Course C^{++}, Exercise List 7

Deadline: 28.04.2016

In this exercise list, we implement *matching*, using the trees of exercise list 5, and `std::vector`.

1. Write `bool operator == ( tree t1, tree t2 )`, that returns true iff the trees `t1` and `t2` are equal. One can write a recursive procedure, but we won't do that, because we want to use `std::vector`.

   We will use the following definition:

   $$\begin{array}{lcl} S \cup \{ f(t_1, \ldots, t_n) \} \equiv \{ g(u_1, \ldots, u_m) \} & \Rightarrow & \textbf{false} \text{ if } n \neq m, \text{ or } f \neq g, \\ S \cup \{ f(t_1, \ldots, t_n) \} \equiv \{ f(u_1, \ldots, u_n) \} & \Rightarrow & S \cup \{ t_1 \equiv u_1, \ldots, t_n \equiv u_n \}, \\ \emptyset & \Rightarrow & \textbf{true}. \end{array}$$

   If one wants to know whether trees t_1, t_2 are equal, one starts with $S = \{ t_1 \equiv t_2 \}$.

   Use `std::vector< std::pair< tree, tree>>` to implement the set of equations. Take the pair at the end, process it, and if necessary put new pairs back at the end.

2. Download **matching.h** from the course homepage.

   Complete `operator( )(const tree& t) const` in **struct matching**. It is very similar to the substitution function of List 5.

   Use of **replacesubtree**, in order to avoid unecessary copying. Make sure that the **non-const** versions of `functor( )` and `operator[]( )` are removed in **class tree**.

3. A tree t_1 can be *matched into a tree* t_2, if there exists a substitution Θ, s.t. $t_1\Theta = t_2$.

   Matching can be implemented recursively, but we interested in using `std::list`, or `std::vector`, so we (=you) will implement matching using the following definition:

   $$\begin{array}{lcl} (\ \Theta,\ S \cup \{\ f(t_1, \ldots, t_n)/g(u_1, \ldots, u_m)\ \}\ ) & \Rightarrow & \textbf{false} \text{ if } n \neq m, \text{ or } f \neq g, \\ (\ \Theta,\ S \cup \{\ f(t_1, \ldots, t_n)/f(u_1, \ldots, u_n)\ \}\ ) & \Rightarrow & (\ \Theta,\ S \cup \{ t_1/u_1, \ldots, t_n/u_n \}\ ), \\ (\ \Theta,\ S \cup \{ V/t \}\ ) & \Rightarrow & (\ \Theta \cup \{ V := t \},\ S\ ) \text{ if } V\Theta \text{ is undefined}, \\ (\ \Theta,\ S \cup \{ V/t \}\ ) & \Rightarrow & \textbf{false} \text{ if } V\Theta \text{ is defined, and } V\Theta \neq t, \\ (\ \Theta,\ S \cup \{ V/t \}\ ) & \Rightarrow & (\ \Theta,\ S\ ) \text{ if } V\Theta \text{ is defined, and } V\Theta = t. \end{array}$$

We assume that f, g are functors that are not variables. V is a tree without subterms with a functor that is a variable. In order to decide whether a functor is a variable, use `matching::isvariable( const std::string& s )`. (A functor is a variable if it starts with an underscore.)

t is an arbitrary tree.

States have form (Θ, S), where Θ is the current matching and S contains the pairs of trees to be matched. Matching starts with $(\emptyset, \{\text{from/into}\})$. Use a **matching** and `std::vector< std::pair<tree,tree>>`.

There is a complication that matching may fail. Since this is not exceptional, one should not throw an exception in this case. We solve the problem by returning a `std::list<matching>`, which is empty, when matching fails, and contains one matching, when matching succeeds.

Below are some examples:

```
f( _X, _Y )       into    f( f(a), f(b) )
     _X := f(a)     _Y := f(b)
f( _X, _X )       into    f( f(a), f(b) )
      fails
f( _X, _Y )       into    g( f(a), f(b) )
      fails
f( _X, _X )       into    f( f(a), f(a) )
     X := f(a)
```

4. If everything went well, it should be possible to adopt the Makefile so that **rewrite_system.h, rewrite_system.cpp** can be included in the program. Run function **test_rewrite( )**, which uses the rewrite system

$$\begin{array}{lcl} X + 0 & \Rightarrow & X \\ X + s(Y) & \Rightarrow & s(X + Y) \\ \\ X \times 0 & \Rightarrow & 0 \\ X \times s(Y) & \Rightarrow & (X \times Y) + X \\ \\ E(X, X) & \Rightarrow & t \end{array}$$

to test if $2 \cdot 2 \cdot 3$ equals $3 \cdot 2 \cdot 2$.

You can also make some other tests.