`Clef` Manual

Eric Blond

7th of November 2009

1 Introduction

I've been thinking for quite a long time on what an ideal computer language should have. So it doesn't come as a surprise that I learnt quite a few languages: `C`, `C++`, `C#`, `Java`, `OCaml`, `Haskell`, `Python`, `Prolog`, `SQL` to name the most popular[1].

A previous exposure to *Computer Algebra Systems* led me to think that being able to represent mathematics within the language was a must be. But such a feature is hard to reach since it requires the language to have serious introspective capabilities.

This manual is not a formal description of a language that's meant to evolve a lot. It can eventually not be backward compatible if I feel that a rough restructuration is necessary for a better language. But don't be afraid: if I went as far as writing a manual for the very first version of this language, it means that I reached some solid conclusions about its grammar and semantic.

2 Background

This language deals with *relations* in a mathematical sense. More formally, let's define the alphabet A as the set of all `Unicode` characters. Over that set, we can define strings set $A^* = \bigcup_{k\in\mathbb{N}} A^k$.

Now given an arity k, we can define the set of relations of arity k to be $\mathcal{R}_k = (A^*)^k$. We can also talk about the set of all relations $\mathcal{R} = \bigcup_{k\in\mathbb{N}} \mathcal{R}_k$.

Now everything we can do with `Clef` is:

- define relations, most likely in a recursive manner
- query relations, i.e. get all the tuples a particular relations contains

3 Tutorial

3.1 Overview

First we need to define a set of relations in a file. Let's suppose we have a file named `database.txt` in the current directory that looks like this:

[1] ... it came as far as `SKI` combinators

```
less-than[one, two]
less-than[two, three]
less-than[three, four]

before[X, Y]
less-than[X, Y]

before[X, Y]
less-than[X, Z]
before[Z, Y]
```

You can then start to use the interpreter like this:

```
eric@samsung:~/Documents/Clef$ python -m clef.main database.txt
Clef v0.1 (build 2009-11-04) by Eric Blond
Loading database...
... loaded 3 declarations
>less-than[X, Y]
+---+-------+-------+
| # |     X |     Y |
+---+-------+-------+
| 1 |   one |   two |
| 2 | three |  four |
| 3 |   two | three |
+---+-------+-------+
> less-than[one, X]
+---+-----+
| # |   X |
+---+-----+
| 1 | two |
+---+-----+
> before[one, X]
+---+-------+
| # |     X |
+---+-------+
| 1 |  four |
| 2 | three |
| 3 |   two |
+---+-------+
> less-than[X, two]
+---+-----+
| # |   X |
+---+-----+
| 1 | one |
+---+-----+
>
Bye!
eric@samsung:~/Documents/Clef$
```

You end a session by just typing `Enter` in an empty line.
You've probably guessed so far that:

- the `database.txt` contains a definition of relations `less-than` and `before`
- during the session, we made some queries involving the first relation, and the result we got were tables

3.2 Database

Now let's have a look at how we can set relations in the database file.

We can actually make relations between *labels*. At the moment, we don't support relations between arbitrary strings but only those between labels which are similar to what we call *identifiers* in more classical languages.

We can define simple relations by just laying down all cases one by one, as in the definitions of `less-than`. As an example, let's see the first simple declaration:

```
less-than[one, two]
```

This declaration mathematically means that $(\texttt{one}, \texttt{two}) \in \texttt{less-than}_2$.

Note that you're free to define relations with different arities and names.

You can also define complex relations that involves so called *conditions*, as you can see in the definition of `before`: declarations like that begin with a *pattern*, and are followed by conditions which are like patterns, but start with spaces/tabulations. Again, let's see as an example the last declaration of that kind:

```
before[X, Y]
less-than[X, Z]
before[Z, Y]
```

One important thing to understand that declaration is that uppercased identifiers like `X` are not labels but *variables* that are related to each other within a declaration (but variables of same name in two different declarations are completely independant).

So what this declaration means in terms of relations is $\forall (x, y) \in (A^*)^2 (\exists z \in A^* (x, z) \in \texttt{less-than}_2 \wedge (z, y) \in \texttt{before}_2) \Rightarrow (x, y) \in \texttt{before}_2$.

Finally, please note that all relations are initially defined as $\emptyset$. So undefined relations can perfectly be queried (try this in the interpreter).

3.3 Queries

Each query must be in a pattern-like form, like the one you can see in the sample session:

```
> less-than[X, Y]
> less-than[one, X]
> less-than[X, two]
```

Each result is a table whose columns are associated to the variables of the pattern. Note that each tuple is only showed once, and that they're order in the lexicographical order.

Finally, let me explain a tricky case we can see when we make a query that involves no variables at all:

```
> before[one, two]
+---+
| # |
+---+
| 1 |
+---+
> before[one, one]
+---+
| # |
+---+
```

In the first case, there is a solution as $(\texttt{one}, \texttt{two}) \in \texttt{before}_2$. So we see that there is one solution there. On the other hand, $(\texttt{one}, \texttt{one}) \notin \texttt{before}_2$, therefore the table doesn't have even one row.