日本ソフトウェア科学会第 27 回大会 (2010 年度) 講演論文集

# Unification algorithm based on $\mu$ equivalence

Fumihiko Yamaguchi

## 1 Introduction

Reasoning on programs will be theoretical fundamentals for automated program verification or synthesis. And unification is basic operation of reasoning, which find substitutions equalizing two objects such as, in this case, two programs. Thus, an unification method that apply to program expressions is desired.

$\lambda$ expression is abstract expression of function, i.e. functional program. And since, it may be convenient for a reasoning system to treat an object as a term, the target object in this paper which is intended to denote a program, is expressed in functional way. In the next section, a formula is defined to denote the target object.

Unification and higher order unification has been actively researched [2] [3]. Unification methods on $\beta\eta$ equivalence of $\lambda$ formulae have also been researched. Since recursive definition is essential to computation, many recursive programs will appear in a reasoning. Therefore, the unification method will be applied to many of them. However, when simply typed $\lambda$ formulae are treated, the fixed point operators cannot be expressed. Even if two recursive programs expressed with fixed point operators are unified by an unification method on $\beta\eta$ equivalence, it's inefficient to try to unify each of many fixed point operators. Thus, simple and special expression of fixed point is desired.

As $\beta\eta$ conversion without type system is powerful enough to be a Turing machine, equalizing two $\lambda$ formulae on $\beta\eta$ equivalence is difficult. Though we can represent recursive functions by fixed point operators such as $Y$ operator, decidability of unification algorithm is desired, when the unification method is considered as a part or a device of reasoning system. To simply denote recursive function, we employ $\mu$ formulae, such that $\mu v.F$ represents the (minimal) solution of the equation $v = F$. From the point of tree automaton, and the point that a program is a grammar of the execution tree, a regular grammar which has finite number of rules corresponds to a program of finite length. And it is known that $\mu$ calculus corresponds to finite state tree automata [4]. Even though a regular grammar derives a tree of arbitrary depth, two regular grammars are identifiable [5]. Therefore, unifiability problem on programs will be decidable.

This paper provides an unification method, whose domain includes fixed point expression, in order to treat recursive functions. Thus, the fixed points should be functions, and the domain of the unification should include function expression. In this paper, fixed points are expressed by $\mu$ formulae, and functions are expressed by $\lambda$ formulae. And for $\lambda$ formulae, this paper does not treat $\beta\eta$ equivalence, but $\alpha$ equivalence.

In most unification methods, when a variable $v$ and a formula $F$ are unified, a substitution $F$ for $v$ is obtained. And when $v$ occurs freely in $F$, some occur-check neglects this substitution. However, the case that a variable $v$ and a formula $F$ should be unified is the case that an equation such that $v$ equals to $F$ is obtained. And the formula which should be substituted to $v$ is the solution of this equation. As with $\mu$ formulae, the solution is rep-

$\mu$ 等価性に基づく単一化アルゴリズム

山口 文彦, Department of Informations Sciences, Graduate School of Tokyo University of Science, 東京理科大学大学院理工学研究科情報科学専攻.

resented by $\mu v.F$. As a special case when $v$ does not occur freely in $F$, $\mu v.F$ equals to $F$. Thus, it seems natural to obtain a substitution $\mu v.F$ for $v$ rather than a substitution $F$ for $v$.

This paper is constructed as follows. The section 2 provides the definition of formulae and the notation of substitution. And then the operation of unification is defined in the section 3. Some discussion are provided in section 4, and the section 5 conclude this paper.

## 2 Preliminaries

Let $\mathcal{V}$ be a countable infinite set of atomic symbols. Each element of $\mathcal{V}$ is called a variable. Let $\mathcal{C}$ also be a countable infinite set of atomic symbols and $\mathcal{V} \cap \mathcal{C}$ be empty. Each element of $\mathcal{C}$ is called a constant.

Let $\mathcal{F}$ be the least set which satisfies following conditions.

- $v \in \mathcal{F}$ when $v \in \mathcal{V}$.
- $c \in \mathcal{F}$ when $c \in \mathcal{C}$.
- $(\lambda v.F) \in \mathcal{F}$ when $v \in \mathcal{V}$ and $F \in \mathcal{F}$. This construction of formula is called $\lambda$ abstraction.
- $(\mu v.F) \in \mathcal{F}$ when $F \in \mathcal{F}$, $v \in \mathcal{FV}(F)$ and $F \not\equiv v$. This construction of formula is called $\mu$ abstraction.
- $(FG) \in \mathcal{F}$ when $F \in \mathcal{F}$ and $G \in \mathcal{F}$. This construction of formula is called application.

Each element of $\mathcal{F}$ is called a formula. The formula $F$ in $(\lambda v.F)$ or $(\mu v.F)$ is called the body of this abstraction.

Parentheses are abbreviated unless ambiguity under the strength of connectives defined as follows: application is associative to the left, and application is stronger than abstraction. Moreover, outer most paretheses are abbreviated.

Let binary relation $\equiv \in \mathcal{F} \times \mathcal{F}$ be the least set which satisfies that $F \equiv F$ for all $F \in \mathcal{F}$. In this paper, the binary relation $\equiv$ represents syntactical equivalence, i.e., for example, $(\lambda x.x) \equiv \lambda x.x$ and $\lambda x.x \not\equiv \lambda y.y$.

$\mathcal{FV}$ is a map from a formula to a set of variables defined as follows:

- $\mathcal{FV}(v)$ is $\{v\}$, where $v \in \mathcal{V}$
- $\mathcal{FV}(c)$ is empty set, where $c \in \mathcal{C}$
- $\mathcal{FV}(\lambda v.F)$ is $\mathcal{FV}(F) - \{v\}$, where $F \in \mathcal{F}$ and $v \in \mathcal{V}$
- $\mathcal{FV}(\mu v.F)$ is $\mathcal{FV}(F) - \{v\}$, where $F \in \mathcal{F}$ and $v \in \mathcal{V}$
- $\mathcal{FV}(FG)$ is $\mathcal{FV}(F) \cup \mathcal{FV}(G)$, where $F, G \in \mathcal{F}$

For a given formula $F$, each element of $\mathcal{FV}(F)$ is called a free variable of $F$. And $\mathcal{BV}$ is also a map from a formula to a set of variables defined as follows:

- $\mathcal{BV}(v)$ is empty set, where $v \in \mathcal{V}$
- $\mathcal{BV}(c)$ is empty set, where $c \in \mathcal{C}$
- $\mathcal{BV}(\lambda v.F)$ is $\mathcal{BV}(F) \cup \{v\}$, where $F \in \mathcal{F}$ and $v \in \mathcal{V}$
- $\mathcal{BV}(\mu v.F)$ is $\mathcal{BV}(F) \cup \{v\}$, where $F \in \mathcal{F}$ and $v \in \mathcal{V}$
- $\mathcal{BV}(FG)$ is $\mathcal{BV}(F) \cup \mathcal{BV}(G)$, where $F, G \in \mathcal{F}$

For a given formula $F$, each element of $\mathcal{BV}(F)$ is called a bound variable of $F$.

$\mathcal{SF}$ is a map from a formula to a set of formulae which is the least set that satisfies following conditions:

- $F \in \mathcal{SF}(F)$, where $F \in \mathcal{F}$
- $\mathcal{SF}(F) \subset \mathcal{SF}(\lambda x.F)$, where $F \in \mathcal{F}$ and $v \in \mathcal{V}$
- $\mathcal{SF}(F) \subset \mathcal{SF}(\mu x.F)$, where $F \in \mathcal{F}$ and $v \in \mathcal{V}$
- $\mathcal{SF}(F) \cup \mathcal{SF}(G) \subset \mathcal{SF}(FG)$, where $F, G \in \mathcal{F}$

For a given formula $F$, each element of $\mathcal{SF}(F)$ is called a sub formula of $F$. And $\mathcal{SF}(F) - \{F\}$ is called the set of proper sub formulae of $F$.

For a given formula $F$, $|F|$ is a natural number called the size of $F$, which is defined as follows:

- $|a|$ is 1 where $a \in \mathcal{V} \cup \mathcal{C}$
- $|\lambda v.F|$ is $|F| + 1$ where $v \in \mathcal{V}$ and $F \in \mathcal{F}$
- $|\mu v.F|$ is $|F| + 1$ where $v \in \mathcal{V}$ and $F \in \mathcal{F}$
- $|FG|$ is $|F| + |G|$ where $F, G \in \mathcal{F}$

A substitution is an operation which replace free variables in a formula into certain formulae. Let $v_1, v_2, \cdots, v_n \in \mathcal{V}$ and $F_1, F_2, \cdots, F_n \in \mathcal{F}$, a substitution which replace each $v_i$ into $F_i$ for $1 \leq i \leq n$ respectively is represented by $[F_1/v_1, F_2/v_2, \cdots, F_n/v_n]$. Especially, in the case that $n = 0$, i.e. a substitution includes no pair of formula and variable, the substitution is called empty and represented by $[]$. A substitution is represented by a lower case greek letter such as $\theta, \rho, \cdots$, when there is no need to describe the detail. The formula obtained by applying a substitution $\theta$ to a formula $F$ is represented in substitution suffixed formula such as $(F\theta)$. To abbreviate parentheses, substitution is associative to the left and the connectivity strength of substitution is defined as the strongest. Thus, for example, $\lambda x.FG\theta\rho \equiv (\lambda x.(F((G\theta)\rho)))$.

Let $\theta$ be a substitution $[F_1/v_1, F_2/v_2, \cdots, F_n/v_n]$, the set of free variables of substitution is defined such that $\mathcal{FV}(\theta)$ is $\bigcup_{1\leq i\leq n} \mathcal{FV}(F_i)$.

## 3 unification

The basic notion of equivalence for $\mu$ formulae is that $\mu v.F$ is equivalent to $F[\mu v.F/v]$.

Equivalence between formulae is returned into the equivalences of their sub formulae: for example, $FG$ and $F'G'$ are equivalent when $F$ and $F'$ are equivalent and $G$ and $G'$ are equivalent. Dividing a formula into its proper sub formulae is decreasing the complexity of the problem. However, returning $\mu v.F$ into $F[\mu v.F/v]$ seems increasing the complexity. The main idea is that two formulae, which are tried to be identified, is memorized, and when the identifying process goes to the same equation, the formulae are identified because there is no witness of difference.

When a variable $v$ and a formula $F$ should be identified, a substitution, i.e. an unification, is obtained. Here, what should be replaced for $v$ is the solution of equation such that $v$ equals to $F$. Thus, what is replaced for $v$ is $\mu v.F$.

$\alpha$ equivalence of $\mu$ formulae is denoted by returning $\mu v.F$ into $F[\mu v.F/v]$ because the variable $v$ in $F$ is replaced to the formula which has no free occurence of $v$. However, there need some rules for $\alpha$ equivalence of $\lambda$ formulae. When $\lambda x.F$ and $\lambda y.G$ should be identified, a fresh variable $z$ is required and the problem is returned into identification of $F[z/x]$ and $G[z/y]$. It seems that the variable $z$ occurs freely in $F[z/x]$ and $G[z/y]$, however no substitution which replace this $z$ is allowed because the process is in the context in which $z$ is bound. Thus, the set of bound variables needs to be memorized as a context. Each variable in this set is called locked.

The binary relation on $\mathcal{F}$, equivalence with substitution denoted by $=$ is defined in **Fig. 1** in terms of sequent calculus. The left hand side of a sequent consists of three elements: a substitution, a set of pairs of formulae and a set of locked variables. The set of pairs of formulae is for memorizing the two formulae to be identified without further proof. At least one formula of the pair is $\mu$ abstraction. And the set of pairs is denoted by $\Gamma$. The set of locked variables is denoted by $\Delta$. When and only when a proof tree whose root is $\theta;;\vdash F=G$ exists, i.e. $\Gamma$ and $\Delta$ are empty, $F$ and $G$ are called unifiable with $\theta$. Here the unifier $\theta$ is obtained. Especially, when $\theta$ is empty, i.e. $[\,];;\vdash F=G$ has a proof tree, $F$ and $G$ are called equivalent.

The construction of formulae forbid the existence of $\mu x.x$. Since $\mu v.F$ is considered as the solution of an equation such that $v$ equals to $F$, $\mu x.x$ denotes the solution of a trivial equation such that $x$ equals to $x$. This means that the variable $x$ can be assinged to any formula. Indeed, $\mu x.x$ equals to any other formula under the definition of equivalence, shown as follows.

$$\frac{[\,];\langle \mu x.x, F\rangle;\vdash \mu x.x=F}{[\,];;\vdash \mu x.x=F}$$

Moreover, $\mu$ abstaction requires that the bound variable occurs freely in its body. Suppose that $\mu$ formulae with no free occurrence of the bound variable in its body is allowed, i.e. in the case that $v\notin\mathcal{FV}(F)$, because $F[\mu v.F/v]$ equals to $F$, $\mu v.F$ is equivalent to $F$; especially, $\mu x.\mu y.x$ is equivalent to the forbidden formula $\mu x.x$ and there may be many forbidden formulae. On the contrary, as in the definition of formulae, when the body of $\mu$ abstraction includes at least one free occurence of the bound variable in its body, and the body is not the variable itself, then the body have to be some other construction. Therefore, a formula which can be identified with the $\mu$ formula should have the same construction. This means that not any formula equals to the $\mu$ formula.

The operation of above unification is terminating. Though the rule for $\mu$ abstraction seems to break the sub formula property, the set of formulae in a proof is finite because $\mu v.F$ into $F[\mu v.F/v]$ is applied only when the target formula is $\mu$ abstraction. The rules of unification traverse the construction of the formulae to be identified. When $F$ is deconstructed into $F$ itself in $n$ steps, and $G$ in $m$ steps, $F=G$ returns into $F=G$ in at most $nm$ steps with memorizing the pair $\langle F,G\rangle$. And $n\leq|F|$ and $m\leq|G|$. Therefore, unification is decidable.

The definition of unification identifies two formulae when there is no witness of difference for equivalence, especially when both formulae are $\mu$ abstraction. For example, the proof tree of $\theta;;\vdash \mu x.Fx=\mu x.Gx$ include the proof of

$$\theta;\Gamma;\Delta\vdash F=F \quad \text{where } F\in\mathcal{F}$$

$$\theta;\langle F',G'\rangle,\Gamma;\Delta\vdash F=G \quad \text{where } F=_\alpha F' \text{ and } G=_\alpha G'$$

$$[\mu v.G/v]\circ\theta;\Gamma;\Delta\vdash v=G \quad \text{where } v\in\mathcal{FV}(G)-\Delta,\ \mathcal{FV}(G)\cap\Delta \text{ is empty and } G\not\equiv v$$

$$[G/v]\circ\theta;\Gamma;\Delta\vdash v=G \quad \text{where } v\in\mathcal{V}-\Delta-\mathcal{FV}(G) \text{ and } \mathcal{FV}(G)\cap\Delta \text{ is empty}$$

$$[\mu v.F/v]\circ\theta;\Gamma;\Delta\vdash F=v \quad \text{where } v\in\mathcal{FV}(F)-\Delta,\ \mathcal{FV}(F)\cap\Delta \text{ is empty and } F\not\equiv v$$

$$[F/v]\circ\theta;\Gamma;\Delta\vdash F=v \quad \text{where } v\in\mathcal{V}-\Delta-\mathcal{FV}(F) \text{ and } \mathcal{FV}(F)\cap\Delta \text{ is empty}$$

$$\frac{\theta;\Gamma;z,\Delta\vdash F[z/x]=G[z/y]}{\theta;\Gamma;\Delta\vdash\lambda x.F=\lambda y.G}\ \text{where } z\in\mathcal{V} \text{ and } z\notin\mathcal{FV}(F)\cup\mathcal{FV}(G)$$

$$\frac{\theta;\langle\mu x.F,G\rangle,\Gamma;\Delta\vdash F[\mu x.F/x]=G}{\theta;\Gamma;\Delta\vdash\mu x.F=G} \qquad \frac{\theta;\langle F,\mu y.G\rangle,\Gamma;\Delta\vdash F=G[\mu y.G/y]}{\theta;\Gamma;\Delta\vdash F=\mu y.G}$$

$$\frac{\theta_1;\Gamma;\Delta\vdash F_1=G_1 \qquad \theta_2;\Gamma;\Delta\vdash F_2\theta_1=G_2\theta_1}{\theta_1\circ\theta_2;\Gamma;\Delta\vdash F_1F_2=G_1G_2}$$

**Fig. 1 unification**

$\mu x.Fx=\mu x.Gx$ in its sub tree as follows, where $x\notin\mathcal{FV}(F)\cup\mathcal{FV}(G)$ :

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{\theta;P_1,P_2;\vdash F=G} \qquad \theta;P_1,P_2;\vdash\mu x.Fx=\mu x.Gx}{\theta;P_1,P_2;\vdash F(\mu x.Fx)=G(\mu x.Gx)}}{\theta;P_1;\vdash F(\mu x.Fx)=\mu x.Gx}}{\theta;;\vdash\mu x.Fx=\mu x.Gx}$$

where $P_1\equiv\langle\mu x.Fx,\mu x.Gx\rangle$
$P_2\equiv\langle F(\mu x.Fx),\mu x.Gx\rangle$

Simultaneously, $F$ and $G$ are on the equalizing process which is indicated by vertical dots above. This example shows that the witness of difference is seeked through the other sub formulae than the variable bound by the $\mu$ abstraction.

## 4 Discussion

In this section, some discussions around the proposed unification is provided.

### 4.1 Scope management

Introducing fresh variables is not enough to prevent from yielding a substitution of a formula including a free variable which is bound by the external context. For example, $\lambda x.xy$ and $\lambda x.xx$ are not unifiable because $x$ is bound in $\lambda x.xx$. The failure proof of this equation is as follows:

$$\dfrac{\dfrac{\theta;;z\vdash z=z \qquad \dfrac{\mathit{fail}}{\theta;;z\vdash y=z}}{\theta;;z\vdash(xy)[z/x]=(xx)[z/x]}}{\theta;;\vdash\lambda x.xy=\lambda x.xx}$$

The sequent $\theta;;z\vdash y=z$ fails because both two possibility fail as follows. One is trying to obtain substitution $[z/y]$, however $\mathcal{FV}(G)\cap\Delta$ is non empty where in this case $G\equiv z$ and $\Delta$ is $\{z\}$. The other is trying to obtain $[y/z]$, however $v\notin\mathcal{V}-\Delta-\mathcal{FV}(F)$ where in this case $v\equiv z$ and $\Delta$ is $\{z\}$. In this paper, the scope of variable is strictly managed such as a substitution of a formula within bound variable is not allowed.

### 4.2 Types

Introducing type system sometimes has advantage such as enabling strong normalizablity. The formula in this paper is also typable with typing rule for $\mu$ abstraction as follows: for a formula $\mu v.F$, all of $v$, $F$ and $\mu v.F$ have the same type. This is because $\mu v.F$ represents the solution of an equation such that $v$ equals to $F$. However, because $\mu$ formula is intended to represent possibly infinite calculation, type system has no advantage from the point of strong normalization.

From the point of type checking for reducing redundant or useless calculation, the type system may have advantage. It is known that when the formula is evaluated in applicable order, only fixed point of higher order function can be calculated in finite steps. This means that the fixed point must be a function, i.e. every $\mu$ formula has function type. Therefore, in the case that the underlying computation is in applicable order, the unification problem between $v$ and $F$ comes and $v \in \mathcal{FV}(F)$, then we can reject the unification when $v$ or $F$ has basic type, or we obtain the unification $[\mu v.F/v]$ with assignment of function type to $v$ and $F$.

### 4.3 Unifier composition with $\mu$ formulae construction

In the definition of unification, the unification rule for formulae which are application, an unifier $\theta_1$ obtained by the left unification and the right one $\theta_2$ are composed. Considering unification problem as equation, unification for application is considered as simultaneous equation. Let $v$ be one of the variables to be replaced by $\theta_1$, when $v \in \mathcal{FV}(\theta_2)$, $v$ should be the solution of the simultaneous equation. For example, let $\theta_1$ be $[(f\ x)/v]$ and $\theta_2$ be $[v/x]$, $\theta_1 \circ \theta_2$ is $[(f\ v)/v, v/x]$ while $v$ should be the solution of equation such that $v$ equals to $(f\ v)$ as the composed substitution indicates. Thus there is a question whether, in this case, the result should be $[\mu v.(f\ v)/v, \mu v.(f\ v)/x]$.

However, by the definition of unification, there is no such case. The right formulae are unified after the substitution $\theta_1$ which is obtained by the left unification. Since any variable replaced by $\theta_1$ does not occur freely in $F_2\theta_1$ nor $G_2\theta_1$, $\mathcal{FV}(\theta_2)$ does not include such variable.

Some other unification methods, typically syntactical unification, check the occurence of free variables when substitution is constructed, not when substitutions are composed. As the unification method provided in this paper introduce $\mu$ formula instead of occur check, the operation of composing substitution is simplified in the similar way.

### 4.4 Effect of lock check

In the case that $x \notin \mathcal{FV}(F) \cup \mathcal{FV}(\theta)$, a substitution relation $(\lambda x.F)\theta \to \lambda x.G$ may have the following proof tree.

$$\frac{x \vdash F[x/x] \to F \qquad x \vdash F\theta \to G}{\vdash (\lambda x.F)\theta \to \lambda x.G}$$

In the left sub tree, the substitution $[x/x]$ cause redundant or useless calculation by three reasons such that it's identical substitution, that $x \notin \mathcal{FV}(F)$ and that $x$ is locked even if $x \in \mathcal{FV}(F)$. Of cause, in such a case, the rule for the case of the same bound variables is intended to be applied.

$$\frac{x \vdash F\theta \to G}{\vdash (\lambda x.F)\theta \to \lambda x.G}$$

However, the rule in the later tree is not a simple special case of that of the former tree, because the later tree is enabled even when $x \in \mathcal{FV}(F)$.

At the beginning, the concept of locked variables is introduced to prevent replacing captured variables. It works to reducing redundant or useless calculation. To obtain the result of substitution, there is a strategy such that a fresh variable is always intoroduced for abstraction. In this strategy, there is no need to check locked variables. However, this strategy yield many $\alpha$ equivalent formulae.

Considering that the unification method is used in some reasoning system, there may be many chance to check the equivalence. It is plausible strategy such that identification of syntactically equivalent formulae is prior to those not syntactical but $\alpha$ equivalent. Therefore, reducing $\alpha$ equivalent formulae cause efficiency for the external resoning system.

Checking locked variables in unification also has effect of efficiency. A fresh variable is introduced for every equation of $\lambda$ abstractions. When the lock check such as $v \in \mathcal{FV}(G) - \Delta$ is not made, substitutions which replace bound variables are obtained. Since the substitution is defined as no bound variable is replaced, lock check has advantage of avoid-

ing useless substitution.

In this manner, the scope of variable is managed strictly. For example, $\lambda x.xy$ and $\lambda x.xx$ are not unifiable because $\lambda x.xx$ equals to $\lambda z.zz$ and the maybe unifier $[x/y]$ cannot unify $\lambda x.xy$ and $\lambda z.zz$. However, unification with substitution of a formula within bound variable for a free variable may be useful. There are three classes of variable: must be bound, must be free and may be both. The former two classes are treated in this paper. As free variables must not be conflicted with bound variables, bound variables are need to be renamed. However, when a substitution includes bound variables, the renaming should be rewinded.

## 5 Conclusion

This paper provides a decidable unification method on $\alpha$ equivalence and $\mu$ equivalence for expressions of functions and fixed points. Introducing $\mu$ formulae allows to denote recursive functions without side effect such as changing environment of function name assigned to its entity.

The development of resoning system with $\mu$ formulae is the future work.

## References

[1] Christian Urban, Andrew Pitts, Murdoch Gabbay, "Nominal Unification", *in* M.Baaz and J.A.Makowsky eds., CSL 2003, LNCS 2803, Springer-Verlag Berlin Heidelberg, pp. 513–527, 2003.

[2] Franz Baader, Wayne Snyder, "Unification Theory", *in* A.Robinson and A.Voronkov eds., 'Handbook of Automated Reasoning', Vol. I, Elsevier Science, chapter 8, pp. 445–533, 2001.

[3] Gilles Dowek, "Higher-Order Unification and Matching", *in* A.Robinson and A.Voronkov eds., 'Handbook of Automated Reasoning', Vol. II, Elsevier Science, chapter 16, pp. 1009–1062, 2001.

[4] David Janin, Igor Walukiewicz, Automata for the modal $\mu$-calculus and related results, Proceedings of the 20th International Symposium, Mathematical Foundations of Computer Science, Prague, Czech Republic, Vol. 969 of Lecture Notes in Computer Science, Springer, pp. 552–562, 1995

[5] Hubert Comon, Max Dauchet, Rmi Gilleron, Florent Jacquemard, Denis Lugiez, Sophie Tison, Marc Tommasi, "Tree Automata Techniques and Applications", `http://www.grappa-univ-lille3.fr/tata/`