251 | waseemilahi | 2008-12-10 21:45:13 -0500 (Wed, 10 Dec 2008) | 1 line

 output.txtfile functions now output clips to the files, just like print
does on the stdout. (I will look into the issue of outputting clips in mapper.)
------------------------------------------------------------------------
r250 | waseemilahi | 2008-12-10 21:31:45 -0500 (Wed, 10 Dec 2008) | 1 line

output.txtfile_*** functions do what print does except it doesn't output clips yet.
The two extensions are append and truncate to choose what the user wants to do with
the already existing file.
------------------------------------------------------------------------
r249 | waseemilahi | 2008-12-10 19:08:43 -0500 (Wed, 10 Dec 2008) | 1 line
```

```
 output test updated
------------------------------------------------------------------------
r248 | waseemilahi | 2008-12-10 19:06:22 -0500 (Wed, 10 Dec 2008) | 1 line

output.txt_truncate and output,txtfile_apend do as their names suggest.
------------------------------------------------------------------------
r247 | waseemilahi | 2008-12-10 18:07:40 -0500 (Wed, 10 Dec 2008) | 1 line

 I think output.txtfile() needs to give a valid filename along with the string.
------------------------------------------------------------------------
r246 | waseemilahi | 2008-12-10 17:56:44 -0500 (Wed, 10 Dec 2008) | 1 line

 output.txtfile() outputs a string to the file with extension .txt, if the
file already exists, it truncates it and if it doesn't then it creates it.
------------------------------------------------------------------------
r245 | waseemilahi | 2008-12-10 17:33:11 -0500 (Wed, 10 Dec 2008) | 1 line

Need to flush the out_channel and close it.
------------------------------------------------------------------------
r244 | waseemilahi | 2008-12-10 16:53:15 -0500 (Wed, 10 Dec 2008) | 1 line

The check for file extension added (I don't know if we need it for Linux, but
windoes cares about extensions). output.txtfile(..) should only care about .txt files.
------------------------------------------------------------------------
r243 | benwarfield | 2008-12-10 16:47:09 -0500 (Wed, 10 Dec 2008) | 1 line

Made clips exist, and print.
------------------------------------------------------------------------
r242 | benwarfield | 2008-12-10 16:46:20 -0500 (Wed, 10 Dec 2008) | 1 line

Commented out useless extra prints.
------------------------------------------------------------------------
r241 | waseemilahi | 2008-12-10 16:18:20 -0500 (Wed, 10 Dec 2008) | 1 line

Added a new token OUTPUT. Changed ast, scanner and parser to accomodate for
output.txtfile format. Haven't yet finished with the output function yet.
For now it only creates/opens a file to write to it.
------------------------------------------------------------------------
r240 | benwarfield | 2008-12-10 15:42:49 -0500 (Wed, 10 Dec 2008) | 1 line

Squashed shift-reduce issues with left-arrow.
------------------------------------------------------------------------
r239 | thierrybm@hotmail.com | 2008-12-10 15:37:43 -0500 (Wed, 10 Dec 2008) | 1 line

we can create empty clips of given size
------------------------------------------------------------------------
r238 | benwarfield | 2008-12-10 15:32:00 -0500 (Wed, 10 Dec 2008) | 1 line

Refactored method calls to use eval_arg_list (and fixed a typo).
```

```
------------------------------------------------------------------------
r237 | benwarfield | 2008-12-10 15:25:25 -0500 (Wed, 10 Dec 2008) | 1 line

Refactored function calls to use eval_arg_list.
------------------------------------------------------------------------
r236 | robstewart2 | 2008-12-10 15:13:27 -0500 (Wed, 10 Dec 2008) | 1 line

added InstrAssign expressions back in
------------------------------------------------------------------------
r235 | thierrybm@hotmail.com | 2008-12-10 14:53:26 -0500 (Wed, 10 Dec 2008) | 1 line

minor
------------------------------------------------------------------------
r234 | robstewart2 | 2008-12-10 14:51:49 -0500 (Wed, 10 Dec 2008) | 1 line

added note to RefManual ERRATA about instruments
------------------------------------------------------------------------
r233 | thierrybm@hotmail.com | 2008-12-10 14:50:42 -0500 (Wed, 10 Dec 2008) | 1 line

now intruments() call the default instruments
------------------------------------------------------------------------
r232 | robstewart2 | 2008-12-10 14:27:06 -0500 (Wed, 10 Dec 2008) | 1 line

changed InstrAssign to InstrDef
------------------------------------------------------------------------
r231 | thierrybm@hotmail.com | 2008-12-10 13:50:17 -0500 (Wed, 10 Dec 2008) | 1 line

added Ben's gcd example to the test suite
------------------------------------------------------------------------
r230 | thierrybm@hotmail.com | 2008-12-09 17:45:34 -0500 (Tue, 09 Dec 2008) | 1 line

minor, comments added
------------------------------------------------------------------------
r229 | thierrybm@hotmail.com | 2008-12-09 17:41:02 -0500 (Tue, 09 Dec 2008) | 1 line

minor modif to instrument_pos functions, better exception catching
------------------------------------------------------------------------
r228 | thierrybm@hotmail.com | 2008-12-09 17:38:37 -0500 (Tue, 09 Dec 2008) | 1 line

function get_instrument_pos works, damn you ocaml syntax that made us lose an hour on this
------------------------------------------------------------------------
r227 | thierrybm@hotmail.com | 2008-12-09 12:29:56 -0500 (Tue, 09 Dec 2008) | 1 line

major change for instruments, now an assignment to handle env, passes basics tests
------------------------------------------------------------------------
r226 | thierrybm@hotmail.com | 2008-12-09 11:48:06 -0500 (Tue, 09 Dec 2008) | 1 line

3 basic tests for instruments
------------------------------------------------------------------------
```

r225 | benwarfield | 2008-12-08 18:55:56 -0500 (Mon, 08 Dec 2008) | 1 line

Created examples directory, with working GCD in it.

---

r224 | robstewart2 | 2008-12-08 18:38:24 -0500 (Mon, 08 Dec 2008) | 1 line

instrument definition is done. clip is in progress and commented out

---

r223 | thierrybm@hotmail.com | 2008-12-04 11:49:35 -0500 (Thu, 04 Dec 2008) | 1 line

more info on the type of whitespace encountered in debug mode

---

r222 | robstewart2 | 2008-12-03 17:47:39 -0500 (Wed, 03 Dec 2008) | 1 line

fixed pattern7.drultest

---

r221 | thierrybm@hotmail.com | 2008-12-03 17:46:02 -0500 (Wed, 03 Dec 2008) | 1 line

one test fixed

---

r220 | robstewart2 | 2008-12-03 17:44:28 -0500 (Wed, 03 Dec 2008) | 1 line

fixed pattern9.drultestout

---

r219 | thierrybm@hotmail.com | 2008-12-03 17:42:06 -0500 (Wed, 03 Dec 2008) | 1 line

removed useless comment

---

r218 | thierrybm@hotmail.com | 2008-12-03 17:40:07 -0500 (Wed, 03 Dec 2008) | 1 line

one test fixed

---

r217 | thierrybm@hotmail.com | 2008-12-03 17:39:14 -0500 (Wed, 03 Dec 2008) | 1 line

small assert added about mapper names

---

r216 | benwarfield | 2008-12-03 17:31:42 -0500 (Wed, 03 Dec 2008) | 1 line

Updated precedence of method calls.

---

r215 | thierrybm@hotmail.com | 2008-12-03 17:24:19 -0500 (Wed, 03 Dec 2008) | 1 line

one test fixed

---

r214 | thierrybm@hotmail.com | 2008-12-03 17:19:16 -0500 (Wed, 03 Dec 2008) | 1 line

one test fixed

---

r213 | benwarfield | 2008-12-03 17:12:12 -0500 (Wed, 03 Dec 2008) | 1 line

```
Changed printing output of beats, and updated tests accordingly.
------------------------------------------------------------------------
r212 | thierrybm@hotmail.com | 2008-12-03 17:07:18 -0500 (Wed, 03 Dec 2008) | 1 line

merged with Ben update
------------------------------------------------------------------------
r211 | benwarfield | 2008-12-03 17:07:05 -0500 (Wed, 03 Dec 2008) | 1 line

Patched svn:ignore on RefManual.
------------------------------------------------------------------------
r210 | robstewart2 | 2008-12-03 17:05:34 -0500 (Wed, 03 Dec 2008) | 1 line

forgot to add the rand tests
------------------------------------------------------------------------
r209 | robstewart2 | 2008-12-03 17:04:44 -0500 (Wed, 03 Dec 2008) | 1 line

added the rand function and tests for it
------------------------------------------------------------------------
r208 | benwarfield | 2008-12-03 17:03:07 -0500 (Wed, 03 Dec 2008) | 1 line

Added svn:ignore property to Proposal directory.
------------------------------------------------------------------------
r207 | benwarfield | 2008-12-03 16:57:29 -0500 (Wed, 03 Dec 2008) | 1 line

Beat methods (note,rest,prev,next) and simple tests.
------------------------------------------------------------------------
r206 | thierrybm@hotmail.com | 2008-12-03 16:54:08 -0500 (Wed, 03 Dec 2008) | 1 line

named mapper works with dollar signs, but not with other aliases like 'p'
------------------------------------------------------------------------
r205 | benwarfield | 2008-12-03 16:23:45 -0500 (Wed, 03 Dec 2008) | 1 line

Printing for Beats (with tests).
------------------------------------------------------------------------
r204 | robstewart2 | 2008-12-03 16:01:04 -0500 (Wed, 03 Dec 2008) | 1 line

added test cases for the slice function
------------------------------------------------------------------------
r203 | benwarfield | 2008-12-03 15:59:42 -0500 (Wed, 03 Dec 2008) | 1 line

Allow access to Beat objects inside map blocks.
------------------------------------------------------------------------
r202 | thierrybm@hotmail.com | 2008-12-03 15:44:55 -0500 (Wed, 03 Dec 2008) | 1 line

checks if we try to assign Beat or PatternAlias, and say something stupid about it
------------------------------------------------------------------------
r201 | thierrybm@hotmail.com | 2008-12-03 15:23:49 -0500 (Wed, 03 Dec 2008) | 1 line
```

<<<<<< removed
---
r200 | benwarfield | 2008-12-03 15:19:23 -0500 (Wed, 03 Dec 2008) | 1 line

Quashed warning in test.ml.
---
r199 | thierrybm@hotmail.com | 2008-12-03 15:16:52 -0500 (Wed, 03 Dec 2008) | 1 line

---
r198 | waseemilahi | 2008-11-30 21:58:20 -0500 (Sun, 30 Nov 2008) | 1 line

Slice method updated.
---
r197 | waseemilahi | 2008-11-27 00:17:40 -0500 (Thu, 27 Nov 2008) | 1 line

Some Tests editted for errors
---
r196 | waseemilahi | 2008-11-26 23:56:56 -0500 (Wed, 26 Nov 2008) | 1 line

Not Much
---
r195 | waseemilahi | 2008-11-26 23:51:50 -0500 (Wed, 26 Nov 2008) | 1 line

Spelling Corrections in the Header Comments
---
r194 | thierrybm@hotmail.com | 2008-11-26 19:27:27 -0500 (Wed, 26 Nov 2008) | 1 line

more comments
---
r193 | thierrybm@hotmail.com | 2008-11-26 19:16:50 -0500 (Wed, 26 Nov 2008) | 1 line

comments
---
r192 | thierrybm@hotmail.com | 2008-11-26 18:39:08 -0500 (Wed, 26 Nov 2008) | 1 line

specific exception created for illegal assignment
---
r191 | thierrybm@hotmail.com | 2008-11-26 18:35:55 -0500 (Wed, 26 Nov 2008) | 1 line

check at runtime for assignment of string and boolean
---
r190 | thierrybm@hotmail.com | 2008-11-26 18:25:27 -0500 (Wed, 26 Nov 2008) | 1 line

concat tests
---
r189 | thierrybm@hotmail.com | 2008-11-26 18:24:41 -0500 (Wed, 26 Nov 2008) | 1 line

concat tests

```
------------------------------------------------------------------------
r188 | benwarfield | 2008-11-26 15:33:49 -0500 (Wed, 26 Nov 2008) | 1 line

Dynamic scoping, and minor modifications to make beats work.
------------------------------------------------------------------------
r187 | thierrybm@hotmail.com | 2008-11-26 15:14:20 -0500 (Wed, 26 Nov 2008) | 1 line

one comment added
------------------------------------------------------------------------
r186 | benwarfield | 2008-11-26 15:13:02 -0500 (Wed, 26 Nov 2008) | 1 line

Made "return" work, and tested it.
------------------------------------------------------------------------
r185 | benwarfield | 2008-11-26 15:12:10 -0500 (Wed, 26 Nov 2008) | 1 line

Noted scope stuff.
------------------------------------------------------------------------
r184 | thierrybm@hotmail.com | 2008-11-26 14:46:18 -0500 (Wed, 26 Nov 2008) | 1 line

some commenting added
------------------------------------------------------------------------
r183 | thierrybm@hotmail.com | 2008-11-26 14:43:36 -0500 (Wed, 26 Nov 2008) | 1 line

some commenting added
------------------------------------------------------------------------
r182 | benwarfield | 2008-11-26 14:37:08 -0500 (Wed, 26 Nov 2008) | 1 line

Whitespace and comment changes.
------------------------------------------------------------------------
r181 | thierrybm@hotmail.com | 2008-11-26 14:00:32 -0500 (Wed, 26 Nov 2008) | 1 line

more tests fixed
------------------------------------------------------------------------
r180 | thierrybm@hotmail.com | 2008-11-26 13:57:58 -0500 (Wed, 26 Nov 2008) | 1 line

syntax of some tests updated, solved most of parser errors
------------------------------------------------------------------------
r179 | thierrybm@hotmail.com | 2008-11-26 13:53:28 -0500 (Wed, 26 Nov 2008) | 1 line

one test corrected
------------------------------------------------------------------------
r178 | benwarfield | 2008-11-26 03:15:22 -0500 (Wed, 26 Nov 2008) | 1 line

One simple test for map expression.
------------------------------------------------------------------------
r177 | benwarfield | 2008-11-26 02:59:02 -0500 (Wed, 26 Nov 2008) | 3 lines

Finished fixes relating to map scope entry, including finally figuring out how to
create the symbol table type we wanted in the first place.  Integrated changes back
```

```
into main interpreter codebase and deleted branch file.
------------------------------------------------------------------------
r176 | robstewart2 | 2008-11-24 18:20:19 -0500 (Mon, 24 Nov 2008) | 1 line

added concat and slice to the intepreter
------------------------------------------------------------------------
r175 | thierrybm@hotmail.com | 2008-11-24 18:01:05 -0500 (Mon, 24 Nov 2008) | 1 line

Ben, this is for you, I cannot fix the eval_arg_list, I've created two dummy
functions, still doesnt compile.... but we're almost there
------------------------------------------------------------------------
r174 | thierrybm@hotmail.com | 2008-11-24 17:46:49 -0500 (Mon, 24 Nov 2008) | 1 line

map may be solved, see function eval_arg_list, but still does not compile
------------------------------------------------------------------------
r173 | benwarfield | 2008-11-24 17:23:54 -0500 (Mon, 24 Nov 2008) | 1 line

BROKEN but Thierry will fix it--further work toward mapCall expression evaluation.
------------------------------------------------------------------------
r172 | waseemilahi | 2008-11-20 22:10:55 -0500 (Thu, 20 Nov 2008) | 1 line

 Repeat with argument value < 1 now raises exception for invalid argument
------------------------------------------------------------------------
r171 | waseemilahi | 2008-11-20 22:00:09 -0500 (Thu, 20 Nov 2008) | 1 line

pattern() is now accepted. Its a pattern of nothing
------------------------------------------------------------------------
r170 | waseemilahi | 2008-11-20 21:37:49 -0500 (Thu, 20 Nov 2008) | 1 line

Length member method done
------------------------------------------------------------------------
r169 | waseemilahi | 2008-11-20 20:31:43 -0500 (Thu, 20 Nov 2008) | 1 line

repeat member method finished
------------------------------------------------------------------------
r168 | benwarfield | 2008-11-20 19:17:32 -0500 (Thu, 20 Nov 2008) | 2 lines

Helper functions for initializing new symbol table when entering
new mapper scope.
------------------------------------------------------------------------
r167 | robstewart2 | 2008-11-20 18:39:59 -0500 (Thu, 20 Nov 2008) | 1 line

added repeat method handling to interpreter
------------------------------------------------------------------------
r166 | thierrybm@hotmail.com | 2008-11-20 18:18:26 -0500 (Thu, 20 Nov 2008) | 1 line

method call now is left associative
------------------------------------------------------------------------
r165 | benwarfield | 2008-11-20 18:00:40 -0500 (Thu, 20 Nov 2008) | 1 line
```

Upgraded patterns, and added mapper creation.

---

r164 | thierrybm@hotmail.com | 2008-11-20 17:57:34 -0500 (Thu, 20 Nov 2008) | 1 line

solved shiftreduce conflict on mcall by adding right additivity

---

r163 | thierrybm@hotmail.com | 2008-11-20 17:29:53 -0500 (Thu, 20 Nov 2008) | 1 line

more future tests

---

r162 | thierrybm@hotmail.com | 2008-11-20 17:26:52 -0500 (Thu, 20 Nov 2008) | 1 line

more future tests

---

r161 | thierrybm@hotmail.com | 2008-11-20 17:17:29 -0500 (Thu, 20 Nov 2008) | 1 line

more future tests

---

r160 | thierrybm@hotmail.com | 2008-11-20 17:09:52 -0500 (Thu, 20 Nov 2008) | 1 line

more future tests

---

r159 | thierrybm@hotmail.com | 2008-11-20 17:03:08 -0500 (Thu, 20 Nov 2008) | 1 line

more future tests

---

r158 | benwarfield | 2008-11-20 17:03:06 -0500 (Thu, 20 Nov 2008) | 1 line

Broke function calls out into their own function, and added a trap for invalid ones.

---

r157 | thierrybm@hotmail.com | 2008-11-20 16:56:45 -0500 (Thu, 20 Nov 2008) | 1 line

more future tests

---

r156 | thierrybm@hotmail.com | 2008-11-20 16:50:52 -0500 (Thu, 20 Nov 2008) | 1 line

more future tests

---

r155 | waseemilahi | 2008-11-20 16:38:57 -0500 (Thu, 20 Nov 2008) | 1 line

Added Basic Patterns

---

r154 | thierrybm@hotmail.com | 2008-11-20 14:16:29 -0500 (Thu, 20 Nov 2008) | 1 line

removeing useless file

---

r153 | thierrybm@hotmail.com | 2008-11-20 14:15:53 -0500 (Thu, 20 Nov 2008) | 1 line

```
minor changes
------------------------------------------------------------------------
r152 | benwarfield | 2008-11-20 00:00:55 -0500 (Thu, 20 Nov 2008) | 1 line

Fixed string-escape bug---test now passes!
------------------------------------------------------------------------
r151 | waseemilahi | 2008-11-19 22:44:07 -0500 (Wed, 19 Nov 2008) | 1 line


------------------------------------------------------------------------
r150 | benwarfield | 2008-11-19 18:45:28 -0500 (Wed, 19 Nov 2008) | 1 line

Variable assignment!!!
------------------------------------------------------------------------
r149 | benwarfield | 2008-11-19 18:09:48 -0500 (Wed, 19 Nov 2008) | 1 line

Added if/elseif/else to interpreter.
------------------------------------------------------------------------
r148 | thierrybm@hotmail.com | 2008-11-19 18:02:58 -0500 (Wed, 19 Nov 2008) | 1 line

all tests pass, pretty good parser
------------------------------------------------------------------------
r147 | thierrybm@hotmail.com | 2008-11-19 17:43:22 -0500 (Wed, 19 Nov 2008) | 1 line

parser kicks a**
------------------------------------------------------------------------
r146 | benwarfield | 2008-11-19 17:42:36 -0500 (Wed, 19 Nov 2008) | 1 line

Interpreter now supports all binary and unary operations (tests included).
------------------------------------------------------------------------
r145 | thierrybm@hotmail.com | 2008-11-19 17:34:39 -0500 (Wed, 19 Nov 2008) | 1 line

small updates
------------------------------------------------------------------------
r144 | thierrybm@hotmail.com | 2008-11-19 17:23:31 -0500 (Wed, 19 Nov 2008) | 1 line

update test
------------------------------------------------------------------------
r143 | thierrybm@hotmail.com | 2008-11-19 17:21:08 -0500 (Wed, 19 Nov 2008) | 1 line

update test
------------------------------------------------------------------------
r142 | thierrybm@hotmail.com | 2008-11-19 17:19:09 -0500 (Wed, 19 Nov 2008) | 1 line

one more test
------------------------------------------------------------------------
r141 | robstewart2 | 2008-11-19 17:08:20 -0500 (Wed, 19 Nov 2008) | 1 line

the interpreter can evaluate int arithmetic and print it
```

```
------------------------------------------------------------------------
r140 | thierrybm@hotmail.com | 2008-11-19 17:07:37 -0500 (Wed, 19 Nov 2008) | 1 line

better random
------------------------------------------------------------------------
r139 | thierrybm@hotmail.com | 2008-11-19 17:04:00 -0500 (Wed, 19 Nov 2008) | 1 line

one more test
------------------------------------------------------------------------
r138 | thierrybm@hotmail.com | 2008-11-19 16:54:08 -0500 (Wed, 19 Nov 2008) | 1 line

better parser, everything except if else ... seems to work
------------------------------------------------------------------------
r137 | thierrybm@hotmail.com | 2008-11-19 16:35:29 -0500 (Wed, 19 Nov 2008) | 1 line

added stupid stuff :)
------------------------------------------------------------------------
r136 | benwarfield | 2008-11-19 16:20:08 -0500 (Wed, 19 Nov 2008) | 1 line

Added environment to execution routines as (stringmap, parent) pair.
------------------------------------------------------------------------
r135 | thierrybm@hotmail.com | 2008-11-19 16:18:55 -0500 (Wed, 19 Nov 2008) | 1 line

a.b() and a.b(a) cases are parsed
------------------------------------------------------------------------
r134 | thierrybm@hotmail.com | 2008-11-19 16:14:46 -0500 (Wed, 19 Nov 2008) | 1 line

a.b case is parsed, now need to work or decide on a.b()
------------------------------------------------------------------------
r133 | thierrybm@hotmail.com | 2008-11-19 15:46:08 -0500 (Wed, 19 Nov 2008) | 1 line

better working LaunchTests, handle stdout and stderr, stderr assumed always at the end
------------------------------------------------------------------------
r132 | thierrybm@hotmail.com | 2008-11-19 15:38:55 -0500 (Wed, 19 Nov 2008) | 1 line

makefile now creates interpreter by default
------------------------------------------------------------------------
r131 | benwarfield | 2008-11-19 15:32:37 -0500 (Wed, 19 Nov 2008) | 2 lines

Added printing of numbers and booleans, added those to the printing test,
added a failing test for \\ and \", and updated svn:ignore to ignore logs.
------------------------------------------------------------------------
r130 | thierrybm@hotmail.com | 2008-11-19 15:21:14 -0500 (Wed, 19 Nov 2008) | 1 line

more real tests like hello world
------------------------------------------------------------------------
r129 | benwarfield | 2008-11-19 15:15:31 -0500 (Wed, 19 Nov 2008) | 1 line

Turned off scanner debugging.
```

```
------------------------------------------------------------------------
r128 | thierrybm@hotmail.com | 2008-11-19 15:11:20 -0500 (Wed, 19 Nov 2008) | 1 line
Launching test updated to reach program interpret
------------------------------------------------------------------------
r127 | thierrybm@hotmail.com | 2008-11-19 14:06:22 -0500 (Wed, 19 Nov 2008) | 1 line
new parser tests
------------------------------------------------------------------------
r126 | thierrybm@hotmail.com | 2008-11-19 14:02:41 -0500 (Wed, 19 Nov 2008) | 1 line
new parser tests
------------------------------------------------------------------------
r125 | thierrybm@hotmail.com | 2008-11-19 13:58:41 -0500 (Wed, 19 Nov 2008) | 1 line
new parser tests
------------------------------------------------------------------------
r124 | thierrybm@hotmail.com | 2008-11-19 13:23:29 -0500 (Wed, 19 Nov 2008) | 1 line
new parser tests
------------------------------------------------------------------------
r123 | thierrybm@hotmail.com | 2008-11-19 13:18:20 -0500 (Wed, 19 Nov 2008) | 1 line
new parser tests, correct the previous wrong extensions
------------------------------------------------------------------------
r122 | thierrybm@hotmail.com | 2008-11-19 13:16:13 -0500 (Wed, 19 Nov 2008) | 1 line
new parser tests
------------------------------------------------------------------------
r121 | thierrybm@hotmail.com | 2008-11-19 13:11:36 -0500 (Wed, 19 Nov 2008) | 1 line
new parser tests
------------------------------------------------------------------------
r120 | robstewart2 | 2008-11-12 18:44:55 -0500 (Wed, 12 Nov 2008) | 1 line
interpreter works for printing
------------------------------------------------------------------------
r119 | robstewart2 | 2008-11-12 18:23:51 -0500 (Wed, 12 Nov 2008) | 1 line
interpreter still doesn't work...
------------------------------------------------------------------------
r118 | robstewart2 | 2008-11-12 18:22:07 -0500 (Wed, 12 Nov 2008) | 1 line
interpreter still doesn't work...
------------------------------------------------------------------------
r117 | benwarfield | 2008-11-12 17:47:19 -0500 (Wed, 12 Nov 2008) | 1 line
Added tests from parser development side (three pass, one fails).
------------------------------------------------------------------------
```

r116 | benwarfield | 2008-11-12 17:38:52 -0500 (Wed, 12 Nov 2008) | 1 line

Escape for paths with spaces (actually by Thierry, but on my computer).

---

r115 | thierrybm@hotmail.com | 2008-11-12 17:38:46 -0500 (Wed, 12 Nov 2008) | 1 line

new drul test

---

r114 | thierrybm@hotmail.com | 2008-11-12 17:30:56 -0500 (Wed, 12 Nov 2008) | 1 line

new drul test

---

r113 | thierrybm@hotmail.com | 2008-11-12 17:26:36 -0500 (Wed, 12 Nov 2008) | 1 line

new drul test

---

r112 | thierrybm@hotmail.com | 2008-11-12 17:23:47 -0500 (Wed, 12 Nov 2008) | 1 line

test if testing program can be found

---

r111 | thierrybm@hotmail.com | 2008-11-12 17:20:16 -0500 (Wed, 12 Nov 2008) | 1 line

program to test parser

---

r110 | benwarfield | 2008-11-12 17:19:27 -0500 (Wed, 12 Nov 2008) | 1 line

Added support for end of file during a comment.

---

r109 | thierrybm@hotmail.com | 2008-11-12 17:18:36 -0500 (Wed, 12 Nov 2008) | 1 line

yeah! finding bugs

---

r108 | benwarfield | 2008-11-12 17:15:44 -0500 (Wed, 12 Nov 2008) | 1 line

Updated svn:ignore property to make status output less annoying.

---

r107 | benwarfield | 2008-11-12 17:14:22 -0500 (Wed, 12 Nov 2008) | 1 line

Detabbed parser, and added if/elseif/else support to parser and printer.

---

r106 | thierrybm@hotmail.com | 2008-11-12 17:13:07 -0500 (Wed, 12 Nov 2008) | 1 line

debugging parser testing

---

r105 | thierrybm@hotmail.com | 2008-11-12 17:05:10 -0500 (Wed, 12 Nov 2008) | 1 line

tester for parser seems to work

---

r104 | thierrybm@hotmail.com | 2008-11-12 16:37:20 -0500 (Wed, 12 Nov 2008) | 1 line

```
remove useless file
------------------------------------------------------------------------
r103 | thierrybm@hotmail.com | 2008-11-12 16:37:08 -0500 (Wed, 12 Nov 2008) | 1 line


------------------------------------------------------------------------
r102 | thierrybm@hotmail.com | 2008-11-12 16:27:08 -0500 (Wed, 12 Nov 2008) | 1 line

to test the parser, use with ./testing
------------------------------------------------------------------------
r101 | robstewart2 | 2008-11-11 16:04:29 -0500 (Tue, 11 Nov 2008) | 1 line

added the interpreter.  not done.  haven't even compiled it
------------------------------------------------------------------------
r100 | benwarfield | 2008-11-11 15:56:46 -0500 (Tue, 11 Nov 2008) | 2 lines

Fixed a bug resulting from change from left-recursion to right-recursion in
expr_list rule.
------------------------------------------------------------------------
r99 | benwarfield | 2008-11-11 15:41:24 -0500 (Tue, 11 Nov 2008) | 1 line

Added ugly-printer for very simple syntax trees.
------------------------------------------------------------------------
r98 | thierrybm@hotmail.com | 2008-11-11 14:52:27 -0500 (Tue, 11 Nov 2008) | 1 line

test suite ready
------------------------------------------------------------------------
r97 | thierrybm@hotmail.com | 2008-11-11 14:29:18 -0500 (Tue, 11 Nov 2008) | 1 line

better testing function
------------------------------------------------------------------------
r96 | benwarfield | 2008-11-11 14:22:34 -0500 (Tue, 11 Nov 2008) | 1 line

Added eol-style property to a couple files.
------------------------------------------------------------------------
r95 | benwarfield | 2008-11-11 14:09:41 -0500 (Tue, 11 Nov 2008) | 2 lines

Resolved expr_list problem by making it comma-separated, and added errata file
to RefManual folder to note such changes.
------------------------------------------------------------------------
r94 | benwarfield | 2008-11-11 13:50:52 -0500 (Tue, 11 Nov 2008) | 1 line

Made escaping a backslash in a string constant work.
------------------------------------------------------------------------
r93 | waseemilahi | 2008-11-10 22:28:55 -0500 (Mon, 10 Nov 2008) | 1 line

Example code from refmanual used for debuging
------------------------------------------------------------------------
```

```
r92 | benwarfield | 2008-11-10 16:22:45 -0500 (Mon, 10 Nov 2008) | 3 lines

Added "map" expressions to parser, and modified AST slightly to reflect the
fact that we do not have the parameter information for a mapper available in
the parser.
------------------------------------------------------------------------
r91 | benwarfield | 2008-11-10 15:32:49 -0500 (Mon, 10 Nov 2008) | 1 line

Made "block" reverse its statement list, so if/else/mapper need not do it.
------------------------------------------------------------------------
r90 | benwarfield | 2008-11-10 15:24:36 -0500 (Mon, 10 Nov 2008) | 2 lines

Added support for "else", and discovered several shift-reduce conflicts in
the grammar as currently specified.
------------------------------------------------------------------------
r89 | benwarfield | 2008-11-10 14:09:37 -0500 (Mon, 10 Nov 2008) | 3 lines

Added a level of indirection for statement lists (woohoo!), and added
assignment and mapper definition statements, as well as if-block, to
statement definition.  No handling of "else" token yet.
------------------------------------------------------------------------
r88 | benwarfield | 2008-11-08 23:35:18 -0500 (Sat, 08 Nov 2008) | 4 lines

Added testing program to actually parse input and see if it contains valid
DruL code.  Many modifications to scanner to make this fly, plus adding
string and boolean constants to the AST and the parser, and making the base
case for the parser (empty program) work.
------------------------------------------------------------------------
r87 | thierrybm@hotmail.com | 2008-11-08 19:52:14 -0500 (Sat, 08 Nov 2008) | 1 line

adding folders for test suite
------------------------------------------------------------------------
r86 | benwarfield | 2008-11-08 18:33:29 -0500 (Sat, 08 Nov 2008) | 1 line

Squished warning in scanner.
------------------------------------------------------------------------
r85 | benwarfield | 2008-11-08 18:31:21 -0500 (Sat, 08 Nov 2008) | 1 line

Reapplied accidentally backed-out bugfixes.
------------------------------------------------------------------------
r84 | robstewart2 | 2008-11-08 18:26:51 -0500 (Sat, 08 Nov 2008) | 1 line

kept my changes, not bens
------------------------------------------------------------------------
r83 | benwarfield | 2008-11-08 18:18:58 -0500 (Sat, 08 Nov 2008) | 2 lines

Added Makefile with appropriate dependencies to build everything as far as it
is so far possible to build it.
------------------------------------------------------------------------
```

```
r82 | benwarfield | 2008-11-08 18:18:30 -0500 (Sat, 08 Nov 2008) | 1 line

Added import of Parser module, and fixed identifier-too-long error message.
------------------------------------------------------------------------
r81 | benwarfield | 2008-11-08 18:00:11 -0500 (Sat, 08 Nov 2008) | 2 lines

Moved AST and Scanner to Parser directory, and fixed a couple of bugs in the
Parser, which now compiles all the way to an object file.
------------------------------------------------------------------------
r80 | benwarfield | 2008-11-08 17:28:29 -0500 (Sat, 08 Nov 2008) | 2 lines

Fixed syntax error and added support for statements and for calling functions
of one parameter.  Like "print".  Hypothetically speaking.
------------------------------------------------------------------------
r79 | benwarfield | 2008-11-08 17:12:29 -0500 (Sat, 08 Nov 2008) | 1 line

Operator precedence and simple expressions added to parser.
------------------------------------------------------------------------
r78 | benwarfield | 2008-11-08 16:35:23 -0500 (Sat, 08 Nov 2008) | 1 line

Removed blank lines and detabbed.
------------------------------------------------------------------------
r77 | benwarfield | 2008-11-08 16:31:02 -0500 (Sat, 08 Nov 2008) | 1 line

Fixed line-ending issues on parser/scanner.
------------------------------------------------------------------------
r76 | benwarfield | 2008-11-08 16:28:47 -0500 (Sat, 08 Nov 2008) | 1 line

Fixed problem with circular dependency in definitions of stmt/expr/mapper.
------------------------------------------------------------------------
r75 | benwarfield | 2008-11-05 22:09:25 -0500 (Wed, 05 Nov 2008) | 1 line

Fixed a couple more minor issues, but not the big one.
------------------------------------------------------------------------
r74 | benwarfield | 2008-11-05 22:08:04 -0500 (Wed, 05 Nov 2008) | 1 line

Fixed some, but not all, of the circularity problems in our AST definition.
------------------------------------------------------------------------
r73 | benwarfield | 2008-11-05 22:00:42 -0500 (Wed, 05 Nov 2008) | 1 line

Line-endings, tab expansion, and name consistency (arithOp/intOp).
------------------------------------------------------------------------
r72 | waseemilahi | 2008-11-05 21:32:11 -0500 (Wed, 05 Nov 2008) | 1 line

Minor change towards getting practical scanner for DruL
------------------------------------------------------------------------
r71 | benwarfield | 2008-11-05 18:11:40 -0500 (Wed, 05 Nov 2008) | 1 line

Improvements to AST.
```

```
------------------------------------------------------------------------
r70 | waseemilahi | 2008-11-03 18:14:48 -0500 (Mon, 03 Nov 2008) | 1 line

dummy parser introduced. need tokens to work in the scanner.
------------------------------------------------------------------------
r69 | waseemilahi | 2008-11-03 18:11:58 -0500 (Mon, 03 Nov 2008) | 1 line

dummy parser introduced. need tokens to work in the scanner.
------------------------------------------------------------------------
r68 | waseemilahi | 2008-11-03 09:37:00 -0500 (Mon, 03 Nov 2008) | 1 line

identifier is less than equal to 64 in length. Also eof termination added.
------------------------------------------------------------------------
r67 | waseemilahi | 2008-11-02 22:37:06 -0500 (Sun, 02 Nov 2008) | 1 line

some changes done to scanner. Still working on the basics.
------------------------------------------------------------------------
r66 | waseemilahi | 2008-10-24 11:07:19 -0400 (Fri, 24 Oct 2008) | 1 line

null removed in scanner
------------------------------------------------------------------------
r65 | benwarfield | 2008-10-22 17:50:47 -0400 (Wed, 22 Oct 2008) | 1 line

Trivial usage change.
------------------------------------------------------------------------
r64 | thierrybm@hotmail.com | 2008-10-22 17:50:33 -0400 (Wed, 22 Oct 2008) | 1 line

date fixed
------------------------------------------------------------------------
r63 | thierrybm@hotmail.com | 2008-10-22 17:48:03 -0400 (Wed, 22 Oct 2008) | 1 line

more on output.txtfile
------------------------------------------------------------------------
r62 | thierrybm@hotmail.com | 2008-10-22 17:34:08 -0400 (Wed, 22 Oct 2008) | 1 line

one textit removed
------------------------------------------------------------------------
r61 | thierrybm@hotmail.com | 2008-10-22 17:32:44 -0400 (Wed, 22 Oct 2008) | 1 line

empty pattern returned
------------------------------------------------------------------------
r60 | benwarfield | 2008-10-22 17:29:53 -0400 (Wed, 22 Oct 2008) | 1 line

Reformatted example code.
------------------------------------------------------------------------
r59 | thierrybm@hotmail.com | 2008-10-22 17:24:52 -0400 (Wed, 22 Oct 2008) | 1 line

typo in date fixed
------------------------------------------------------------------------
```

```
r58 | thierrybm@hotmail.com | 2008-10-22 17:24:32 -0400 (Wed, 22 Oct 2008) | 1 line

date fixed
------------------------------------------------------------------------
r57 | benwarfield | 2008-10-22 17:14:08 -0400 (Wed, 22 Oct 2008) | 1 line

Tweaked comment definition.
------------------------------------------------------------------------
r56 | thierrybm@hotmail.com | 2008-10-22 17:13:59 -0400 (Wed, 22 Oct 2008) | 1 line

just to be sure
------------------------------------------------------------------------
r55 | benwarfield | 2008-10-22 17:08:34 -0400 (Wed, 22 Oct 2008) | 1 line

Parenthesized example print statements, and removed newline characters.
------------------------------------------------------------------------
r54 | thierrybm@hotmail.com | 2008-10-22 17:07:04 -0400 (Wed, 22 Oct 2008) | 1 line

one line added to the print subsection
------------------------------------------------------------------------
r53 | thierrybm@hotmail.com | 2008-10-22 16:58:37 -0400 (Wed, 22 Oct 2008) | 1 line

() added
------------------------------------------------------------------------
r52 | thierrybm@hotmail.com | 2008-10-22 16:54:27 -0400 (Wed, 22 Oct 2008) | 1 line

better output
------------------------------------------------------------------------
r51 | benwarfield | 2008-10-22 16:49:51 -0400 (Wed, 22 Oct 2008) | 2 lines

Added $vars to Identifiers section, and changed reference from Mapper
section to Map section (since that section is more relevant).
------------------------------------------------------------------------
r50 | thierrybm@hotmail.com | 2008-10-22 16:48:42 -0400 (Wed, 22 Oct 2008) | 1 line

null and more on $
------------------------------------------------------------------------
r49 | thierrybm@hotmail.com | 2008-10-22 16:44:18 -0400 (Wed, 22 Oct 2008) | 1 line

null and more on $
------------------------------------------------------------------------
r48 | thierrybm@hotmail.com | 2008-10-22 16:34:16 -0400 (Wed, 22 Oct 2008) | 1 line

better indentation
------------------------------------------------------------------------
r47 | thierrybm@hotmail.com | 2008-10-22 16:33:35 -0400 (Wed, 22 Oct 2008) | 1 line

slice added
------------------------------------------------------------------------
```

```
r46 | benwarfield | 2008-10-22 16:25:53 -0400 (Wed, 22 Oct 2008) | 1 line

Added "beat" concept, and removed terminal semicolon from mapper definition.
------------------------------------------------------------------------
r45 | benwarfield | 2008-10-22 15:49:45 -0400 (Wed, 22 Oct 2008) | 1 line

Resolved a bunch of declare/define confusion.
------------------------------------------------------------------------
r44 | benwarfield | 2008-10-22 15:13:25 -0400 (Wed, 22 Oct 2008) | 1 line

Checking in various changes on behalf of Rob (whitespace and edits, largely).
------------------------------------------------------------------------
r43 | thierrybm@hotmail.com | 2008-10-22 13:50:08 -0400 (Wed, 22 Oct 2008) | 1 line

$ added in the example
------------------------------------------------------------------------
r42 | thierrybm@hotmail.com | 2008-10-22 13:34:14 -0400 (Wed, 22 Oct 2008) | 1 line

small typos and $ sign in anonymous mappers
------------------------------------------------------------------------
r41 | waseemilahi | 2008-10-22 09:25:03 -0400 (Wed, 22 Oct 2008) | 1 line

Made a folder for AST.
------------------------------------------------------------------------
r40 | waseemilahi | 2008-10-22 09:14:40 -0400 (Wed, 22 Oct 2008) | 1 line

I did add semicolons at the end of each mapper definition. They are
statements and statements end with a semicolon.
------------------------------------------------------------------------
r39 | waseemilahi | 2008-10-22 08:55:37 -0400 (Wed, 22 Oct 2008) | 1 line

Semicolons added at the end,of where anonymous mapper is used, because those
are definitely assignment statements.
------------------------------------------------------------------------
r38 | benwarfield | 2008-10-22 01:45:50 -0400 (Wed, 22 Oct 2008) | 1 line

Typographic cleanup, and minor textual revision, for beginning of example code.
------------------------------------------------------------------------
r37 | benwarfield | 2008-10-22 01:43:14 -0400 (Wed, 22 Oct 2008) | 1 line

Spruced up expression/statement section.
------------------------------------------------------------------------
r36 | waseemilahi | 2008-10-21 21:27:07 -0400 (Tue, 21 Oct 2008) | 1 line

Example Code Added. Some of the explanation in example section moved to the
appropriate earlier sections
------------------------------------------------------------------------
r35 | thierrybm@hotmail.com | 2008-10-20 17:50:00 -0400 (Mon, 20 Oct 2008) | 1 line
```

```
namepsace per type
------------------------------------------------------------------------
r34 | thierrybm@hotmail.com | 2008-10-20 17:45:54 -0400 (Mon, 20 Oct 2008) | 1 line

grrrrr
------------------------------------------------------------------------
r33 | benwarfield | 2008-10-20 17:34:29 -0400 (Mon, 20 Oct 2008) | 1 line

Spruced up expressions and statements a bit (boolean values).
------------------------------------------------------------------------
r32 | thierrybm@hotmail.com | 2008-10-20 17:24:29 -0400 (Mon, 20 Oct 2008) | 1 line


------------------------------------------------------------------------
r31 | thierrybm@hotmail.com | 2008-10-20 16:44:47 -0400 (Mon, 20 Oct 2008) | 1 line

new block scope section
------------------------------------------------------------------------
r30 | thierrybm@hotmail.com | 2008-10-20 16:25:17 -0400 (Mon, 20 Oct 2008) | 1 line


------------------------------------------------------------------------
r29 | benwarfield | 2008-10-20 14:09:17 -0400 (Mon, 20 Oct 2008) | 1 line

Closed scope for mappers, single namespace for identifiers.
------------------------------------------------------------------------
r28 | thierrybm@hotmail.com | 2008-10-20 13:26:34 -0400 (Mon, 20 Oct 2008) | 1 line

minor changes section 2.6
------------------------------------------------------------------------
r27 | thierrybm@hotmail.com | 2008-10-20 13:23:30 -0400 (Mon, 20 Oct 2008) | 1 line

minor changes section 2.6
------------------------------------------------------------------------
r26 | thierrybm@hotmail.com | 2008-10-20 13:13:49 -0400 (Mon, 20 Oct 2008) | 1 line


------------------------------------------------------------------------
r25 | thierrybm@hotmail.com | 2008-10-20 12:57:01 -0400 (Mon, 20 Oct 2008) | 1 line


------------------------------------------------------------------------
r24 | robstewart2 | 2008-10-20 12:48:17 -0400 (Mon, 20 Oct 2008) | 1 line


------------------------------------------------------------------------
r23 | benwarfield | 2008-10-20 02:30:20 -0400 (Mon, 20 Oct 2008) | 1 line

Rather rough draft of statement/expression/block section.
```

```
------------------------------------------------------------------------
r22 | thierrybm@hotmail.com | 2008-10-19 18:05:25 -0400 (Sun, 19 Oct 2008) | 1 line

improved map and mapper
------------------------------------------------------------------------
r21 | thierrybm@hotmail.com | 2008-10-19 17:44:15 -0400 (Sun, 19 Oct 2008) | 1 line

pattern section improved
------------------------------------------------------------------------
r20 | benwarfield | 2008-10-15 17:35:03 -0400 (Wed, 15 Oct 2008) | 1 line

Set svn:eol-style=native
------------------------------------------------------------------------
r19 | benwarfield | 2008-10-15 17:34:34 -0400 (Wed, 15 Oct 2008) | 1 line

Changes made during meeting (with whitespace issues ironed out).
------------------------------------------------------------------------
r18 | thierrybm@hotmail.com | 2008-10-15 17:21:40 -0400 (Wed, 15 Oct 2008) | 1 line

intro and examples added from proposal
------------------------------------------------------------------------
r17 | thierrybm@hotmail.com | 2008-10-15 16:44:45 -0400 (Wed, 15 Oct 2008) | 1 line

intro and examples added from proposal
------------------------------------------------------------------------
r16 | waseemilahi | 2008-10-15 10:29:38 -0400 (Wed, 15 Oct 2008) | 1 line

Minor addition to the manual.(Few typos corrected)
------------------------------------------------------------------------
r15 | waseemilahi | 2008-10-15 09:40:40 -0400 (Wed, 15 Oct 2008) | 4 lines

Some Basic functionality added.
I am using test code inside lex to check for certain
conditions. After we have the parser it will look
quite different.
------------------------------------------------------------------------
r14 | waseemilahi | 2008-10-15 09:20:22 -0400 (Wed, 15 Oct 2008) | 1 line


------------------------------------------------------------------------
r13 | benwarfield | 2008-10-13 17:05:47 -0400 (Mon, 13 Oct 2008) | 1 line

Scratch version of AST, by Ben and Rob.
------------------------------------------------------------------------
r12 | thierrybm@hotmail.com | 2008-10-13 15:31:32 -0400 (Mon, 13 Oct 2008) | 1 line

minor
------------------------------------------------------------------------
r11 | thierrybm@hotmail.com | 2008-10-12 15:42:47 -0400 (Sun, 12 Oct 2008) | 1 line
```

```
some basics stuff added
------------------------------------------------------------------------
r10 | thierrybm@hotmail.com | 2008-10-11 14:35:50 -0400 (Sat, 11 Oct 2008) | 1 line

outputs
------------------------------------------------------------------------
r9 | thierrybm@hotmail.com | 2008-10-11 14:20:10 -0400 (Sat, 11 Oct 2008) | 1 line

most of the sections are there
------------------------------------------------------------------------
r8 | thierrybm@hotmail.com | 2008-10-11 13:54:22 -0400 (Sat, 11 Oct 2008) | 1 line

beginning of the draft
------------------------------------------------------------------------
r7 | thierrybm@hotmail.com | 2008-10-11 13:36:20 -0400 (Sat, 11 Oct 2008) | 1 line


------------------------------------------------------------------------
r6 | waseemilahi | 2008-10-09 20:39:20 -0400 (Thu, 09 Oct 2008) | 2 lines

Directory for scanner added to the project.
Scanner doesn't contain anything useful yet.
------------------------------------------------------------------------
r5 | waseemilahi | 2008-10-09 19:56:57 -0400 (Thu, 09 Oct 2008) | 1 line

The transfer of all the files from old repository to the current one.
------------------------------------------------------------------------
r4 | waseemilahi | 2008-10-09 19:53:34 -0400 (Thu, 09 Oct 2008) | 1 line


------------------------------------------------------------------------
r3 | waseemilahi | 2008-10-09 19:51:39 -0400 (Thu, 09 Oct 2008) | 1 line


------------------------------------------------------------------------
r2 | waseemilahi | 2008-10-09 19:47:55 -0400 (Thu, 09 Oct 2008) | 1 line


------------------------------------------------------------------------
r1 | (no author) | 2008-10-08 15:23:21 -0400 (Wed, 08 Oct 2008) | 1 line

Initial directory structure.
------------------------------------------------------------------------
```

Appendix C

Code Listings

C.1 Language code

C.1.1 drul_interpreter.ml

```
(*
**************************************************************************
*                 DruL - Drumming Language
*
* Creation of R. Stewart, T. Bertin-Mahieux, W. Ilahi  and B. Warfield
*              rs2660       tb2332              wki2001        bbw2108
*
* for the class COMS W4115: Programming Language and Translators
*
* file: drul_interpreter.ml
*
*                         INTERPRETER
*
* This file contains the interpreter for DruL. It receives an AST
* and interprets the code.
* This code is written in OCaml.
*
**************************************************************************
*)

open Drul_ast
open Drul_main
open Drul_types

let run p env = match p with
    Content(statements) -> Random.self_init(); ignore(execlist statements env)

let _ =
```

```
let unscoped_env = {symbols = NameMap.empty; parent = None} in
let arglen = Array.length Sys.argv in
let input_stream = if 1 < arglen then open_in Sys.argv.(1) else stdin in
let lexbuf = Lexing.from_channel input_stream in
let programAst = Drul_parser.program Drul_scanner.token lexbuf in
try run programAst unscoped_env
with Type_error(msg,line)          ->
    Printf.fprintf stderr "Type error on line %d: %s\n" line msg
| Invalid_function(msg,line)       ->
    Printf.fprintf stderr "Invalid function call on line %d: %s\n" line msg
| PatternParse_error(msg,line)     ->
    Printf.fprintf stderr "Invalid pattern string on line %d: %s\n" line msg
| Invalid_argument(msg,line)       ->
    Printf.fprintf stderr "Incorrect function arguments on line %d: %s\n" line msg
| Undefined_identifier(msg,line)   ->
    Printf.fprintf stderr "Reading undefined identifier '%s' attempted on line %d.\n" msg line
| Illegal_assignment(msg,line)     ->
    Printf.fprintf stderr "Illegal assignment attempted on line %d: %s\n" line msg
| Instruments_redefined(msg,line) ->
    Printf.fprintf stderr "Instrument redefinition attempted on line %d: %s\n" line msg
| Illegal_division(msg,line)       ->
    Printf.fprintf stderr "Division by zero attempted on line %d: %s\n" line msg
| Failure(msg)                     ->
    Printf.fprintf stderr "Untrapped internal error! (error message: %s)\n" msg
```

C.1.2 drul_main.ml

```
(*
**************************************************************************
*                 DruL - Drumming Language
*
* Creation of R. Stewart, T. Bertin-Mahieux, W. Ilahi  and B. Warfield
*              rs2660       tb2332              wki2001        bbw2108
*
* for the class COMS W4115: Programming Language and Translators
*
* file: drul_main.ml
*
*       MAIN
*
* This file contains the main driver functions for the DruL interpreter.
* This code is written in OCaml.
*
**************************************************************************
*)

open Drul_ast
open Drul_types
open Drul_helpers
open Drul_output

(* default instruments *)
let default_instr = ["hh_c"; "sd_ac"; "bd"; "cowbell"]
```

```
let keyword_map =
    List.fold_left
    (fun m k -> NameMap.add k true m)
    NameMap.empty
    ["clip";"rand";"mapper";"concat";"pattern";"return";"instruments";
    "slice";"print";"output";"map";"if";"else";"elseif";"true";"false"]

(* exception used to handle return statement, similar to MicroC from Edwards *)
exception Return_value of drul_env

(*
    inside a map, do one step!
    return is saved as "return" in the env
    current index is saved as "$current" in the env
*)
let rec one_mapper_step maxiters current st_list env current_pattern =
    if (maxiters == current) then Pattern(current_pattern)
    else
        let retval = Pattern([]) in
        let env = add_key_to_env env "return" retval in
        let env = add_key_to_env env "$current" (Int(current)) in
        let block_line = (match (List.hd st_list) with
                Expr(e)      -> e.lineno
            |   Return(e)    -> e.lineno
            |   Assign(_,_,lineno) -> lineno
            |   MapDef(_,_,_,lineno) -> lineno
            |   IfBlock(e,_,_)       -> e.lineno
            |   InstrDef(_,lineno)  -> lineno
            |   EmptyStat(lineno)   -> lineno
        ) in
        let newenv = execlist_returning st_list env in
        let new_st = newenv.symbols in
        let return = NameMap.find "return" new_st in
        let current_pattern =
        (
            match return with
                Pattern(bools) -> current_pattern @ bools
            |   Beat(alias_bools,idx) ->
                    if ((idx >= 0) && (idx < (Array.length alias_bools) ))
                    then current_pattern @ [alias_bools.(idx)]
                    else current_pattern
            |   _ -> (raise (Illegal_assignment
                ("attempt to return an illegal value from this mapper",block_line)
                ))
        )
        in
        let current = current + 1 in
        one_mapper_step maxiters current st_list newenv current_pattern

(*
    run a named mapper,
    find the mapper in the env,
    and cast it to a anonymous mapper
*)
and run_named_mapper mapname argList env lineno =
  let savedmapper = get_key_from_env env mapname lineno in
```

```
    match savedmapper with
    Mapper(mapname2, a_list, stat_list) ->
      (* check if we receive the good number of patterns *)
      if List.length a_list != List.length argList then raise
        (Invalid_argument ("wrong number of inputs for named mapper", lineno))
      else if String.compare mapname mapname2 != 0 then raise
        (Failure "in run_named_mapper, should not happen (intern mapper name problem)")
      else run_mapper stat_list argList env a_list
      (* if given name is not bound to a mapper, Type_error *)
      | _ -> raise
        (Type_error ("we were expecting a mapper, name associated with something else", lineno))

(*
    main function of a map, takes a list of statement (body of the mapper)
    evaluate the arg_list, which should be a list of patterns
    launches the iteration (one_mapper_step)
*)
and run_mapper statement_list arg_list env a_list =
    let arg_list_evaled = eval_arg_list arg_list env in
    let map_env = get_map_env env arg_list_evaled a_list in
    let max_iters = find_longest_list arg_list_evaled in
    one_mapper_step max_iters 0 statement_list map_env []

(* evaluate an expr_list when we know that they're all patterns *)
and eval_arg_list arg_list env = match arg_list with
        [] -> []
    |   headExp::tail ->
        (
            let headVal =  evaluate headExp env
            in headVal :: (eval_arg_list tail env)
        )



(* evaluate expressions, no modifications to the environment! *)
and evaluate e env = match e.real_expr with
        FunCall(fname, fargs) -> function_call fname fargs env e.lineno
    |   MethodCall(objectExpr, mname, margs) -> method_call objectExpr mname margs env
    |   CStr (x) -> Str (x)
    |   CBool(x) -> Bool(x)
    |   CInt (x) -> Int (x)
    |   Var(name) -> let fetched = get_key_from_env env name e.lineno in        (
            match fetched with
                PatternAlias(alias) -> beat_of_alias env alias e.lineno
            |   other -> other
        )
    |   UnaryMinus(xE) -> let xV = evaluate xE env in
        (
            match xV with
                Int(x) -> Int(-x)
            |   _      -> raise (Type_error ("you can't negate that, dorkface", e.lineno))
        )
    |   UnaryNot(xE) ->  let xV = evaluate xE env in
        (
            match xV with
                Bool(x) -> Bool(not x)
```

```
            |   _       -> raise (Type_error ("you can't contradict that", e.lineno))
        )
    |   ArithBinop(aExp, operator, bExp) ->
        let aVal = evaluate aExp env in
        let bVal = evaluate bExp env in
        (
            match (aVal, operator, bVal) with
                (Int(a), Add,  Int(b)) -> Int(a + b)
            |   (Int(a), Sub,  Int(b)) -> Int(a - b)
            |   (Int(a), Mult, Int(b)) -> Int(a * b)
            |   (Int(a), Div,  Int(b)) -> if(b != 0) then Int(a / b) else raise
                    (Illegal_division( "Divisor evaluates to 0", e.lineno))
            |   (Int(a), Mod,  Int(b)) -> Int(a mod b)
            |   _ -> raise (Type_error ("cannot do arithmetic on non-integers", e.lineno))
        )
    |   LogicBinop(aExp, operator, bExp) ->
        let aVal = evaluate aExp env in
        let bVal = evaluate bExp env in
        (
            match (aVal, operator, bVal) with
                (Bool(x), And, Bool(y)) -> Bool(x && y)
            |   (Bool(x), Or,  Bool(y)) -> Bool(x || y)
            |   _ -> raise (Type_error ("cannot do logical operations except on booleans", e.lineno))
        )
    |   Comparison(aExp, operator, bExp) ->
        let aVal = evaluate aExp env in
        let bVal = evaluate bExp env in
        (
            match (aVal, operator, bVal) with
                (Int(a), LessThan,    Int(b)) -> Bool(a <  b)
            |   (Int(a), GreaterThan, Int(b)) -> Bool(a >  b)
            |   (Int(a), LessEq,      Int(b)) -> Bool(a <= b)
            |   (Int(a), GreaterEq,   Int(b)) -> Bool(a >= b)
            |   (Int(a), EqualTo,     Int(b)) -> Bool(a == b)
            |   (Int(a), NotEqual,    Int(b)) -> Bool(a != b)
            |   _ -> raise (Type_error ("cannot do that comparison operation", e.lineno))
        )
    |   MapCall(someMapper,argList) ->
        (
            match someMapper with
                AnonyMap(stList)  -> run_mapper stList argList env []
            |   NamedMap(mapname) -> run_named_mapper mapname argList env e.lineno
        )
    |   InstrAssign(instName, patExpr) -> let patVal = evaluate patExpr env in
        (
            match patVal with
                Pattern(p) -> InstrumentAssignment(instName, p)
            |   _          -> raise (Invalid_argument ("Only patterns can be assigned to instruments", e.lineno))
        )



(*
    function calls, anything looking like a() or a(something)
    the major 'match' is done on a
*)
```

```
and function_call fname fargs env lineno =
    let fargvals = eval_arg_list fargs env in
    match (fname, fargvals) with
        ("pattern", []) -> Pattern([])
    |   ("pattern", [v]) ->
        (
            match v with
                Str(x) ->
                (
                    let charlist = Str.split (Str.regexp "") x
                    in let revlist =
                        List.fold_left
                        (
                            fun bl str ->
                            (
                                match str with
                                    "0" -> false
                                |   "1" -> true
                                |   _   -> raise (PatternParse_error
                                    ("Patterns definitions must be a string of 0's and 1's", lineno))
                            ) :: bl
                        )
                        [] charlist
                        in Pattern(List.rev revlist)
                )
            |   _ -> raise (Type_error ("Pattern definitions take a string argument",lineno))
        )
    |   ("print", []) -> print_endline ""; Void
    |   ("print", [v]) ->
        (
            match v with
                Str(x)     -> print_endline x; Void
            |   Int(y)     -> print_endline(string_of_int y); Void
            |   Bool(z)    -> print_endline(if z then "TRUE" else "FALSE"); Void
            |   Pattern(p) -> let pstr = string_of_pattern p in print_endline pstr;Void
            |   Beat(_,_)  -> print_endline(string_of_beat v); Void
            |   Clip(ar) -> print_endline(string_of_clip ar env); Void
                | _ -> print_endline("Dunno how to print this yet."); Void
        )
    |   ("concat", concat_args) -> let catenated = concat_pattern_list concat_args lineno in Pattern(catenated)
    |   ("rand", []) -> Int(Random.int 2)
    |   ("rand", [argVal]) ->
        (
            match argVal with
                Int(bound) -> if bound > 0 then Int(Random.int bound)
                              else raise
                                (Invalid_argument ("'rand' expects a positive integer argument", lineno))
            |   _ -> raise (Invalid_argument ("'rand' expects an integer argument", lineno))
        )
    |   ("rand", _) -> raise
            (Invalid_argument ("'rand' expects a single, optional, positive, integer argument", lineno))
    |   ("clip", argList) -> make_clip argList env lineno
    |   (other, _) -> (* TODO: currently also catches invalid argument-counts,
                         which should probably be intercepted further up the line *)
            let msg = "Function name '" ^ other ^ "' is not a valid function." in
                raise (Invalid_function (msg, lineno))
```

```
(*
Method Calls, anything looking like a.b() or a.b(something)
the major 'match' is usually done on both a and b
*)
and method_call objectExpr mname margs env =
    let objectVal = evaluate objectExpr env in
    let argVals   = eval_arg_list margs env in
    let lineno = objectExpr.lineno          in
    match (objectVal, mname, argVals) with
        (Pattern(x), "repeat", margs) ->
        (
            match margs with
                [argVal] ->
                (
                    match argVal with
                    Int(y) -> if      (y  < 0) then raise
                                (Invalid_argument ("Repeat can only accept non-negative integers", lineno))
                              else if (y == 0) then Pattern([])
                              else let rec repeatPattern p n = if n == 1 then p else p @ repeatPattern p (n-1)
                              in Pattern(repeatPattern x y)
                    | _ -> raise
                       (Invalid_function ("Method repeat expects an integer argument", lineno))
                )
            |   _ -> raise (Invalid_function ("Method repeat expects a single argument", lineno))
        )
    |   (Pattern(x), "length", margs) ->
        (
             match margs with
                []  ->  Int(List.length x)
            |   _   -> raise (Invalid_function ("Method length expects no arguments", lineno))
        )
    |   (Pattern(x), "reverse",argVal) ->
                (
                    match argVal with
                        []         -> Pattern(List.rev x)
                    |   _          -> raise (Invalid_function("Method reverse expects no arguments",lineno))
                )
    |   (Pattern(x), "slice", [startVal; lenVal]) ->
        (
            match (startVal, lenVal) with
                (Int(s), Int(l)) -> if s < 1 || (s > List.length x && List.length x > 0)
                                    then raise (Invalid_argument("the start position is out of bounds", lineno))
                                    else if l < 0  then raise
                                        (Invalid_argument ("the length must be non-negative", lineno))
                                    else  let rec subList inList i minPos maxPos =
                                    (
                                        match inList with
                                          []         -> []
                                        | head::tail -> if      i < minPos then subList tail (i+1) minPos maxPos
                                                        else if i = maxPos then [head]
                                                        else if i > maxPos then []
                                                        else           head :: (subList tail (i+1) minPos maxPos)
                                    )
```

```
                                in Pattern(subList x 1 s (s+l-1))
        |   (_, _) -> raise
            (Invalid_argument ("slice must be given integer values for start position and length", lineno))
    )
|   (Beat(a,i), "isnull", []) -> let beatval = state_of_beat objectVal in
    (
        match beatval with
            Some(_)     -> Bool(false)
        |   None        -> Bool(true)
    )
|   (Beat(a,i), "note", []) -> let beatval = state_of_beat objectVal in
    (
        match beatval with
            Some(yesno) -> Bool(yesno)
        |   None        -> Bool(false)
    )
|   (Beat(a,i), "rest", []) -> let beatval = state_of_beat objectVal in
    (
        match beatval with
            Some(yesno) -> Bool(not yesno)
        |   None        -> Bool(false)
    )
|   (Beat(a,i), "prev", [offsetVal]) ->
    (
        match offsetVal with
            Int(offsetInt) -> let newidx = i - offsetInt in Beat(a,newidx)
        |   _              -> raise
                (Invalid_function ("Beat method 'prev' requires an integer argument", lineno))
    )
|   (Beat(a,i), "next", [offsetVal]) ->
    (
        match offsetVal with
            Int(offsetInt) -> let newidx = i + offsetInt in Beat(a,newidx)
        |   _              -> raise
                (Invalid_function ("Beat method 'next' requires an integer argument", lineno))
    )
|   (Beat(a,i), "asPattern", []) -> let beatval = state_of_beat objectVal in
    (
        match beatval with
            Some(yesno) -> Pattern([yesno])
        |   None        -> Pattern([])
    )
|   (Clip(ar), "outputText", args) ->
    (
        match args with
        [Str(fileName)] ->
            if (String.length fileName) < 1
            then raise (
                Invalid_argument ("Output filename is empty", lineno)
            )
            else
                let formatted_clip = string_of_clip ar env in
                let out = open_out fileName in
                    output_string out formatted_clip;
                    close_out out;
                    Void
```

```
            | _ -> raise (Invalid_function ("clip method 'outputText' requires a filename", lineno))
        )
    |   (Clip(ar), "outputMidi", args) ->
        (
            match args with
            [Str(fileName); Int(tempo)] ->
                if (String.length fileName) < 1
                    then raise (Invalid_argument ("Output filename empty", lineno))
                else if tempo < 1
                    then raise (Invalid_argument ("Tempo must be positive", lineno))
                else
                    let out = Unix.open_process_out ("midge -q -o " ^ fileName) in
                    output_string out (midge_of_clip ar env tempo);
                    let output_status = (Unix.close_process_out out) in (match output_status with
                        Unix.WEXITED(_) -> ignore();
                        | _ -> raise (Failure "midge process terminated abnormally")
                        );
                    Void
                    | _ -> raise
                        (Invalid_function ("clip method 'outputMidi' requires a filename and tempo", lineno))
        )
    |   (Clip(ar), "outputLilypond", args) ->
        (
            let fileName = (match args with Str(f)::_ -> f
                | _ -> raise
                    (Invalid_function ("clip method 'outputLilypond' requires a filename and title", lineno)))
                in
            let clipname = (match args with _::[] -> "DruL Output" | _::[Str(n)] -> n
                | _ ->raise
                    (Invalid_function ("clip method 'outputLilypond' requires a filename and title", lineno)))
                in
                if (String.length fileName) < 1
                then raise (Invalid_argument ("Output filename empty", lineno))
                else
                    let out = open_out fileName in
                    output_string out (lilypond_page_of_clip ar env clipname);
                    close_out out;
                    Void
        )
    | _ -> raise (Invalid_function ("Undefined method function",lineno))


(* similar to evaluate, but handles cases like assignment, where the environment is modified *)
and execute s env = match s with
    Expr(e) -> ignore(evaluate e env); env
|   IfBlock(tExpr, iftrue, iffalse) -> let tVal = evaluate tExpr env in
    (
        match tVal with
            Bool(true)  -> execlist iftrue env
        |   Bool(false) ->
            (
                match iffalse with
                    Some(stlist) -> execlist stlist env
                |   None         -> env
            )
```

```
        |   _ -> raise (Type_error ("test of if block must be a boolean", tExpr.lineno))
    )
|   Assign(varName,valExpr, lineno) ->
    (
        if (NameMap.mem varName keyword_map) then
            raise(Illegal_assignment("can't use keyword '" ^ varName ^ "' as a variable", lineno))
        else
        let valVal = evaluate valExpr env in
            (
                match valVal with
                    Bool(x)         -> raise(Illegal_assignment ("can't assign a boolean", lineno))
                |   Str(x)          -> raise(Illegal_assignment ("can't assign a string", lineno))
                |   Beat(x,y)       -> raise(Illegal_assignment ("can't assigna beat", lineno))
                |   PatternAlias(x) -> raise(Illegal_assignment ("can't assign a PatternAlias", lineno))
                |   Mapper(_,_,_)   -> raise(Illegal_assignment ("can't assign a mapper",lineno))
                | _ -> add_key_to_env env varName valVal
                    (* Does in fact mask variables in outer scope! Not an error! *)
            )
    )
|   MapDef(mapname, formal_params, contents, lineno) ->
            if (NameMap.mem mapname keyword_map)
        then raise(Illegal_assignment("can't use keyword '" ^ mapname ^ "' as a mapper name", lineno))
        else
        if (NameMap.mem mapname env.symbols)
        then raise (Illegal_assignment("can't give an in-use name to a mapper", lineno))
        else
            let newMapper = Mapper(mapname,formal_params,contents) in
            let newST = NameMap.add mapname newMapper env.symbols in
            {symbols = newST; parent = env.parent}
|   Return(retExpr) ->
    (
        match env.parent with
            None -> raise(Failure "in execute, case Return, should not happen (None parent?)")
        |   _    -> if (not (NameMap.mem "return" env.symbols)) then raise (Failure "still don't")
                    else
                    let retVal = evaluate retExpr env in
                    let newenv = add_key_to_env env "return" retVal in
                    raise (Return_value newenv)
    )
|   InstrDef(argList, lineno) ->
    (
        try
            ignore(get_key_from_env env "instruments" lineno);
            raise (Instruments_redefined ("don't do that", lineno)) (*XXX could be improved...*)
        with
            Undefined_identifier (_,_) ->
              (* make sure were not in a map, so env.parent == None *)
              (match env.parent with Some(_) ->
               raise (Illegal_assignment ("can't define instruments inside mappers", lineno))
                | _ ->
              let strList = eval_arg_list argList env in
                let str_to_string a =
                (
                    match a with
                        Str(s) -> s
                    |   _      -> raise (Invalid_argument ("instruments takes a list of strings", lineno))
```

```
                ) in
                let stringList = List.map str_to_string strList in
                (
                    match stringList with
                        [] -> add_key_to_env env "instruments" (Instruments(default_instr)) (* default *)
                    |   _ -> let instVal = Instruments(stringList) in
                                add_key_to_env env "instruments" instVal
                )
              )
            | Instruments_redefined(e,i) -> raise (Instruments_redefined (e,i))
            (*| Illegal_assignment(e,i) -> raise (Illegal_assignment (e,i))*)
            | _ -> raise (Failure "in execute, case InstrDef, unexpected exception")
    )
|   EmptyStat(_) -> env


and execlist slist env = List.fold_left (fun env s -> execute s env) env slist

(* special case used for mapper, when we expect a return value *)
and execlist_returning slist env =
    try List.fold_left (fun env s -> execute s env) env slist
    with
        Return_value(newenv) -> newenv
    |   other -> raise other
```

C.1.3 drul_helpers.ml

```
(*
**************************************************************************
*                DruL - Drumming Language
*
* Creation of R. Stewart, T. Bertin-Mahieux, W. Ilahi  and B. Warfield
*             rs2660       tb2332              wki2001        bbw2108
*
* for the class COMS W4115: Programming Language and Translators
*
* file: drul_helpers.ml
*
*       HELPERS
*
* This file contains the helper functions (anything that is not required
* to be mutually recursive with "evaluate") for the DruL interpreter.
* This code is written in OCaml.
*
**************************************************************************
*)

open Drul_ast
open Drul_types



(*
```

```
    create an empty clip of given size (an array of empty patterns)
    assumes non empty list (clipLen > 0)
*)
let emptyClip clipSize =
    let rec emptyPatternList len =
        if len == 1 then [[]]
        else (List.append [[]] (emptyPatternList (len - 1)))
    in Array.of_list (emptyPatternList clipSize)



(*
    turn a pattern object (list of booleans) into an array, and return
    pairs of (array, alias) to be added to the symbol table
*)
let rec get_alias_list p_list a_list counter =
    let newcounter = counter + 1 in
        match (p_list,a_list) with
        ([],[]) -> []
    |   ([],oops) -> raise (Failure "not enough patterns provided to mapper")
    |   (thispat::rest,[]) -> (thispat, "$" ^ string_of_int counter) :: get_alias_list rest [] newcounter
    |   (thispat::rest,thisalias::other_aliases) ->
            let dollar_alias = "$" ^ (string_of_int counter) in
            [(thispat, dollar_alias); (thispat, thisalias)]
            @ get_alias_list rest other_aliases newcounter



(*
    given a NameMap and a (pattern, alias) pair,
    add the appropriate information to the NameMap
    (at this point, an array of the beats is the pattern)
*)
let add_pattern_alias  symbol_table pair =
    let p_obj = fst(pair) in
    let alias = snd(pair) in
    let p_list =
    (
        match p_obj with
            Pattern(pat) -> pat
        |   _            -> raise (Failure "in add_pattern_alias, should not happen")
    ) in
    let p_array = Array.of_list p_list in
    let beat_holder = PatternAlias(p_array)
    in NameMap.add alias beat_holder symbol_table

(*
    use the above functions to add the correct entries to a new symbol table
    before entering a "map" block
*)
let init_mapper_st p_list a_list =
    let alias_list = get_alias_list p_list a_list 1
    in List.fold_left add_pattern_alias NameMap.empty alias_list

(* create a new symbol table with the appropriate aliases, and link it to the parent *)
let get_map_env parent_env p_list a_list =
```

```
    let new_symbol_table = init_mapper_st p_list a_list
    in {symbols = new_symbol_table; parent = Some(parent_env)}

(* is called by find_longest_list *)
let maxlen_helper currmax newlist =
    match newlist with
    Pattern(patlist) ->
    (
        let currlen = List.length patlist in
        if (currlen > currmax) then currlen else currmax
    )
    | _ -> raise (Failure "in maxlen_helper, should not happen (not a pattern?)")

(* find the length of the longest list *)
let find_longest_list patternlist = List.fold_left maxlen_helper 0 patternlist

(*
    Adds a given key & value to env in (env, parentEnv).
    Returns the modified env.
*)
let add_key_to_env env key value =
    match env with {symbols = old_st; parent = whatever} ->
        let new_st = NameMap.add key value old_st
        in {symbols = new_st; parent = whatever}

(* retrieve the value for a given key from the environment
    or its parent.
   If the value is a PatternAlias, then use some magic to transform
   it into a Beat
*)
let rec get_key_from_env env key lineno =
    if NameMap.mem key env.symbols then NameMap.find key env.symbols
    else match env.parent with
            Some(parent_env) -> get_key_from_env parent_env key lineno
        |   None -> raise (Undefined_identifier (key, lineno))

(* takes an alias, turns it into a beat object (used in mapper) *)
and  beat_of_alias env alias lineno =
    let currentVar = get_key_from_env env "$current" lineno
    in match currentVar with
            Int(currentVal) -> Beat(alias,currentVal)
        |   _ -> raise (Failure "in beat_of_alias, can't have a non-integer in $current")

let state_of_beat beat =
    match beat with
        Beat(pattern_data,idx) ->
            let pattern_length = Array.length pattern_data in
            if (idx < 0 or idx >= pattern_length) then None else Some(pattern_data.(idx))
    |   _ -> raise (Failure "in state_of_beat, should not happen (not a beat?)")

(* get an array with the names of the current instruments in it *)
let get_instr_name_array env =
    (* TODO: make this a less hackish way to avoid passing that line-number around? *)
    let drulInstrList = get_key_from_env env "instruments" 0 in
    match drulInstrList with
            Instruments(l) -> Array.of_list l
```

```
        | _ -> raise (Failure "slot for instruments does not contain instruments")


(*
find the position of an instrument in the instruments in the env, returns -1 if doesn't find it
*)
let get_instrument_pos env instrName lineno =
    try
        let instrListDrul = get_key_from_env env "instruments" lineno in
        match instrListDrul with
            Instruments(instrList) ->
                let rec find_pos strList counter  =
                (
                    match strList with
                        []          -> -1
                    |   head::tail  -> if (String.compare head instrName) == 0 then counter
                                       else find_pos tail (counter + 1)
                )
                in find_pos instrList 0
        |   _ -> raise (Failure "in get_instrument_pos, weird stuff in env for instruments...")
    with
        Undefined_identifier(e,i) -> raise (Failure "in get_instument_pos, instrument not saved in env yet")
    |   Failure(e)                -> raise (Failure e)
    |   _                         -> raise (Failure "in get_instrument_pos, wrong or new exception")

(* concat patterns into one *)
let rec concat_pattern_list plist lineno =
    match plist with
        [] -> []
    |   Pattern(x)::tail -> x @ (concat_pattern_list tail) lineno
    |   _ -> raise (Invalid_argument ("concat only concatenates patterns", lineno))


(*
get an empty clip (clip with the right number of empty patterns)
and fills it from a pattern list
*)
let rec fill_in_clip_patterns empty_clip pattern_list idx lineno = match pattern_list with
        [] -> Clip(empty_clip) (* not technically empty any more *)
            (* TODO: catch array out of bounds here *)
    |   Pattern(p)::tail ->
            ignore(empty_clip.(idx) <- p);
            fill_in_clip_patterns empty_clip tail (idx + 1) lineno
    |   InstrumentAssignment(_,_)::tail ->
            raise (Invalid_argument ("clip arguments may not mix styles", lineno))
    |   _ ->
            raise (Invalid_argument ("clip arguments must all evaluate to patterns", lineno))

(*
similar as fill_in_clip_patterns, but deals with the InstrumentAssignments 'hihat' <- pattern("1")
*)
let rec fill_in_clip_instr_assigns empty_clip assignment_list env lineno = match assignment_list with
        [] -> Clip(empty_clip) (* not technically empty any more *)
    |   InstrumentAssignment(instrName,p)::tail ->
            let idx = get_instrument_pos env instrName lineno in
              if idx < 0
```

```
            then raise (Invalid_argument ("unknown instrument name '" ^ instrName ^"'",lineno))
            else
          ignore(empty_clip.(idx) <- p); fill_in_clip_instr_assigns empty_clip tail env lineno
    |   Pattern(_)::tail ->raise (Invalid_argument ("clip arguments may not mix styles",lineno))
    |   _   -> raise (Invalid_argument ("clip arguments must all evaluate to instrument assignments",lineno))


(* first function in order to make a clip *)
let make_clip argVals env lineno =
    try
    (
        let instrument_list = get_key_from_env env "instruments" lineno in
        let num_instrs =
        (
            match instrument_list with
                Instruments(i) -> List.length i
            |   _              -> raise (Failure "in make_clip, should not happen")
        ) in
        let new_clip = emptyClip num_instrs in
        let first_arg = List.hd argVals in
        (
            match first_arg with
                Pattern(_) -> fill_in_clip_patterns new_clip argVals 0 lineno
            |   InstrumentAssignment(_,_) ->fill_in_clip_instr_assigns new_clip argVals env lineno
            |   _ -> raise
                    (Invalid_argument ("clip arguments must be patterns or instrument assignments", lineno))
        )
    )
    with Undefined_identifier("instruments",i) -> raise
        (Illegal_assignment ("trying to create a clip before defining instruments", i))
```

C.1.4 drul_output.ml

```
(* helper functions for all non-trivial forms of output
    * created by Ben Warfield
    * (contents also authored partially by Rob Stewart--this file is a refactor)
    * 12/17/2008
*)


open Drul_types
open Drul_helpers

(* Oh, Printf.sprintf... we've only just met, and yet already I hate you with
 * a grim, joyless spite that would do a COBOL programmer proud.
 *)

let lilypond_staff_format  = (
    "   \\new DrumStaff\n\t\\with{
        instrumentName = \"%s\"
        drumStyleTable = #percussion-style
        \\override StaffSymbol #'line-count = #1
        \\remove Time_signature_engraver    \n\t}\n\t\\drummode { %s }\n"
    : ('a -> 'b -> 'c, unit, string) format
```

```
    )

let lilypond_page_format = (
    "\\header{\n\ttitle = \"%s\"\n}\n<<\n%s\n>>\n\\version \"2.10.33\"\n"
    : ('a -> 'b -> 'c, unit, string) format
)

let string_of_beat b =
    let state = state_of_beat b in
    match state with
        None    -> "NULL"
    |   Some(b) -> if b then "NOTE" else "REST"


(* turn a pattern into a string, using predefined strings for "yes" and "no" *)
let folded_pattern p ifyes ifno =
    List.fold_left (fun a x -> a ^ (if x then ifyes else ifno)) "" p


(* get a string out of a pattern, pattern("0101") becomes "0101" *)
let string_of_pattern p = folded_pattern p "1" "0"

(* get a midge-formatted string for the supplied instrument out of a pattern *)
let string_of_instr_pattern p i = folded_pattern p (i ^ " ") "r "


(* problem: getting the name in makes this less generic *)
let lilypond_staff_of_pattern iname p =
    let note_string = folded_pattern p "tri4 " "r4 " in
    let tmp = lilypond_staff_format  in
    Printf.sprintf tmp iname note_string




let lilypond_page_of_clip clip_contents env title =
    let inames = get_instr_name_array env in
    assert ((Array.length inames) >= (Array.length clip_contents));
    let staff_strings = Array.mapi
        (fun idx pat -> lilypond_staff_of_pattern inames.(idx) pat)
        clip_contents in
    let all_staffs = Array.fold_left (fun a b -> a ^ b) "" staff_strings in
    Printf.sprintf lilypond_page_format title all_staffs

let string_of_clip clip_contents env =
    let instrument_names = get_instr_name_array env in
    assert ((Array.length instrument_names) >= (Array.length clip_contents));
    let formatted_strings = Array.mapi
        (fun idx p -> instrument_names.(idx) ^":\t" ^ string_of_pattern p)
        clip_contents in
    let all_patterns = Array.fold_left
        (fun a str -> a ^ "\t" ^ str ^ "\n")
        "" formatted_strings in
    "[\n" ^ all_patterns ^ "]"
```

```
let midge_of_clip clip_contents env tempo =
    let inames = get_instr_name_array env in
    assert ((Array.length inames) >= (Array.length clip_contents));
    let pattern_strings = Array.mapi
        (fun idx p -> if (0 < List.length p)
            then (
                "\t@channel 10 " ^ inames.(idx) ^ " { /L4/"
                ^ (string_of_instr_pattern p inames.(idx)) ^ " }\n"
            )
            else ""
        )
        clip_contents in
    "@head {\n"
        ^ "$tempo " ^ (string_of_int tempo) ^ "\n"
        ^ "$time_sig 4/4" ^ "\n"
        ^ "}\n"
        ^"@body {\n"
        ^ (Array.fold_left (fun a s->a^s) "" pattern_strings)
        ^ "\n}\n"
```

C.1.5 drul_printer.ml

```
(* Drul_printer package
    Pretty-print a Drul AST
    11/11/2008
*)

open Drul_ast

let string_of_intop = function
        Add -> "Addition"
    |   Sub -> "Subtraction"
    |   Mult ->"Multiplication"
    |   Div  -> "Division"
    |   Mod  -> "Modulus"

let string_of_compop = function
        EqualTo -> "Equality test"
    |   NotEqual -> "Inequality test"
    |   LessThan -> "Less than"
    |   GreaterThan ->"Greater than"
    |   LessEq  -> "Less than/equal to"
    |   GreaterEq -> "Greater than/equal to"

let string_of_boolop = function
        And -> "Conjunction"
    |   Or -> "Disjunction"


let rec string_of_expr = function
        CInt(x) -> "Constant integer " ^ string_of_int(x)
    |   CStr(s) -> "Constant string [" ^  s ^"]"
    |   CBool(b)-> "Constant " ^ if b then "TRUE" else "FALSE"
    |   Var(id) -> "Variable name " ^ id
```

```
    |   UnaryMinus(neg) -> "Arithmetic negation of " ^ string_of_expr(neg)
    |   UnaryNot(bool)  -> "Logical negation of " ^ string_of_expr(bool)
    |   ArithBinop(a,op,b) ->"Arithmetic operation: " ^ string_of_intop(op)
                ^ ": left operand= "^ string_of_expr(a)
                ^ "; right operand= " ^ string_of_expr(b)
    |   LogicBinop(a,op,b) -> string_of_boolop(op) ^ " of " ^ string_of_expr(a)
                ^ " with "^ string_of_expr(b)
    |   Comparison(a,op,b) -> "Comparison of type " ^ string_of_compop(op)
                ^ ": left operand= " ^ string_of_expr(a)
                ^ "; right operand= " ^ string_of_expr(b)
    |   FunCall(name,arglist) -> "Call to function '"^name
        ^ "' with these arguments: "
        ^ List.fold_left (fun a ex -> a ^ string_of_expr(ex) ^ "; ") "" arglist
    |   MapCall(m,arglist) -> "Called 'map' on arguments: "
            ^ (List.fold_left (fun a ex -> a ^ string_of_expr(ex) ^ "; ") "" arglist)
            ^ " Using Mapper=" ^ string_of_mapper(m)
and  string_of_mapper = function
        NamedMap(name) -> name
    |   AnonyMap(list) -> "a statement list we can't evaluate yet"
and string_of_statement = function
        Expr(e) -> "Simple statement: " ^ string_of_expr(e)
    |   IfBlock(exp,stlist,None) -> "If block.  Condition: " ^ string_of_expr(exp)
            ^ string_of_block "TRUE" stlist ^ "\t(No else)\n"
    |   IfBlock(exp,stlist,Some(elses)) ->"If block.  Condition: " ^ string_of_expr(exp)
            ^ string_of_block "TRUE" stlist
            ^ string_of_block "FALSE" elses
    |   _ -> "Something not yet covered."
and string_of_block name stlist =
    "\tStatements in block "^ name^ ":\n"
    ^ List.fold_left (fun s x -> s ^ "\t" ^string_of_statement(x) ^ "\n") "" stlist


let string_of_program = function
    Content(l) -> "Statements in this program:\n"
        ^ List.fold_left (fun s x -> s ^ string_of_statement( x ) ^ "\n" ) "" l
```

C.1.6 drul_types.ml

```
(*
************************************************************************
*                DruL - Drumming Language
*
* Creation of R. Stewart, T. Bertin-Mahieux, W. Ilahi  and B. Warfield
*             rs2660       tb2332              wki2001        bbw2108
*
* for the class COMS W4115: Programming Language and Translators
*
* file: drul_types.ml
*
*       TYPES
*
* This file contains the internal type and exception declarations
* required by the interpreter and printing/checking functions.
*
```

```
*************************************************************************
*)

open Drul_ast

module NameMap = Map.Make(String)

(* most of the exceptions *)
exception Type_error             of string * int
exception Invalid_function       of string * int
exception PatternParse_error     of string * int
exception Invalid_argument       of string * int
exception Undefined_identifier   of string * int
exception Illegal_assignment     of string * int
exception Instruments_redefined  of string * int
exception Illegal_division       of string * int

type pattern       = bool list
type pattern_alias = bool array

(* type of every object in DruL *)
type drul_t =
        Void
    |   Int     of int
    |   Str     of string
    |   Bool    of bool
    |   Pattern of pattern
    |   Clip    of pattern array
    |   Mapper  of (string * string list * statement list)
    |   PatternAlias of pattern_alias
    |   Beat of pattern_alias * int
    |   Instruments of string list
    |   InstrumentAssignment of string * pattern

(*  symbol table for DruL:
    the current environment is 'symbols': a map from string to drul_t,
    the parent is another drul_env
*)
type drul_env =
{
    symbols: drul_t NameMap.t;
    parent:  drul_env option
}
```

C.1.7 drul_parser.mly

```
%{
open Drul_ast
open Lexing

let debug str = if (true) then ignore(print_endline str) else ignore()

let string_of_two_positions start_pos end_pos =
    let start_line = start_pos.pos_lnum in
```

```
    let end_line = end_pos.pos_lnum in
    let start_char = start_pos.pos_cnum - start_pos.pos_bol in
    let end_char = end_pos.pos_cnum - end_pos.pos_bol in
    if ( end_line = start_line ) then
        if ( end_char == start_char )
            then Printf.sprintf "on line %d after character %d"
                start_line start_char
            else Printf.sprintf "on line %d between characters %d and %d"
                start_line start_char end_char
    else
        Printf.sprintf "between char %d of line %d and char %d of line %d"
            start_char start_line end_char end_line

let parse_error str =
    let start_pos = Parsing.symbol_start_pos() in
    let end_pos   = Parsing.symbol_end_pos()   in
    prerr_endline ("Syntax error " ^  string_of_two_positions start_pos end_pos);
    exit(2)

%}

%token <int> IF ELSE ELSEIF RETURN
%token <int> TRUE FALSE
%token <int> MAP MAPDEF LARROW CLIP
%token <int> SEMI LPAREN RPAREN LBRACE RBRACE COMMA PLUS MINUS TIMES DIVIDE
%token <int> ASSIGN EQ NEQ LT LEQ GT GEQ EOF MCALL AND OR NOT MOD
%token <int> INSTRUMENTS
%token <int    * int> INTLITERAL
%token <string * int> STRLITERAL ID

%left LIST
%nonassoc ELSE
%left ASSIGN LARROW
%left INSTRUMENTS
%left OR
%left AND
%left NEQ EQ
%left LT GT LEQ GEQ
%left PLUS MINUS
%left TIMES DIVIDE MOD
%nonassoc UMINUS NOT
%left MCALL

%start program
%type<Drul_ast.program> program
%%

expr:
        INTLITERAL { { real_expr = CInt(fst($1)); lineno = snd($1) } }
    |   STRLITERAL { { real_expr = CStr(fst $1);  lineno = snd($1) } }
    |   TRUE       { { real_expr = CBool(true);   lineno = $1      } }
    |   FALSE      { { real_expr = CBool(false);  lineno = $1      } }
    |   ID         { { real_expr = Var(fst $1);   lineno = snd($1) } }
    |   expr PLUS   expr { { real_expr = ArithBinop($1, Add,         $3); lineno = $2 } }
    |   expr MINUS  expr { { real_expr = ArithBinop($1, Sub,         $3); lineno = $2 } }
    |   expr TIMES  expr { { real_expr = ArithBinop($1, Mult,        $3); lineno = $2 } }
```

```
    |   expr DIVIDE expr { { real_expr = ArithBinop($1, Div,         $3); lineno = $2 } }
    |   expr MOD    expr { { real_expr = ArithBinop($1, Mod,         $3); lineno = $2 } }
    |   expr EQ     expr { { real_expr = Comparison($1, EqualTo,     $3); lineno = $2 } }
    |   expr NEQ    expr { { real_expr = Comparison($1, NotEqual,    $3); lineno = $2 } }
    |   expr LT     expr { { real_expr = Comparison($1, LessThan,    $3); lineno = $2 } }
    |   expr GT     expr { { real_expr = Comparison($1, GreaterThan, $3); lineno = $2 } }
    |   expr LEQ    expr { { real_expr = Comparison($1, LessEq,      $3); lineno = $2 } }
    |   expr GEQ    expr { { real_expr = Comparison($1, GreaterEq,   $3); lineno = $2 } }
    |   expr AND    expr { { real_expr = LogicBinop($1, And,         $3); lineno = $2 } }
    |   expr OR     expr { { real_expr = LogicBinop($1, Or,          $3); lineno = $2 } }
    |   MINUS expr %prec UMINUS { { real_expr = UnaryMinus($2); lineno = $1 } }
    |   NOT expr { { real_expr = UnaryNot($2); lineno = $1 } }
    |   ID LPAREN expr_list RPAREN { { real_expr = FunCall(fst($1), $3); lineno = snd($1) } }
    |   ID LPAREN           RPAREN { { real_expr = FunCall(fst($1), []); lineno = snd($1) } }
    |   expr MCALL ID LPAREN           RPAREN { { real_expr = MethodCall($1, fst($3), []); lineno = $2 } }
    |   expr MCALL ID LPAREN expr_list RPAREN { { real_expr = MethodCall($1, fst($3), $5); lineno = $2 } }
    |   LPAREN expr RPAREN { { real_expr = $2.real_expr; lineno = $1} }
    |   MAP LPAREN expr_list RPAREN block { { real_expr = MapCall(AnonyMap($5),      $3); lineno = $1 } }
    |   MAP LPAREN expr_list RPAREN ID    { { real_expr = MapCall(NamedMap(fst($5)), $3); lineno = $1 } }
    |   STRLITERAL LARROW expr { { real_expr = InstrAssign(fst($1), $3); lineno = $2 } }

statement:
        expr SEMI { Expr($1) }
    |   RETURN expr SEMI { Return($2) }
    |   MAPDEF ID LPAREN id_list RPAREN block { MapDef((fst $2), List.rev $4, $6, snd($2)) }
    |   ID ASSIGN expr SEMI { Assign(fst($1), $3, snd($1)) }
    |   IF LPAREN expr RPAREN block iftail { IfBlock($3, $5, $6) }
    |   INSTRUMENTS LPAREN expr_list RPAREN SEMI { InstrDef($3, $1) }
    |   INSTRUMENTS LPAREN RPAREN SEMI           { InstrDef([], $1) }
    |   SEMI { EmptyStat($1) }

block:
    LBRACE st_list RBRACE { List.rev $2 }

id_list:
        ID { [fst($1)] }
    |   id_list COMMA ID { fst($3)::$1 }

expr_list:
        expr { [$1] }
    |   expr COMMA expr_list { $1::$3 }

st_list:
    /* staring into the abyss */ { [] }
    | st_list statement { $2::$1 } /* build statement list backward */

program:
    st_list { Content(List.rev $1) }

iftail:
        ELSEIF LPAREN expr RPAREN block iftail { Some([ IfBlock($3,$5,$6) ]) }
    |   ELSE block { Some($2) }
    |   /* nothing */ { None }
;
```

C.1.8 drul_scanner.mll

```
{
    open Drul_parser
    open Lexing
    let debugging   = ref false
    let standalone  = ref false
    let line_number = ref 1
    let set_debug() = debugging := true
    let debug str   = if (!debugging) then ignore(print_endline str) else ignore()
    let escape_re   = Str.regexp "\\\\\\\(\\\\\\\|\"\\\)"
    (* "\\\\\\([\\\"]\\)" also works, almost as ugly *)
    let escape_repl = "\\1"


    (* In 3.11 this is built in to Lexing, but alas, I have 3.10...*)
    (* This code largely borrowed from a newgroup post by Till Varoquaux
     * complaining about it not being built in:
     * http://caml.inria.fr/pub/ml-archives/caml-list/2008/03/4575c51493931878a25de6b1712a4b24.en.html
     *)
    let new_line lexbuf =
    incr line_number;
    let pos = lexbuf.lex_curr_p in
    lexbuf.lex_curr_p <- {
        pos with
            pos_lnum = pos.pos_lnum + 1;
            pos_bol = pos.pos_cnum
    }

}

let digit      =  ['0' - '9']+
let identifier =  ['a'-'z' 'A'-'Z' '_']['a'-'z' 'A'-'Z' '_' '0'-'9']*

rule token = parse
        ' '                             { debug("whitespace 'b '"); token lexbuf }
    |   '\t'                            { debug("whitespace 't'");  token lexbuf }
    |   '\r'                            { debug("whitespace 'r'");  token lexbuf }
    |   '\n'                            { debug("whitespace 'n'");
                                          new_line lexbuf;
                                          token lexbuf }
    |   "//"                            { debug "COMMENT"; comment lexbuf }
    |   '('                             { debug "LPAREN"; LPAREN(!line_number) }
    |   ')'                             { debug "RPAREN"; RPAREN(!line_number) }
    |   '{'                             { debug "LBRACE"; LBRACE(!line_number) }
    |   '}'                             { debug "RBRACE"; RBRACE(!line_number) }
    |   ';'                             { debug "SEMI";   SEMI(!line_number)   }
    |   ','                             { debug "COMMA";  COMMA(!line_number)  }
    |   '+'                             { debug "PLUS";   PLUS(!line_number)   }
    |   '-'                             { debug "MINUS";  MINUS(!line_number)  }
    |   '*'                             { debug "TIMES";  TIMES(!line_number)  }
    |   '/'                             { debug "DIVIDE"; DIVIDE(!line_number) }
    |   '='                             { debug "ASSIGN"; ASSIGN(!line_number) }
    |   "=="                            { debug "EQ";  EQ(!line_number)  }
    |   "!="                            { debug "NEQ"; NEQ(!line_number) }
    |   '!'                             { debug "NOT"; NOT(!line_number) }
```

```
    |     '%'                          { debug "MOD"; MOD(!line_number) }
    |     '<'                          { debug "LT";  LT(!line_number)  }
    |     "<="                         { debug "LEQ"; LEQ(!line_number) }
    |     '>'                          { debug "GT";  GT(!line_number)  }
    |     ">="                         { debug "GEQ"; GEQ(!line_number) }
    |     "&&"                         { debug "AND"; AND(!line_number) }
    |     "||"                         { debug "OR";  OR(!line_number)  }
    |     '.'                          { debug "MCALL"; MCALL(!line_number)    }
    |     "true"                       { debug "TRUE";  TRUE(!line_number)     }
    |     "false"                      { debug "FALSE"; FALSE(!line_number)    }
    |     "if"                         { debug "IF";      IF(!line_number)     }
    |     "else"                       { debug "ELSE";    ELSE(!line_number)   }
    |     "elseif"                     { debug "ELSEIF"; ELSEIF(!line_number) }
    |     "mapper"                     { debug "MAPDEF"; MAPDEF(!line_number) }
    |     "map"                        { debug "MAP";     MAP(!line_number)    }
    |     "return"                     { debug "RETURN"; RETURN(!line_number) }
    |     "instruments"                { debug "INSTRUMENTS"; INSTRUMENTS(!line_number) }
    |     "<-"                         { debug "LARROW"; LARROW(!line_number) }
    |     '$' digit as numbers         { debug("index variable " ^ numbers); ID(numbers, !line_number) }
    |     identifier as ide            {
                                           if ((String.length ide) <= 64)
                                           then
                                           (
                                               debug("identifier " ^ ide);
                                               ID(ide,!line_number)
                                           )
                                           else raise (Failure("ID TOO LONG: " ^ ide))
                                       }
    |     digit as dig                 { debug ("digits " ^ dig); INTLITERAL(int_of_string dig, !line_number) }
    |     '"' (( ('\\' [ '"' '\\' ] ) | [^ '\r' '\n' '\\' '"'] )* as rawstr) '"'
                                       {
                                           (* TODO: accept newlines, then raise "illegal character in string?" *)
                                           let fixedstr = Str.global_replace escape_re escape_repl rawstr in
                                           debug(("string constant [" ^ fixedstr ^ "]"));
                                           STRLITERAL(fixedstr, !line_number)
                                       }
    |     eof                          { debug "EOF"; EOF(!line_number) }
    |     _  as char                   { raise (Failure("illegal character " ^ Char.escaped char)) }

and comment = parse
        '\n'                           { new_line lexbuf; token lexbuf }
    |     eof                          { debug "EOF"; EOF(!line_number) }
    |     _                            { comment lexbuf                 }


{
    if (!standalone) then
     let lexbuf = Lexing.from_channel stdin in
     let rec nexttoken buf = ignore(token buf);nexttoken buf
     in nexttoken lexbuf
    else ignore()
}
```

C.1.9 test.ml

```
open Drul_ast

let _ =
let lexbuf = Lexing.from_channel stdin in
let _ = Drul_parser.program Drul_scanner.token lexbuf in
print_endline "Parsed program (somewhat) successfully!"
(*let listing = Printer.string_of_program program in
print_string listing *)
```

C.1.10 treedump.ml

```
open Drul_printer

let _ =
let lexbuf = Lexing.from_channel stdin in
let program = Drul_parser.program Drul_scanner.token lexbuf in
print_endline (string_of_program program)
```

C.1.11 drul_ast.mli

```
(* AST scratch *)

type intOp  = Add | Sub | Mult | Div | Mod

type compOp = EqualTo | NotEqual | LessThan | GreaterThan | LessEq | GreaterEq

type boolOp = And | Or

type mapper =
        AnonyMap of statement list
    |   NamedMap of string

and expr =
        CInt  of int
    |   CStr  of string
    |   CBool of bool
    |   Var of string
    |   UnaryMinus of tagged_expr
    |   UnaryNot   of tagged_expr
    |   ArithBinop of tagged_expr * intOp * tagged_expr
    |   LogicBinop of tagged_expr * boolOp * tagged_expr
    |   Comparison of tagged_expr * compOp * tagged_expr
    |   FunCall    of              string * tagged_expr list
    |   MethodCall of tagged_expr * string * tagged_expr list
    |   MapCall    of mapper              * tagged_expr list
    |   InstrAssign of string * tagged_expr

and statement =
```

```
        Expr     of tagged_expr
    |   Return   of tagged_expr
    |   Assign   of string * tagged_expr * int
    |   MapDef   of string * string list * statement list * int
    |   IfBlock  of tagged_expr * statement list * statement list option
    |   InstrDef of tagged_expr list * int
    |   EmptyStat of int

and tagged_expr = { real_expr : expr ; lineno : int }

type program = Content of statement list
```

C.1.12 Makefile

```
OC = ocamlc
CFLAGS = # none for now

OBJS = drul_scanner.cmo drul_parser.cmo drul_types.cmo \
    drul_helpers.cmo drul_output.cmo drul_main.cmo
LIBS = str.cma unix.cma

all : $(OBJS) drul

testing: test.cmo $(OBJS)
    $(OC) $(CFLAGS) -o testing $(LIBS) $(OBJS) test.cmo

treedump: treedump.cmo $(OBJS)
    $(OC) $(CFLAGS) -o treedump $(LIBS) $(OBJS) drul_printer.cmo treedump.cmo

scantest: drul_scanner.cmo scantest.cmo
    $(OC) $(CFLAGS) -o scantest $(LIBS) $<

drul: drul_interpreter.cmo $(OBJS) drul_ast.cmi
    $(OC) $(CFLAGS) -o drul $(LIBS) $(OBJS) drul_interpreter.cmo

drul_scanner.ml : drul_scanner.mll
    ocamllex $<

drul_parser.ml drul_parser.mli : drul_parser.mly
    ocamlyacc $<

%.cmo : %.ml
    $(OC) $(CFLAGS) -c $<

%.cmi : %.mli
    $(OC) $(CFLAGS) -c $<

.PHONY : clean
clean :
    rm -f  drul_parser.ml drul_parser.mli drul_scanner.ml *.cmo *.cmi testing treedump drul

# Generated by ocamldep *.ml *.mli
drul_helpers.cmo: drul_types.cmo drul_ast.cmi
drul_helpers.cmx: drul_types.cmx drul_ast.cmi
```

```
drul_interpreter.cmo: drul_types.cmo drul_scanner.cmo drul_parser.cmi \
    drul_main.cmo drul_ast.cmi
drul_interpreter.cmx: drul_types.cmx drul_scanner.cmx drul_parser.cmx \
    drul_main.cmx drul_ast.cmi
drul_main.cmo: drul_types.cmo drul_helpers.cmo drul_ast.cmi
drul_main.cmx: drul_types.cmx drul_helpers.cmx drul_ast.cmi
drul_output.cmo: drul_types.cmo drul_helpers.cmo
drul_output.cmx: drul_types.cmx drul_helpers.cmx
drul_parser.cmo: drul_ast.cmi drul_parser.cmi
drul_parser.cmx: drul_ast.cmi drul_parser.cmi
drul_printer.cmo: drul_ast.cmi
drul_printer.cmx: drul_ast.cmi
drul_scanner.cmo: drul_parser.cmi
drul_scanner.cmx: drul_parser.cmx
drul_types.cmo: drul_ast.cmi
drul_types.cmx: drul_ast.cmi
test.cmo: drul_scanner.cmo drul_parser.cmi drul_ast.cmi
test.cmx: drul_scanner.cmx drul_parser.cmx drul_ast.cmi
treedump.cmo: drul_scanner.cmo drul_printer.cmo drul_parser.cmi
treedump.cmx: drul_scanner.cmx drul_printer.cmx drul_parser.cmx
drul_parser.cmi: drul_ast.cmi
```

C.2 Test Code

C.2.1 LaunchTests.py

```
#! /usr/bin/env python
"""
DruL team, Columbia (2008) PLT class
copyright DruL team

contact: tb2332@columbia.edu

name: LaunchTests.py
language: python
programer: Thierry Bertin-Mahieux

main program of the test suite, launch all tests that it can find.
"""


import os
import sys
import glob
import time
import tempfile



drulpath = "../"
testspath = "./Tests/"
```

```
logspath = "./LOGS/"
mainprog = "../Parser/drul"

# returns a list of file in current dir
# to use with os.walk
def grab_tests(arg=list(),path="",names=""):
    tests = glob.glob(os.path.join(os.path.abspath(path),'*.drultest'))
    for t in tests:
        arg.append(t)
    return arg


# make sure that all tests found have a corresponding output
# if not, program exits
def make_sure_tests_have_outputs(tests):
    noout = list()
    for t in tests :
        if not os.path.exists(t + 'out'):
            print (t+'out')
            noout.append(t)
        if len(noout) > 0:
            print 'problem,',len(noout),'tests have no corresponding output'
            print 'we stop testing..... go solve it! and grab a beer'
            print 'files that cause problems:'
            for t in noout:
                print t
            sys.exit(0)


# launch any command, return outputs
def command_with_output(cmd):
    if not type(cmd) == unicode :
        cmd = unicode(cmd,'utf-8')
    #should this be a part of slashify or command_with_output?
    #if sys.platform=='darwin' :
    #    cmd = unicodedata.normalize('NFC',cmd)

    (child_stdin,child_stdout,child_stderr) = os.popen3(cmd.encode('utf-8'))
    data1 = child_stdout.read()
    data2 = child_stderr.read()
    child_stdout.close()
    child_stderr.close()
    return (data1,data2)


# launch one test, given a test path, returns output lines
# (output is first written to a file, than read)
def launch_one_test(tpath):
    #cmd = 'head -20 ' + tpath
    cmd = mainprog + " < '" + tpath + "'"
    (outdata,outerr) = command_with_output(cmd)
    # write to a tempfile, then read it
    # dumb, but easier to compare with a saved output file
    tempfname = "tempfileTODELETE.txt"
    tempf = open(tempfname,'w')
    tempf.write(outdata)
```

```
    tempf.write(outerr)
    tempf.close()
    outlines = read_file(tempfname)
    os.unlink(tempfname)
    return outlines



# read file given a path, return lines
def read_file(p):
    fIn = open(p,'r')
    res = fIn.readlines()
    fIn.close()
    return res


# compare two list of lines, returns true or false
def compare_2set_of_lines(lines1,lines2):
    if len(lines1) != len(lines2):
        return False
    for k in range(len(lines1)):
        if lines1[k] != lines2[k]:
            return False
    return True


# create_log_file, returns a path
# if path already exists, add something at the end
def create_log_file():
    res = "LOG_tests_"
    res += str(time.ctime()).replace(' ','_')
    res += '.log'
    res = os.path.abspath(os.path.join(logspath,res))
    if os.path.exists(res):
        counter = 1
        while os.path.exists(res):
            counter = counter + 1
            res = res[:-4] + '(' + str(counter) + ').log'
    return res


# add lines to a log path, can pass in one string or list of string
def add_to_log(logf,lines):
    # open log file, creates it if needed
    #if os.path.exists(logf):
    #    flog = open(logf,'w')
    #else:
    #    flog = open(logf,'a')
    flog = open(logf,'a')
    # if string
    if type(lines) == type(" "):
        flog.write(lines + '\n')
    else:
        for l in lines:
            flog.write(l + '\n')
    # close
```

```
    flog.close()


# help menu
def die_with_usage():
    print '*********************************************************'
    print 'Welcome to DruL test suite'
    print 'to launch test, type:'
    print '   LaunchTests.py -go'
    print ''
    print 'test files should end in: .drultest'
    print 'and corresponding outputs: .drultestout'
    print 'Of course, test names must match, like:'
    print "'testpattern1.drultest' and 'testpattern1.drultestout'"
    print '*********************************************************'
    sys.exit(0)


#*********************************************************
# MAIN

if __name__ == '__main__' :

    # launch help menu if needed
    if len(sys.argv) < 2 or sys.argv[1] != "-go":
        die_with_usage()


    # check if testing program exists and can be found
    if not os.path.exists(mainprog):
        print "you didn't install the main program, make drul"
        sys.exit(0)


    # grab all tests
    tests = list()
    os.path.walk(testspath,grab_tests,tests)

    # make sure all tests have an output
    make_sure_tests_have_outputs(tests)

    # make sure we found tests
    if len(tests) == 0:
        print "dummass, there's no tests"
        sys.exit(0)
    else :
        print 'launching',len(tests),'tests'

    # get logfile
    logfile = create_log_file()


    # launch every test
    counter = 0
    countpassed = 0
```

```
countfailed = 0
for t in tests:
    counter = counter + 1
    newout = launch_one_test(t)
    goodout = read_file(t + 'out')
    isOK = compare_2set_of_lines(newout,goodout)
    if isOK:
        countpassed = countpassed + 1
        add_to_log(logfile,str(counter) + ') test PASSED: '+t)
    else :
        countfailed += 1
        add_to_log(logfile,str(counter) + ') test FAILED: '+t)
        add_to_log(logfile,'****************************************')
        add_to_log(logfile,'*should be:*')
        add_to_log(logfile,goodout)
        add_to_log(logfile,'*and it is:*')
        add_to_log(logfile,newout)
        add_to_log(logfile,'****************************************')

# results
print 'passed',countpassed,'tests out of',counter
add_to_log(logfile,'########## SUMMARY:')
add_to_log(logfile,'passed '+str(countpassed)+' tests out of '+str(counter))
```

C.2.2 General test files

../TestSuite/Tests/assign1.drultest

```
a = 3;
p = pattern("01");
map (p) a;
print("bad");
```

../TestSuite/Tests/assign1.drultestout

```
Type error on line 4: we were expecting a mapper, name associated with something else
```

../TestSuite/Tests/assign2.drultest

```
p = pattern("10");
mapper concat (p) {}
print("bad");
```

../TestSuite/Tests/assign2.drultestout

```
Illegal assignment attempted on line 2: can't use keyword 'concat' as a mapper name
```

../TestSuite/Tests/assign3.drultest

```
p = pattern("10");
mapper rand (p) {}
print("bad");
```

../TestSuite/Tests/assign3.drultestout

```
Illegal assignment attempted on line 2: can't use keyword 'rand' as a mapper name
```

../TestSuite/Tests/assign4.drultest

```
p = pattern("10");
mapper slice (p) {}
print("bad");
```

../TestSuite/Tests/assign4.drultestout

```
Illegal assignment attempted on line 2: can't use keyword 'slice' as a mapper name
```

../TestSuite/Tests/assign5.drultest

```
p = pattern("10");
mapper pattern (p) {}
print("bad");
```

../TestSuite/Tests/assign5.drultestout

```
Illegal assignment attempted on line 2: can't use keyword 'pattern' as a mapper name
```

../TestSuite/Tests/beat_asPattern1.drultest

```
p1 = map (pattern("1111")) { return concat($1.asPattern(), pattern("0")); };
print(p1);

p2 = map (pattern("1010"), p1) { return $1.asPattern().repeat(3); };
print(p2);
```

../TestSuite/Tests/beat_asPattern1.drultestout

```
10101010
111000111000
```

../TestSuite/Tests/beat_note_rest.drultest

```
foo = pattern("1");
bar = pattern("10");
map (foo,bar) {
    if($1.note()) {print("$1 note");}
    else {print("$1 not note");}
    if($1.rest()) {print("$1 rest");}
    else {print("$1 not rest");}
    if($2.note()) {print("$2 note");}
    else {print("$2 not note");}
    if($2.rest()) {print("$2 rest");}
    else {print("$2 not rest");}
};
```

../TestSuite/Tests/beat_note_rest.drultestout

```
$1 note
$1 not rest
$2 note
$2 not rest
$1 not note
$1 not rest
$2 not note
$2 rest
```

../TestSuite/Tests/beat_simple_prevnext.drultest

```
a = pattern("1010");
map(a){
    print($1.prev(1));
    print($1.next(1));
};
```

../TestSuite/Tests/beat_simple_prevnext.drultestout

```
NULL
REST
NOTE
NOTE
REST
REST
NOTE
NULL
```

../TestSuite/Tests/beat_simple_yesno.drultest

```
a = pattern("101");
map (a) {print($1);};
```

../TestSuite/Tests/beat_simple_yesno.drultestout

```
NOTE
REST
NOTE
```

../TestSuite/Tests/beat_simple_yesnomaybe.drultest

```
a = pattern("1010");
b = pattern("001");
map (a, b){print($2);};
```

../TestSuite/Tests/beat_simple_yesnomaybe.drultestout

```
REST
REST
NOTE
NULL
```

../TestSuite/Tests/clip1.drultest

```
a = pattern("1111");
b = pattern("1");

instruments("fred","mabel");
c = clip(a,b);
print(c);

d = clip(
    "mabel" <- a,
    "fred"  <- b
);
print(d);
```

../TestSuite/Tests/clip1.drultestout

```
[
    fred:   1111
    mabel:  1
]
[
    fred:   1
    mabel:  1111
]
```

../TestSuite/Tests/clip2.drultest

```
instruments();
print(
    clip(
        pattern("1010")
    )
);
```

../TestSuite/Tests/clip2.drultestout

```
[
    hh_c:   1010
    sd_ac:
    bd:
    cowbell:
]
```

../TestSuite/Tests/clip3.drultest

```
clip = 1;
print("hey, I could assign something to clip, hummmm");
```

../TestSuite/Tests/clip3.drultestout

```
Illegal assignment attempted on line 2: can't use keyword 'clip' as a variable
```

../TestSuite/Tests/clip4.drultest

```
instruments();

clip("a" <- pattern("01010"));

print("bad...");
```

../TestSuite/Tests/clip4.drultestout

```
Incorrect function arguments on line 4: unknown instrument name 'a'
```

../TestSuite/Tests/concat1.drultest

```
p1 = pattern("1");
p2 = concat(p1);
print(p2); // should print 1
```

../TestSuite/Tests/concat1.drultestout

```
1
```

../TestSuite/Tests/concat2.drultest

```
p1 = pattern("1");
p2 = pattern("0");
p3 = concat(p1 , p2);
print(p3); // should print 10
```

../TestSuite/Tests/concat2.drultestout

```
10
```

../TestSuite/Tests/concat3.drultest

```
p1 = pattern("1");
p2 = pattern("0");
p3 = concat( p1 , pattern("") , p2);
print(p3); // should print 10
```

../TestSuite/Tests/concat3.drultestout

```
10
```

../TestSuite/Tests/concat4.drultest

```
p1 = concat(pattern());
print(p1); // should get ""
```

../TestSuite/Tests/concat4.drultestout

../TestSuite/Tests/concat5.drultest

```
p = concat(pattern(),pattern("10"),concat(pattern("0"),pattern("1")));
print (p); // should get 1001
```

../TestSuite/Tests/concat5.drultestout

```
1001
```

../TestSuite/Tests/concat6.drultest

```
print (concat() ); // should print ""
```

../TestSuite/Tests/concat6.drultestout

../TestSuite/Tests/dividebyzero.drultest

```
1/0;
```

../TestSuite/Tests/dividebyzero.drultestout

```
Division by zero attempted on line 1: Divisor evaluates to 0
```

../TestSuite/Tests/easycomparisons.drultest

```
print(1 < 2);
print(1 > 2);
print(1 == 2);
print(1 != 2);
print(1 == 2 || 1 <= 2);
print(42 >= 0);
```

../TestSuite/Tests/easycomparisons.drultestout

```
TRUE
FALSE
FALSE
TRUE
TRUE
TRUE
```

../TestSuite/Tests/falseassign.drultest

```
false = 4;
```

../TestSuite/Tests/falseassign.drultestout

```
Syntax error on line 1 between characters 0 and 5
```

../TestSuite/Tests/gcd.drultest

```
p1 = pattern("1").repeat(352);
p2 = pattern("1").repeat(40);

mapper subtract(a, b) {
    if( (a.note() || a.rest()) && (b.note() || b.rest() ) ) {
        return pattern("");
    } elseif (a.note() || a.rest() ) {
        return pattern("1");
    } else {
        return pattern("0");
    }
}

mapper squishrests(a) {
    if ( a.note() ) {
        return pattern("1");
    }
    else {
        return pattern("");
    }
}

mapper gcd(a, b) {
    if (    !a.prev(1).note() && !a.prev(1).rest()
         && !b.prev(1).note() && !b.prev(1).rest() ) {
        tmp = map (p1,p2) subtract;
        print(tmp.length());
        if ( tmp.length() == 0 ) {
            //print("length is 0!");
             print("in return spot");
             return p1;
        } elseif ( ( map(tmp) squishrests).length() > 0) {
            print("a gt b");
            p1 = tmp;
        } else {
            print("b gt a");
            p2 = tmp;
        }
        return map(p1,p2) gcd;
    }
    return pattern("");
}
p3 = map(p1,p2) gcd;

print(p3.length());
```

../TestSuite/Tests/gcd.drultestout

```
312
a gt b
272
a gt b
232
a gt b
192
a gt b
152
a gt b
112
a gt b
72
a gt b
32
a gt b
8
b gt a
24
a gt b
16
a gt b
8
a gt b
0
in return spot
8
```

../TestSuite/Tests/helloworld.drultest

```
print("hello world");
```

../TestSuite/Tests/helloworld.drultestout

```
hello world
```

../TestSuite/Tests/if-elseif-else.drultest

```
if (true) {print("yes");} else {print("no");}
//yes
if(false) { print("nope") ; } print("got here");
// got here

if(false) {
```

```
    print("death everywhere");
} elseif (true) {
    print("got it!");
} else {
    print("noooo!");
}
```

../TestSuite/Tests/if-elseif-else.drultestout

```
yes
got here
got it!
```

../TestSuite/Tests/instrum1.drultest

```
instruments("allo","everyone");
print("done"); // should return done
```

../TestSuite/Tests/instrum1.drultestout

```
done
```

../TestSuite/Tests/instrum2.drultest

```
instruments(); // should print error

print("done");
```

../TestSuite/Tests/instrum2.drultestout

```
done
```

../TestSuite/Tests/instrum3.drultest

```
instruments("thierry","rocks");

a = 3 * 2;

instruments("always!");

print("should fail!!!!");
```

../TestSuite/Tests/instrum3.drultestout

```
Instrument redefinition attempted on line 6: don't do that
```

../TestSuite/Tests/instrum4.drultest

```
instruments = 4;
print("shouldn't be able to assign something to 'instruments'");
```

../TestSuite/Tests/instrum4.drultestout

```
Syntax error on line 1 after character 0
```

../TestSuite/Tests/instrum5.drultest

```
p = map (pattern("1")) {instruments("a");};
print("bad");
```

../TestSuite/Tests/instrum5.drultestout

```
Illegal assignment attempted on line 1: can't define instruments inside mappers
```

../TestSuite/Tests/map_alias.drultest

```
mapper foo (a) { return pattern("0"); }
bar = foo;
baz = pattern("111");
print(map (baz) foo);
print(map (baz) bar);
```

../TestSuite/Tests/map_alias.drultestout

```
Illegal assignment attempted on line 2: can't assign a mapper
```

../TestSuite/Tests/mapper_bad_return1.drultest

```
mapper fred (a, b, c) {
    if (a.note() ) {
        return true;
    } else {
        return b;
    }
}

map (pattern("101"), pattern(""), pattern("10101") ) fred;
```

../TestSuite/Tests/mapper_bad_return1.drultestout

```
Illegal assignment attempted on line 2: attempt to return an illegal value from this mapper
```

../TestSuite/Tests/mapper_bad_return2.drultest

```
mapper fred (a, b, c) {
    ; // pathology forever!
    if (a.note() ) {
        return true;
    } else {
        return b;
    }
}

map (pattern("101"), pattern(""), pattern("10101") ) fred;
```

../TestSuite/Tests/mapper_bad_return2.drultestout

```
Illegal assignment attempted on line 2: attempt to return an illegal value from this mapper
```

../TestSuite/Tests/mapper_empty.drultest

```
map( pattern("1010") ) {print("beat");};
map(pattern(""), pattern("10101") ) { print("counting to five");};
```

../TestSuite/Tests/mapper_empty.drultestout

```
beat
beat
beat
beat
counting to five
counting to five
counting to five
counting to five
counting to five
```

../TestSuite/Tests/mapper_nobeats.drultest

```
print( map (pattern("0000")) { return pattern("10"); } );
```

../TestSuite/Tests/mapper_nobeats.drultestout

```
10101010
```

../TestSuite/Tests/mapper_read_outer_scope.drultest

```
a = pattern("1001");
b = pattern("10");
map (b) {print(a);};
```

../TestSuite/Tests/mapper_read_outer_scope.drultestout

```
1001
1001
```

../TestSuite/Tests/mapper_return_beat.drultest

```
print( map (pattern("1010"), pattern("1101")) {
 if($1.note() ) { return $2; }
});
```

../TestSuite/Tests/mapper_return_beat.drultestout

```
10
```

../TestSuite/Tests/output1.drultest

```
instruments();

c = clip( pattern("1010001") , pattern() , pattern("000") );

c.outputText("file.txt");

print("done");
```

../TestSuite/Tests/output1.drultestout

```
done
```

../TestSuite/Tests/parse_error_1.drultest

```
print(a);
print(b)
print(c)
```

../TestSuite/Tests/parse_error_1.drultestout

```
Syntax error on line 2 between characters 0 and 8
```

../TestSuite/Tests/parse_error_2.drultest

```
foo bar baz // not so hot, this syntax
```

../TestSuite/Tests/parse_error_2.drultestout

```
Syntax error on line 1 after character 0
```

../TestSuite/Tests/parse_error_3.drultest

```
// this is to show that we can have initial errors *after*
// some comments
+
= a
;
```

../TestSuite/Tests/parse_error_3.drultestout

```
Syntax error on line 1 after character 0
```

../TestSuite/Tests/parse_error_4.drultest

```
// this is to show that we can have initial errors *after*
// some comments
foo();
bar();
+
= a
;
```

../TestSuite/Tests/parse_error_4.drultestout

```
Syntax error between char 0 of line 1 and char 6 of line 4
```

../TestSuite/Tests/pattern1.drultest

```
a = pattern("101");
print(a);
```

../TestSuite/Tests/pattern1.drultestout

```
101
```

../TestSuite/Tests/pattern10.drultest

```
p0 = map(pattern("1101"))
{
    if ($1.note() && $1.next(1).note()) { return pattern(""); }
    elseif ($1. note())                 { return pattern("1"); }
    else                                { return pattern("0"); }
};

print(p0); // should be 101

p1 = map( map(pattern("1101"))
          {
            if ($1.note() && $1.next(1).note()) { return pattern(""); }
            elseif ($1. note())                 { return pattern("1"); }
            else                                { return pattern("0"); }
          })
{
    if ($1.note()) { return pattern("1"); }
        else { return pattern(); }
};

print(p1); // should return 11
```

../TestSuite/Tests/pattern10.drultestout

```
101
11
```

../TestSuite/Tests/pattern11.drultest

```
mapper mymapper (p)
{
    if (p.note()) { return pattern("11"); }
    else        { return pattern("0"); }
};
p1 = pattern("010");
p2 = map (p1) mymapper;
print (p2) ; // should be 0110
```

../TestSuite/Tests/pattern11.drultestout

```
0110
```

../TestSuite/Tests/pattern12.drultest

```
p11 = map(pattern("1111"))
{
 if ($1.note() && $1.next(1).note() && $1.next(2).note() ) { return pattern("1"); }
 else { return pattern("0"); }
};
print(p11); // should return 1100
```

../TestSuite/Tests/pattern12.drultestout

```
1100
```

../TestSuite/Tests/pattern13.drultest

```
p11 = map(pattern("10101"))
{
 if ($1.prev(1).note() && $1.next(1).note()) { return pattern("1"); }
 else { return pattern("0"); }
};

print(p11); // should return 01010
```

../TestSuite/Tests/pattern13.drultestout

```
01010
```

../TestSuite/Tests/pattern14.drultest

```
p0 = map(pattern("110110110"))
{
 if ($1.prev(1).note() || $1.next(1).note()) { return pattern("1"); }
 else { return pattern("0"); }
};

print(p0); // should return 111111111
```

../TestSuite/Tests/pattern14.drultestout

```
111111111
```

../TestSuite/Tests/pattern15.drultest

```
p0 = map(pattern("001") , pattern("111") )
{
 if ($1.note() && $2.note()) { return pattern("1"); }
 else { return pattern("0"); }
};

print(p0); // should return 001
```

../TestSuite/Tests/pattern15.drultestout

```
001
```

../TestSuite/Tests/pattern16.drultest

```
p0 = map(pattern("111") , pattern("1111"))

{
  if ($1.note() && $2.note()) { return pattern("1"); }
  else                        { return pattern("0"); }
};

print(p0); // should return 1110
```

../TestSuite/Tests/pattern16.drultestout

```
1110
```

../TestSuite/Tests/pattern17.drultest

```
p0 = map(pattern("010101") , pattern("111000") , pattern("000001") )

{
  if (($1.note() || $2.note()) && $3.rest() ) { return pattern("1"); }
  else                                        { return pattern("0"); }
};

print(p0); // should return 111100
```

../TestSuite/Tests/pattern17.drultestout

```
111100
```

../TestSuite/Tests/pattern18.drultest

```
p0 = map(pattern("010101") , pattern() )

{
  if ($1.note() || 1==1 ) { return pattern("1"); }
  else                    { return pattern("0"); }
};

print(p0); // should return 111111
```

../TestSuite/Tests/pattern18.drultestout

```
111111
```

../TestSuite/Tests/pattern19.drultest

```
// takes every even index, starting at 0

pat = pattern("00101110100010"); // even indexes: 0111101

helper = pattern("10").repeat( pat.length() / 2 );

p0 = map( pat , helper)
{
   if ( $2.note() )
```

```
    {
    if ( $1.note() ) { return pattern("1"); }
        else             { return pattern("0"); }
    }
   else   { return pattern(""); }
};

print(p0); // should return 0111101
```

../TestSuite/Tests/pattern19.drultestout

```
0111101
```

../TestSuite/Tests/pattern2.drultest

```
p1 = pattern("0101");
p2 = p1.repeat(3);
print(p2);
```

../TestSuite/Tests/pattern2.drultestout

```
010101010101
```

../TestSuite/Tests/pattern20.drultest

```
// copy

p0 = map( pattern("000111010101") )
{
    if ( $1.note() ) { return pattern("1"); }
        else             { return pattern("0"); }
};

print(p0); // should return 000111010101
```

../TestSuite/Tests/pattern20.drultestout

```
000111010101
```

../TestSuite/Tests/pattern21.drultest

```
pattern = 3;
print("just assigned something to 'pattern'");
```

../TestSuite/Tests/pattern21.drultestout

```
Illegal assignment attempted on line 1: can't use keyword 'pattern' as a variable
```

../TestSuite/Tests/pattern22.drultest

```
a = pattern("31");
print("bad");
```

../TestSuite/Tests/pattern22.drultestout

```
Invalid pattern string on line 3: Patterns definitions must be a string of 0's and 1's
```

../TestSuite/Tests/pattern3.drultest

```
p1 = pattern("001");
p2 = pattern("111");
p3 = pattern("101");

p4 = concat(p2,p3,p1);
print(p4);
```

../TestSuite/Tests/pattern3.drultestout

```
111101001
```

../TestSuite/Tests/pattern4.drultest

```
p1 = pattern();
print(p1);
p2 = pattern("");
print(p2);
print("end");
```

../TestSuite/Tests/pattern4.drultestout

```
end
```

../TestSuite/Tests/pattern5.drultest

```
p1 = concat( pattern("01") , pattern("10") , pattern() , pattern("") );
print ( p1 );
```

../TestSuite/Tests/pattern5.drultestout

```
0110
```

../TestSuite/Tests/pattern6.drultest

```
p1 = pattern("01110").repeat(5);
a = p1.length();
print(a);
```

../TestSuite/Tests/pattern6.drultestout

```
25
```

../TestSuite/Tests/pattern7.drultest

```
p1 = pattern("101");
p2 = map (p1)
{
    if ($1.note()) { return pattern("11"); }
    else           { return pattern("0"); }
};
print(p2);
```

../TestSuite/Tests/pattern7.drultestout

```
11011
```

../TestSuite/Tests/pattern8.drultest

```
p1 = pattern("1110111");
p2 = map(p1)
{
    if ($1.note()) { return pattern(""); }
    else           { return pattern("1"); }
};
print(p2);
```

../TestSuite/Tests/pattern8.drultestout

```
1
```

../TestSuite/Tests/pattern9.drultest

```
p9 = map(pattern("1101"))
{
    if ($1.note() && $1.next(1).note()) { return pattern("1"); }
    else                                { return pattern("0"); }
};
print(p9);
```

../TestSuite/Tests/pattern9.drultestout

```
1000
```

../TestSuite/Tests/pattern_reverse1.drultest

```
p1 = pattern("010101");
p2 = pattern("101010");

print(concat(p2.reverse(),p1.reverse()));
```

../TestSuite/Tests/pattern_reverse1.drultestout

```
010101101010
```

../TestSuite/Tests/print.drultest

```
print("thierry");
print (    "rulzzzzz!");
print("!@#$%^&*()__)*&%^_+HSVWUO@_@");
print("//");
print(123456);
print(true); print(false);
```

../TestSuite/Tests/print.drultestout

```
thierry
rulzzzzz!
!@#$%^&*()__)*&%^_+HSVWUO@_@
//
123456
TRUE
FALSE
```

../TestSuite/Tests/print_stringescapes.drultest

```
print("hello /\\ ~hello |- NIL");
print("I'm really \"excited\" about this test...");
```

../TestSuite/Tests/print_stringescapes.drultestout

```
hello /\ ~hello |- NIL
I'm really "excited" about this test...
```

../TestSuite/Tests/rand1.drultest

```
r = rand();

if (0 <= r && r <= 1) { print("It works!"); }
else                  { print("What the hell?"); }
```

../TestSuite/Tests/rand1.drultestout

```
It works!
```

../TestSuite/Tests/rand2.drultest

```
r = rand(4);

if (0 <= r && r <= 3) { print("It works!"); }
else                  { print("What the hell?"); }
```

../TestSuite/Tests/rand2.drultestout

```
It works!
```

../TestSuite/Tests/rand3.drultest

```
rand = 4;
print("assigned something to 'rand'");
```

../TestSuite/Tests/rand3.drultestout

```
Illegal assignment attempted on line 1: can't use keyword 'rand' as a variable
```

../TestSuite/Tests/return1.drultest

```
p = pattern("111");

p2 = map (p)
{
   return $1.next(1);
};

print(p2);
```

../TestSuite/Tests/return1.drultestout

```
11
```

../TestSuite/Tests/return2.drultest

```
p = pattern("111");

p2 = map (p)
{
   return $1.prev(1);
};

print(p2);
```

../TestSuite/Tests/return2.drultestout

```
11
```

../TestSuite/Tests/slice1.drultest

```
p3 = pattern("0011100");
print(p3.slice(3, 3));
```

../TestSuite/Tests/slice1.drultestout

```
111
```

../TestSuite/Tests/slice2.drultest

```
p3 = pattern("0011100");
print(p3.slice(1, 3));
```

../TestSuite/Tests/slice2.drultestout

```
001
```

../TestSuite/Tests/slice3.drultest

```
p3 = pattern("0011100");
print(p3.slice(5, 3));
```

../TestSuite/Tests/slice3.drultestout

```
100
```

../TestSuite/Tests/trueassign.drultest

```
true = 3;
```

../TestSuite/Tests/trueassign.drultestout

```
Syntax error on line 1 between characters 0 and 4
```

../TestSuite/Tests/truthtable.drultest

```
print(true && true);
print(true && false);
print(false || true);
print(true || false);
```

../TestSuite/Tests/truthtable.drultestout

```
TRUE
FALSE
TRUE
TRUE
```

../TestSuite/Tests/unaryops.drultest

```
print(-3);
print(!true);
```

../TestSuite/Tests/unaryops.drultestout

```
-3
FALSE
```

../TestSuite/Tests/variable_readwrite.drultest

```
a = 42;
print(a);
```

../TestSuite/Tests/variable_readwrite.drultestout

```
42
```

C.2.3 LaunchTestsParser.py

```
#! /usr/bin/env python
"""
DruL team, Columbia (2008) PLT class
copyright DruL team

contact: tb2332@columbia.edu

name: LaunchTests.py
language: python
programer: Thierry Bertin-Mahieux

main program of the test suite, launch all tests that it can find.
"""
```

```
import os
import sys
import glob
import time
import tempfile



drulpath = "../"
testspath = "./ParserTests/"
logspath = "./LOGS/"
testingprog = "../Parser/testing"     #actual program to test stuff


# returns a list of file in current dir
# to use with os.walk
def grab_tests(arg=list(),path="",names=""):
    tests = glob.glob(os.path.join(os.path.abspath(path),'*.drultest'))
    for t in tests:
        arg.append(t)
    return arg


# launch any command, return outputs (stdin and stderr)
def command_with_output(cmd):
    if not type(cmd) == unicode :
        cmd = unicode(cmd,'utf-8')
    #should this be a part of slashify or command_with_output?
    #if sys.platform=='darwin' :
    #    cmd = unicodedata.normalize('NFC',cmd)

    (child_stdin,child_stdout,child_stderr) = os.popen3(cmd.encode('utf-8'))
    data1 = child_stdout.read()
    data2 = child_stderr.read()
    child_stdout.close()
    child_stderr.close()
    return (data1,data2)


# launch one test, given a test path, returns stdout or stderr
# (output is first written to a file, than read)
def launch_one_test(tpath):
    cmd = testingprog + " < '" + tpath + "'"
    (outdata,outerr) = command_with_output(cmd)
    return (outdata,outerr)


# read file given a path, return lines
def read_file(p):
    fIn = open(p,'r')
    res = fIn.readlines()
    fIn.close()
    return res
```

```
# compare two list of lines, returns true or false
def check_output(lines):
    if lines == "":
        return True
    if lines.count("Fatal error:") > 0 :
        return False
    return True


# create_log_file, returns a path
# if path already exists, add something at the end
def create_log_file():
    res = "LOG_parsertests_"
    res += str(time.ctime()).replace(' ','_')
    res += '.log'
    res = os.path.abspath(os.path.join(logspath,res))
    if os.path.exists(res):
        counter = 1
        while os.path.exists(res):
            counter = counter + 1
            res = res[:-4] + '(' + str(counter) + ').log'
    return res


# add lines to a log path, can pass in one string or list of string
def add_to_log(logf,lines):
    flog = open(logf,'a')
    # if string
    if type(lines) == type(" "):
        flog.write(lines + '\n')
    else:
        for l in lines:
            flog.write(l + '\n')
    # close
    flog.close()



# help menu
def die_with_usage():
    print '*********************************************************'
    print 'Welcome to DruL test suite'
    print 'to launch test, type:'
    print '   LaunchTests.py -go'
    print ''
    print 'test files should end in: .drultest'
    print 'and corresponding outputs: .drultestout'
    print 'Of course, test names must match, like:'
    print "'testpattern1.drultest' and 'testpattern1.drultestout'"
    print '*********************************************************'
    sys.exit(0)


#*********************************************************
# MAIN
```

```
if __name__ == '__main__' :

    # launch help menu if needed
    if len(sys.argv) < 2 or sys.argv[1] != "-go":
        die_with_usage()


    # check if testing program exists and can be found
    if not os.path.exists(testingprog):
        print "you didn't install the testing program, make testing"
        sys.exit(0)


    # grab all tests
    tests = list()
    os.path.walk(testspath,grab_tests,tests)

    # make sure we found tests
    if len(tests) == 0:
        print "dummass, there's no tests"
        sys.exit(0)
    else :
        print 'launching',len(tests),'tests'

    # get logfile
    logfile = create_log_file()


    # launch every test
    counter = 0
    countpassed = 0
    countfailed = 0
    for t in tests:
        counter = counter + 1
        (out,outerr) = launch_one_test(t)
        isOK = check_output(outerr)
        if isOK:
            countpassed = countpassed + 1
            add_to_log(logfile,str(counter) + ') test PASSED: '+t)
        else :
            countfailed += 1
            add_to_log(logfile,str(counter) + ') test FAILED: '+t)
            add_to_log(logfile,'****************************************')
            add_to_log(logfile,'last lines:')
            if len(out) < 100 :
                add_to_log(logfile,out)
                add_to_log(logfile,outerr)
            else :
                add_to_log(logfile,out[-100:])
                add_to_log(logfile,outerr)
            add_to_log(logfile,'****************************************')

    # results
    print 'passed',countpassed,'tests out of',counter
    add_to_log(logfile,' ')
```

```
    add_to_log(logfile,'########## SUMMARY:')
    add_to_log(logfile,'passed '+str(countpassed)+' tests out of '+str(counter))
```

C.2.4 Parser test files

../TestSuite/ParserTests/comparisons.drultest

```
a = 1;
b = 2;
a < b;
a <= b;
a > b;
a >= b;
a != b;
a == b;
a > b > a;
(a <= b);
a == b != a > b < a >= a <= b;
```

../TestSuite/ParserTests/complexmap1.drultest

```
map (hi , you)
{
    $1.note();
    $2.rest();
    a = pattern("01");
    if ($1.rest()) { return pattern(""); }
    elseif ($2.note()) {return a;}
    else { return a.repeat(2); }
};
```

../TestSuite/ParserTests/concat.drultest

```
p1 = pattern("01");
p2 = concat(p1 , p1);
p3 = concat(p1,p2,p1);
```

../TestSuite/ParserTests/dollarsign.drultest

```
p_new_rev = map (p_new)
{
    $1.rest();
};


map (hi , you)
{
    $1.note();
    $2.rest();
};
```

../TestSuite/ParserTests/if1.drultest

```
if (1 == 2)
    {a = 3;}

if (2 == 2) {}

if (4 == 4)
{
    "allo";
   b = 1;
}
```

../TestSuite/ParserTests/if2.drultest

```
if (false && true) {pattern("01");}

elseif ( pattern("01") == pattern("001") )
    { if ( 3 != 2 ) {print("allo");}
        }

elseif (true || false || (pattern("0101").repeat(4) >= pattern("0101") ))
{ print ("yo!!!!!!!!!!");}

else { a =2;;;;;}
```

../TestSuite/ParserTests/ifbare.drultest

```
if (1 > foo) { bar; }
1;
```

../TestSuite/ParserTests/ifelse1.drultest

```
a = 1;
if (a == 1) {b = 3;}
else {b = 4;}
```

../TestSuite/ParserTests/ifelseif.drultest

```
if (1 > 3) { foo; } elseif (1 < 3) {bar ;}
```

../TestSuite/ParserTests/ifelseifelse.drultest

```
if(foo) { 1; } elseif (bar) { 2; } else {3;}
```

../TestSuite/ParserTests/instrument.drultest

```
instruments (yo, man ,whats ,up);

intruments(can, we, set, more, complex ,        things);

intruments   (  whats, up )     ;
```

../TestSuite/ParserTests/logicalAND.drultest

```
a = 1;
b = 2;
a && b;
true && true;
false && false;
true && false;
false && true;
true && false && true;
false && false && true && true;
(false && true) && ((false && false) && true);
```

../TestSuite/ParserTests/logicalOR.drultest

```
a = 1;
b = 2;
a || b;
true || true;
false || false;
true || false;
false || true;
true || false || true;
false || false || true || true;
(false || true) || ((false || false) || true);
```

../TestSuite/ParserTests/logicalORAND.drultest

```
a = 1;
b = 2;
(false || true && false);
(true && false || true);
(a || b && 3 || false && true);
(true || false) && ((false && true || true) || true);
```

../TestSuite/ParserTests/mapper.drultest

```
mapper mymapper (p)
{
    return pattern("1");
}

p = pattern("01");
p2 = map (p) mymapper;


mapper mymapper2 (bla)
{
    a = 3;
    b = 4;
    res = pattern ("010101");
    return res;
}
```

../TestSuite/ParserTests/mappercall.drultest

```
map(a ,b ,c ,-3 ) //that will be a problem, that will...
{a + 3; b+ 15; "foo";}
;
```

../TestSuite/ParserTests/noendline.drultest

```
//
```

../TestSuite/ParserTests/patternrepeat.drultest

```
a = pattern("001");

b = pattern("01").repeat(4);
d = a.repeat(3);
```

../TestSuite/ParserTests/pnote.drultest

```
// are there other use possibles?
// is p.note() or p.note ? samething for rest

p.note();
p.rest();
```

../TestSuite/ParserTests/print.drultest

```
print ("1");
print(         "allo");
print ("yo!3748473222937`1-232-/._(*&^%$#@");

print(pattern(""));
print( pattern ("010111001"));

a = pattern("11110");
print (a);
b = 3;
print( b );

c = clip(a);
print (c);
```

../TestSuite/ParserTests/rand.drultest

```
// not sure of the syntax, no examples in the Reference Manual

rand();

a = rand(1);
```

../TestSuite/ParserTests/refmanexamplecode.drultest

```
// copied/paste from the RefManual, current version on 11/19/2008

//This code manipulates some patterns, associate them to instruments and
//sends them to outputs.
//First the Instrument definition. It has to be done before
//any clips are created, otherwise there will be an error.

instruments(hihat ,bassdrum ,crash ,snare); //define four instruments

//Integer variables used as tempos for clips.

a = 350;
b = 300;

//Patterns.

p1 = pattern("100100100");
p2 = pattern("");//empty pattern
p3 = pattern("0");//pattern with only one  rest  in it.
p4 = pattern("1");//pattern with only one  note  in it.

//p_concat is essentially concatenation of three patterns.

p_concat = concat(p1, pattern("11110000"), pattern("00011"));

//Make a new pattern using above patterns and
//the library methods  repeat  and  slice .

p_custom = concat( p2, p3.repeat(2), p4.repeat(3),
    p3.repeat(2) ,p4.repeat(4), p_concat );
p_custom_new = concat(p_custom ,p3.repeat(2) ,p_concat ,p4.repeat(3));
p_new = concat( p_custom_new.slice(4,10),
    p_concat.slice(5,p1.length()), p3.repeat(7) );

//Now some complex pattern manipulation.
//New Patterns.

alternate_beats = pattern("10").repeat(8);
P_concat_new = concat(p_concat , p_custom);

//Anonymous mapping.
```

```
p_new_rev = map (p_new)
{
    if ($1.rest()) { pattern("1"); }
    else           { pattern("0"); }
};

//Mapper definitions.

mapper newMapper1 (p_any)
{
    if (p_any.note())   { return pattern("1"); }
    else                { return pattern(""); }
}

mapper newMapper2 (p_any ,alternate_beats)
{
    if (alternate_beats.rest()) { return pattern("");} //pattern of length 0
    elseif (p_any.note())       { return pattern("1");}
    else                        { return pattern("0");}
}

mapper improved_newMapper2(p_any, alternate_beats)
{
    if (alternate_beats.rest())   { return pattern("");  }
    elseif (p_any.note())         { return pattern("1"); }
    elseif (p_any.next(1).note()) { return pattern("1"); }
    else                          { return pattern("0"); }
}
p_custom_new_notes   = map (p_custom_new) myMapper1;
p_concat_new_downbeats = map (p_concat_new) newMapper2;

//print out the created patterns to Standard Output.

print("Output from Sample DruL Code:");
print(p_concat);
print(p_custom);
print(p_custom_new);
print(p_new);
print(p_new_rev);
print(p_custom_new_notes);
print(p_concat_new_downbeats);
print("END OF OUTPUT");

//Pattern associations using clips.

// CLIP SYNTAX HAS TO BE REDEFINED
clip_complete = clip
(
 hihat    <- p_concat_new_downbeats,
 bassdrum <- p_custom_new_notes,
 crash    <- p_new_rev,
 snare    <- p_new
);

//output clip as a midi file
```

```
out.midi("out_file1.midi",clip_complete,a);//a = tempo (Beats per minute)

// Last instrument has an empty beat-pattern.

clip_partial = clip(p_concat ,p_custom_new ,p_custom);

//output clip as a midi file

out.midi("out_file2.midi",clip_partial,b);//b = tempo
```

../TestSuite/ParserTests/simpleint.drultest

```
a;


b;
a;c;
d=1;
d = 3;
a       =     d;
```

../TestSuite/ParserTests/simplepattern.drultest

```
a = pattern("01");
b = pattern("");
c =pattern(    "01010001010101001010101001010101001"          );
```

../TestSuite/ParserTests/simplestring.drultest

```
"allo";
"  yo ";
"drul rocks!";
"17681217298190@#$%^&*()_#";
   "//";
"";

a = "01010101"; // may be bad
b = "";         // may be bad
 c = a + b;       // may be bad
```

../TestSuite/ParserTests/stdC.drultest

```
a = 1;
b = 2;
c = 3;
d = a * b;
d = a * b * c;
e = 1 * 3;
f = c / b;
f = c / b / a;
g = 4 /2;
g = 12 / 24;
h = a % b;
h = a % b % c;
i = 3 % 14;
(3 % 4 / 5) * ((a * 2 / h) % ((9 / 3) * (14 *5)));
```