université PARIS-SACLAY | FACULTÉ DES SCIENCES D'ORSAY

Due date: tuesday march 31, 2026, 20h00

Automates et Applications
M1–MPRI Orsay

Home assignment

Instructions

You should work on this assignment **alone**. You send by email at `kn@lmf.cnrs.fr` an archive file (ZIP or `tar.something`, but not RAR), which contains a single directory named `auto_mpri_dm` which contains:

- A single PDF for the pen and paper question
- A subdirectory with your code for the coding question

The PDF file can be either typeset or handwritten and scanned in that case make sure the definition is sufficient but that the resulting file is not too large, at most 10MB.

At the latest on April 2nd you will receive an acknowledgement by email (usually as a response to your email).

1 Validation of XML documents

The goal of the assignment is to define types, similar to DTDs for XML documents. Such types are written in files with the following syntax:

```
type t = A[ x* | y ]
type x = B[]
type y = choco[]
```

Formally, a *set of type definitions* is a sequence of definitions of the form `type` t `=` l`[` r `]` where t is the name of the type and r is a regular expressions over type names which can use operators " `*` ", " `?` ", concatenation and union "`|`". One can also use parentheses to disambiguate the application of operators. In such definitions, labels, ranged over by l can be:

- A sequence of letters, `_` or digits, starting with a letter (such as `choco`), which denote the fixed name of an XML element.
- The character `*` to denote that any element name is allowed.
- An expression of the form `~`l_1`~`$l_2 \ldots$`~`l_n to indicate that all element names except $l_1, \ldots, l_n$ are allowed.

The goal is to compile such types into a particular kind of tree automata. You can assume that files do not contain any text, comment nor attribute and only markup (that is we limit ourselves to the tree structure of XML documents). For instance:

```
<A><B></B><B></B><B></B></A>
```

There are no restrictions on types other than those given by the syntax above, in particular (contrary to DTDs) the type may not be deterministic top-down:

```
type t = f[ x y ]
type t = f[ y x ]
type x = a[]
type y = b[]
```

We see that this file contains two definitions for `t` and describes a non-deterministic top-down language. Another example is this one:

```
type lst = L[ any lst ]
type lst = N[]
type any = *[ any* ]
```

This type represents an encoding of "linked lists" composed of two cases, the empty list (`N[]`) and the inductive case. Elements of the list have no restriction. For instance the following XML document is valid for the type `lst`:

```
<L><hello></hello><L><how><are><you></you></are></how><N></N></L></L>
```

Since XML Document do not have a fixed alphabet, we choose to encode them as binary trees using the "first-child, next-sibling" encoding (see lecture). We assume a countable, *infinite* set of labels Σ that verify the syntactic criterion for labels and contains an extra symbol # which cannot occur in an XML document, that is $\Sigma = \{\#^0, \texttt{choco}^2, \texttt{L}^2, \texttt{N}^2, \texttt{hello\_world}^2, \ldots\}$. Every symbol is binary except for # which is constant.

In this context an *alternating tree automaton* is an automaton of the form (Q, I, δ) where Q is a set of states, I is a set of root states and $\delta : q \mapsto \mathbb{B}(\Sigma, Q)$ is the transition function. Here $\mathbb{B}(\Sigma, Q)$ is the set of Boolean formulæ which are finite productions of the following grammar:

$$\begin{array}{rcll} \phi & := & \mathsf{lab}(l) & \text{for } l \in \Sigma \\ & | & \downarrow_1 q & \text{for } q \in Q \\ & | & \downarrow_2 q & \text{for } q \in Q \\ & | & \bullet q & \text{for } q \in Q \\ & | & \phi \vee \phi & \\ & | & \phi \wedge \phi & \\ & | & \neg\phi & \end{array}$$

A run is a function $\mathsf{r} : \mathsf{dom}(t) \to 2^Q$ which verifies the following

- $\mathsf{r}(\epsilon) \subseteq I$
- $\forall \pi \in \mathsf{dom}(t), \forall q \in \mathsf{r}(\pi)$, the formula $\delta(q)$ is true

To determine the truth value of the formula for ta path π, let $q \in \mathsf{r}(\pi)$ and let $\phi = \delta(q)$. Then, by case on ϕ

- $\phi \equiv \downarrow_1 q'$ is true iff $q' \in \mathsf{r}(\pi 1)$
- $\phi \equiv \downarrow_2 q'$ is true iff $q' \in \mathsf{r}(\pi 2)$
- $\phi \equiv \bullet q'$ is true iff $q' \in \mathsf{r}(\pi)$
- $\mathsf{lab}(l)$ is true iff $l = t(\pi)$

and disjunction, conjunction and negation are evaluated as usual. Furthermore, the transition function δ defines a set of *contractive* formulæ, that is there are no "cyclic dependencies" of formulæ that go through $\bullet$ move (which correspond to epsilon moves). That is, you can assume that δ will never have transitions such that:

$$\begin{array}{rcl} q_1 & \mapsto & \bullet q_2 \\ q_2 & \mapsto & \bullet q_3 \\ q_3 & \mapsto & \bullet q_1 \end{array}$$

(and like wise with more complex formulæ on the right-hand side), since such definitions might be ill-founded.

Lastly a tree t is recognized by $\mathcal{A}$ if there exists a run and the set of all recognized tree is the language of $\mathcal{A}$ written $L_{\mathcal{A}}$.

2 Questions

1. (2 points) Give an automaton that recognizes the binary tree where the root is `a`, all internal nodes are `b` or `c` and the leaves are `#`.
2. (5 points) Give a (recursive) procedures that take as input a tree t and an automaton $\mathcal{A} = (Q, I, \delta)$ and which returns true if and only if $t \in L_{\mathcal{A}}$. You are free to use the formalism of your choice (pseudo-code, inference rules, ...) but it must be detailed. In particular, you must give a sub-procedure that computes the truth value of a formula. You must also give an argument that shows that your procedure terminates (for instance a well-founded order that decreases at each recursive call).

3. (1 point) Show that a regular bottom-up tree automaton on a fixed alphabet $\Sigma' = \{\#^0, a^2, b^2\}$ can be expressed by an alternating tree automaton.

4. (3 points) Show the converse, that is that an alternating tree automaton on a fixed alphabet $\Sigma' = \{\#^0, a^2, b^2\}$ can be expressed by a non-deterministic bottom-up tree automaton (we ask for the encoding but not for the proof).

5. (2 points) Show how to translate a type (as described in the introduction) in to an alternating tree automaton.

   Remark : you can generalize Thompson's construction for word automata.

6. (7 points) In the programming language of your choice give a data-type for alternating tree automata code and **either** :

   - A data-type for binary trees and the function that tests if a tree belongs to the language of an automaton or (easy, worth 5 points at most);
   - A data-type for XML types (such as given in the introduction) and the function that translate such a type into an alternating tree automaton (harder).