

TP Programmation

L3

06 January 2012

Please submit the following by email to `cheval@lsv.ens-cachan.fr` before 23:59 on 15th January 2012 in order to evaluate the 2nd half of Projet Programmation.

- A short report on: What did you achieve in the second half of the course? How? You can explain the difficulties you faced, innovative ideas you used, the complexity aspects of your algorithm, possible improvements etc.
- Code with proper documentation.
- Implement some parsing and display functions (see below).
- The input (translated into the input of the parsing functions) and the output of the following:
 1. Most general unifier for $h(x, g(y, x))$ and $h(g(y, y), g(y, y))$.
 2. Normal form of the term $f(x, e)$ in the rewrite system $\{f(x, f(y, z)) \rightarrow f(f(x, y), z); f(e, x) \rightarrow x; f(i(x), x) \rightarrow e\}$
 3. Is there a recursive path ordering with status (considering only lexicographic and multiset orderings as the basic ordering) to show the termination of the rewriting system $\{(x + y) + z \rightarrow x + (y + z); s(y) \times x \rightarrow x + (y \times x)\}$.
 4. Find the critical pairs of the system $\{f(x, g(y, z)) \rightarrow g((x, y), f(x, z)); g(g(x, y), z) \rightarrow g(x, g(y, z))\}$
 5. Find a convergent term rewriting system equivalent to the following set of identities $\{f(g(f(x))) \approx x; f(g(f(x))) \approx f(g(x))\}$. Choose an appropriate reduction order.

NB: Your code will be tested on other inputs as well.

For the parsing and display functions, we use string as input. We assume that a constant is represented with

```
type signature

(** [parse_signature "f(2), e(0), g(1)"] parse the given signature with symbol
functions and their arity *)
val parse_signature : string -> signature

(** With the previous signature, we consider that "e" and "e()" are both valid
term.*)
```

```
val parse_term : string -> signature -> term
val parse_equation : string -> signature -> term * term
(** [parse_rewriting_rule "t_1 -> t_2"] output [(t_1,t_2)] *)
val parse_rewriting_rule : string -> signature -> term * term
```

For the display functions, it is similar :

```
val display_term : term -> string
val display_equation : term * term -> string
val display_rewriting_rule : term * term -> string
```

As usual, this signature is minimal but you can include more functions if you want. Be sure to implement the previous functions with both DAG and Tree modules.

Have fun !