Tree Data Decision Diagrams

Jean-Michel COUVREUR, Duy-Tung NGUYEN
LIFO (Laboratoire d’Informatique Fondamentale d’Orléans),
University of Orleans,
Léonard de Vinci street B.P. 6759, F-45067 ORLEANS Cedex 2
http://www.univ-orleans.fr/lifo/
couvreur@univ-orleans.fr

Abstract

In this paper, we present Tree Data Decision Diagrams, a compact data structure of symbolic verification based on term rewriting systems. By this way, we can benefit termination researches in term rewriting systems to improve the model-checking quality. Our experimental implementation uses tree automata technique that provides the capability to maintain the internal representation of data in canonical form.

Keywords: Modelling, model-checking, term rewriting system, symbolic verification, data decision diagrams, termination, tree automata

1. INTRODUCTION

Tree Data Decision Diagrams (TDDD for short) is a hierarchical version of Data Decision Diagrams (DDD) [2, 3] developed in the spirit of the well-known Binary Decision Diagram (BDD) [1].

DDD are a directed acyclic graph structure that manipulates (a priori unbounded) integer domain variables, and which offers a flexible and compositional definition of operations through inductive homomorphisms. In [4], we presented data structure called Set Decision Diagrams (SDD), as an arc of the structure is labeled by a set of values, instead of a single valuation. The set is itself represented by a SDD or DDD, thus in effect we label the arcs of our structure with references to SDD or DDD, introducing hierarchy in the data structure.

TDDD likes DDD structures with representing sets of sequences of assignments of the variables and their values, but each variable has either a value or a sub sequence.

Considering TDDD in term rewriting systems (TRS for short) will be not only compact as SDD (more than DDD) but its operators are also very simpler and more flexibly than the others (more detail in section 3).

Moreover, the two basic techniques used for proving TDDD termination are path orders and semantic labelling. Path orders are direct techniques to prove termination, while semantic labelling is technique for transforming a TRS to another one in such a way that termination of the original TRS can be concluded from (relative) termination of the transformed TRS. For further details we refer to [8, 9, 11, 12].

When a model is proved terminating, we call it a well-designed. Consequently, model designed by TRS can detect the non-terminating component and help developers get oriented to the better designed. When a model is well-designed, the model-checker will generate all of reachability states if the system resources are enough.

At the end of this paper, we report on our experience of implementations using tree automata technique [13] that provides the capability to maintain the internal representation of data in canonical form.

2. PRELIMINARIES

2.1. Term rewriting systems and termination proving

Term rewriting systems. A signature is a countable set $F$ of function symbols (or operators). Associated with every $f \in F$ is a natural number denoting its arity. Function symbols of arity 0 are called constants. Let $T(F, V)$ be the set of all terms built from $F$ and a countably infinite set $V$ of variables, disjoint from $F$. If $t$ is a term then $Var(t)$ denotes the set of variables occurring in $t$. A term $t$ is called ground if $Var(t) = \emptyset$. The set of all ground terms is denoted by...
A term $t$ is called linear if it does not contain multiple occurrences of the same variable. The root symbol of a term $t$ is defined as follows: \( \text{root}(t) = t \) if $t$ is a variable and \( \text{root}(t) = f \) if $t = f(t_1, ..., t_n)$. The size \(|t|\) of a term $t$ is the number of variables and function symbols occurring in $t$.

We introduce a fresh constant symbol $\square$, named hole. A context $C$ is a term in $T(F \cup \{\square\}, V)$. The designation term is restricted to members of $T(F, V)$. A context may contain zero, one or more holes. If $C$ is a context with $n$ holes and $t_1, ..., t_n$ are terms then $C[t_1, ..., t_n]$ denotes the result of replacing from left to right the holes in $C$ by $t_1, ..., t_n$. A term $s$ is a sub-term of a term $t$ if there exists a context $C$ such that $t = C[s]$. A sub-term $s$ of $t$ is proper, denoted by $t \triangleright s$, if $s \neq t$. A substitution is a map $\sigma$ from $V$ to $T(F, V)$. If $\sigma$ is a substitution and $t$ a term then $t\sigma$ denotes the result of applying $\sigma$ to $t$. We call $t\sigma$ an instance of $t$. A binary relation $\triangleright$ on terms is a rewrite relation if it is closed under contexts and substitutions, i.e. if $t \triangleright s$ then $C[t\sigma] \triangleright C[s\sigma]$ for all contexts $C$ (with precisely one hole) and substitutions $\sigma$.

A rewrite rule is a pair $(l, r)$ of terms such that the left-hand side (lhs) $l$ is not a variable and variables which occur in the right-hand side (rhs) $r$ occur also in $l$, i.e. $\text{Var}(r) \subseteq \text{Var}(l)$. Rewrite rules $(l, r)$ will henceforth be written as $l \Rightarrow r$. A rewrite rule is collapsing if its rhs is a single variable. A rewrite rule is duplicating if its rhs contains more occurrences of some variable than its lhs. A rewrite rule is left-linear (right-linear) if its lhs (rhs) is a linear term.

A TRS is a pair $(F, R)$ consisting of a signature $F$ and a set $R$ of rewrite rules between terms in $T(F, V)$. If $(F, R)$ is a TRS then $\Rightarrow_R$ denotes the smallest rewrite relation on $T(F, V)$ containing $R$. So $t \Rightarrow_R s$ if there exists a rewrite rule $l \Rightarrow r$ in $R$, a substitution $\sigma$ and a context $C$ such that $t = C[\sigma]$ and $s = C[r\sigma]$. The sub-term $l\sigma$ of $t$ is called a redex and we say that $t$ rewrites to $s$ by contracting redex $l\sigma$. We call $t \Rightarrow_R s$ a rewrite or reduction step. If $C = \square$ then we speak of a root reduction. The transitive closure of $\Rightarrow$ is denoted by $\Rightarrow^+_R$ and $\Rightarrow_R^*$ denotes the transitive-reflexive closure of $R$. If $t \Rightarrow^+_R s$ we say that $t$ reduces to $s$.

Termination proving. Termination of TRS is an undecidable problem even with finite $(F, R)$.

A rewrite relation that is also a (strict) partial order is called a rewrite order. An order $\triangleright$ is called well-founded if there is no infinite descending sequence $t_1 \triangleright t_2 \triangleright \ldots$.

An order $\triangleright$ on $T(F)$ is called monotonic if $t \triangleright u \Rightarrow f(..., t, ...) \triangleright f(..., u, ...)$ for all $f \in F$. A TRS $(F, R)$ and an order $\triangleright$ are called compatible if $t \triangleright u$ for all rewrite steps $t \Rightarrow_R u$. For compatibility with a monotonic order it suffices to check that $l\sigma \triangleright r\sigma$ for all rules $l \Rightarrow r$ in $R$ and all ground substitution $\sigma$. It is well-known that a TRS is terminating if it is compatible with some monotonic well-founded order. An order $\triangleright$ on $T(F)$ is said to have the sub-term property if $f(..., t, ...) \triangleright t$ for all $f \in F$ and $t \in T(F)$. The monotonic order satisfying the sub-term property is called a simplification order. A direct consequence of Kruskal’s theorem [8] is that any simplification order over a finite signature is well-founded.

A TRS $(F, R)$ is compatible with a rewrite order $\triangleright$ on $T(F, V)$ if $l \triangleright r$ for every rewrite rule $l \Rightarrow r$ of $R$. It is easy to show that a TRS is terminating if and only if it is compatible with a well-founded rewrite order. The simplification is as the following:

- A simplification order is a rewrite order $\triangleright$ with the sub-term property, i.e. $C[t] \triangleright t$ for all contexts $C \neq \square$ (with precisely one hole) and terms $t$.
- A TRS is called simplifying if it is compatible with a simplification order.
- A TRS is called simply terminating if it is compatible with a well-founded simplification order.

Clearly every simply terminating TRS is both simplifying and terminating. A simplifying TRS $(F, R)$ with $F$ or $R$ finite is simply terminating, as a consequence of Kruskal’s Tree Theorem [8]. There exists (infinite) simplifying and terminating TRSs that are not simply terminating, see [10]. This does not concern us too much as we will deal with decidability issues in the sequel, in which one considers only finite (both with respect to signature and set of rewrite rules) TRSs.

The recursive path order (RPO) is introduced by Dershowitz. Kamin and Levy present the lexicographic path order (LPO), a well-known variant of the RPO. They are defined recursively as follows:

Let $\gg$ be any order on the signature $F$. Then for two ground terms $t = f(t_1, ..., t_n)$ and $u = g(u_1, ..., u_m)$ one has $t \gg u$ if:  

- $t_i = u_i$ or $t_i \triangleright u_i$ for some $i = 1, \ldots, n$,
- $f \gg g$ and $t_i \gg u_i$ for all $i = 1, \ldots, m$, or
Tree Data Decision Diagrams

- \( f = g \) and \( \{ t_1, \ldots, t_n \} \uparrow_{p\!o} \{ u_1, \ldots, u_m \} \) with RPO
  (or \( \{ t_1, \ldots, t_n \} \uparrow_{p\!o} \{ u_1, \ldots, u_m \} \) with LPO).

Here for any order \( \uparrow \) the order \( \uparrow_{l\!e\!x} \) means the lexicographic extension of \( \uparrow \) to sequences. The lexicographic comparison has to be done in a fixed direction; in the paper it will be from right to left. It should be noted that only sequences of equal length are compared, since they require that every symbol has a fixed arity. It is well-known that \( \uparrow_{p\!o} \) is monotonic and has the sub-term property. Further \( \uparrow_{l\!e\!x} \) is total on ground terms iff \( \uparrow \) is total on \( F \).

Semantic labelling provides a technique for proving termination, making classical techniques like path orders applicable even for non-simplifying TRS's. Let \( M \) be a model for a TRS \( R \) over \( F \). Choose for every \( f \in F \) a non-empty set \( S_f \) of labels and a map \( \pi_f : M^n \rightarrow S_f \), where \( n \) is the arity of \( f \). So TRS \( < F, R > \) can be terminating if the \( < F^{lab}, R^{lab} > \) is terminating.

2.2. Non-deterministic Top-down finite tree automaton

A Non-deterministic Top-down finite tree automaton [13] (NFTA for short) over \( F \) is a tuple \( A = (Q, F, I, \triangle) \) where \( Q \) is a set of states (states are unary symbols), \( I \subseteq Q \) is a set of initial states, and \( \triangle \) is a set of rewrite rules of the following type:

\[ q(f(t_1, \ldots, t_n)) \rightarrow f(q_1(t_1), \ldots, q_n(t_n)), \]

where \( n \geq 0, f \in F, q, q_1, \ldots, q_n \in Q, t_1, \ldots, t_n \in T. \)

When \( n = 0 \), i.e. when the symbol is a constant symbol \( c \), a transition rule of NFTA is of the form \( q(c) \rightarrow c \). For simplifying the automata representation, we can name the state \( \bar{c} \in Q \) for each corresponding constant symbol \( c \). Ex: \( 1(1) \rightarrow 1 \) or \( \bar{s}(\bar{s}) \rightarrow \bar{s} \).

An automaton starts at the root and moves downward, associating along a run a state with each sub-term inductively. The tree language \( L(A) \) recognized by \( A \) is the set of all ground terms \( t \) for which there is an initial state \( q \) in \( I \) such that \( q(t) \rightarrow^* \triangle t. \)

We can organize data structure for finite tree automaton (i.e. no cycle exists in the tree automaton) with respecting the canonicity. It should be noted that a canonical finite tree automaton has not always a minimal state number.

In infinite case, we must use minimization algorithm for Non-deterministic tree automaton like bisimulation minimization [14, 15], etc.

2.3. Data Decision Diagrams and Set Decision Diagrams

Data Decision Diagrams [2, 3] are a directed acyclic graph structure that manipulates (a priori unbounded) integer domain variables, and which offers a flexible and compositional definition of operations through inductive homomorphisms.

DDD are data structure for representing finite sets of assignments sequences of the form \( e_1 \overset{x_1}{\rightarrow} e_2 \overset{x_2}{\rightarrow} \ldots e_n \overset{x_n}{\rightarrow} 1 \)
where \( e_i \) are variables and \( x_i \) are the assigned integer values. When an ordering on the variables is fixed and the values are boolean, DDD coincides with the well-known Binary Decision Diagram. However DDD assume no variable ordering and, even more, the same variable may occur many times in the same assignment sequence. Moreover, variables are not assumed to be part of all paths. Therefore, the maximal length of a sequence is not fixed, and sequences of different lengths can coexist in a DDD. This feature is very useful when dealing with dynamic structures like queues.

Operators on these structures are not hard-coded, but a class of operators, called homomorphisms, is introduced to allow transition rules coding. A special kind of homomorphisms uses only local information to a node in its definition. Together with composition, concatenation, union, etc operations, general homomorphisms are defined.

DDD have two terminals: as usual for decision diagram, 1-leaves stand for accepting terminators and 0-leaves for non-accepting ones. Since there is no assumption on the variable domains, the non-accepted sequences are suppressed from the structure. 0 is considered as the default value and is only used to denote the empty set of sequence.
Set Decision Diagrams [4] are data structures for representing sequences of assignments of the form \( e_1 \in a_1; e_2 \in a_2; \ldots; e_n \in a_n \), where \( e_i \) are variables and \( a_i \) are sets of values. SDD can therefore simply be seen as a different encoding for set of assignment sequences of the same form as those of DDD, obtained by flattening the structure, i.e. as a DDD defined as \( \bigcup_{a_1 \in a_1} \bigcup_{a_2 \in a_2} \ldots \bigcup_{a_n \in a_n} e_1 \xRightarrow{a_1} e_2 \xRightarrow{a_2} \ldots e_n \xRightarrow{a_n} 1. \)

SDD allows to generalize some of these patterns of good decision diagram usage, in an open and flexible framework, inductive homomorphisms. SDD are naturally adapted to the representation of state spaces composed in parallel behavior, with event based synchronizations. The structure of a model is reflected in the hierarchy of the decision diagram encoding, allowing sharing of both operations and state representation. SDD allow to flexibly compute local fixpoints.

3. TREE DATA DECISION DIAGRAMS

3.1. Definitions

We consider four kinds of symbols: Reducible symbols (Upper-case characters) are denoted by \( \Gamma = \{ A, B, C, \ldots \} \). Irreducible symbols or dummies (Lower-case characters) are denoted by \( \sum = \{ a, b, c, \ldots \} \). Constant symbols (Values symbols): \( C = \{ 0, 1, 2, \ldots \} \) and a special termination symbol \( \$ \). We notice the signature (or function symbols set) \( F = \sum \cup \Gamma \cup C \cup \{ \$ \} \).

**Definition 3.1 (TDDD terms set).** A term \( t \) in \( T(F) \) is defined inductively by:

\[
t ::= H(t, t) \mid f(t, t) \mid c
\]

with \( H \in \Gamma, f \in \sum, c \in C \cup \{ \$ \} \).

We represent ground term set as an extension of term \( t \):

\[
s ::= H(s, s) \mid f(s, s) \mid s + s \mid t
\]

with \( H \in \Gamma, f \in \sum, t \in T(F) \).

On term sets, we have the following linear properties:

- \( \alpha(s, s_1 + s_2) = \alpha(s, s_1) + \alpha(s, s_2) \)
- \( \alpha(s_1 + s_2, s) = \alpha(s_1, s) + \alpha(s_2, s) \)

where \( \alpha \) is either reducible or irreducible symbol.

The set of upper-case (or lower-case) symbols appearing in a term set \( s \) is denoted by \( FUN(s) \) (or \( fun(s) \)).

**Definition 3.2 (Rules set).** Rules set \( R \) is represented under form: \( l \rightarrow r \), where \( l, r \) are terms containing variables ranging over \( T \), i.e. \( l, r \in T(F, V) \) with the variables in \( V \) are denoted as \( x, y, z, \ldots \).

A term is closed if it does not containing variable. Notice that term set we consider are finite and may be repeated by an acyclic NFTA which is defined in section 2.2:

\[
q (f(t_1, t_2)) \rightarrow f(q_1(t_1), q_2(t_2)),
\]

where \( f \in F, q_1, q_2 \in Q, t_1, t_2 \) are terms.

We are interested in the tree language \( L(A) \), the term set \( s \subseteq L(A) \) is considered as a set of \( t \) each which \( t \in s \) there is an initial state \( q \) in \( f \) such that \( q(t) \rightarrow^*_A t \).

We say that term set is irreducible if it does not contain any reducible symbol (i.e. \( FUN(s) = 0 \)), we called it TDDD tree. DDD are also term sets where term is under form \( f(i, t) \) with \( i \) is a constant.

**Definition 3.3 (Root reduction).** The root reduction \( s' = \triangle_R(s) \) is defined as the reduction of term set \( s \) with \( H \) is the root of \( < F, R > \) and \( s, s' \) are irreducible term sets.

**Definition 3.4 (Fixed point computation).** The fixed point \( FP_R(s) \) of a rules set \( R \) from an initial term set \( s \) is defined recursively like:

\[
FP_R(s) ::= s + \triangle_R(FP_R(s)).
\]
We aim at finding the solution (or partial solution) for the termination problem of root reduction \((\square_R(s))\) and fixed point computation \((FP_R(s))\).

Sometime, TRS design requires only an unary symbol \(H\). In this case, special symbol \(\$\) will occur on the left sub-term position for a binary representation, ex \(H(\$, t)\).

**Example 1** The corresponding TRSs of a value increment of variable \(c\):

\[
\begin{align*}
& r_0 \colon H(\$, \$) \rightarrow \$ & \text{If sub-term of } H \text{ is } \$, \text{ eliminate } H. \\
& r_1 \colon H(\$, 0) \rightarrow 1 & \text{If sub-term of } H \text{ is a value,} \\
& r_2 \colon H(\$, 1) \rightarrow 2 & \text{increase it} \\
& r_3 \colon H(\$, 2) \rightarrow 3 & \text{If sub-term of } H \text{ is variable } a \text{ or } d, \\
& r_4 \colon H(\$, a(x, y)) \rightarrow a(x, H(\$, y)) & \text{let } H \text{ go down to right hand sub-term} \\
& r_5 \colon H(\$, d(x, y)) \rightarrow d(x, H(\$, y)) & \text{If sub-term of } H \text{ is variable } c, \text{ let } H \text{ go down to left hand sub-term} \\
& r_6 \colon H(\$, c(x, y)) \rightarrow c(H(\$, x), y) & \text{in order to change value of } c \text{ in the next step} \\
& r_7 \colon H(\$, b(x, y)) \rightarrow b(x, H(\$, y)) & \text{If sub-term of } H \text{ is variable } b, \\
& r_8 \colon H(\$, b(x, y)) \rightarrow b(H(\$, x), y) & \text{let } H \text{ go down to each sub-term.}
\end{align*}
\]

The interesting questions are:

- How to enumerate every reached terms from an initial term set:
  \[
s = a(0, b(c(1, \$), d(2, \$))) + b(d(0, \$), a(0, \$))?
\]
- Whether \(\triangle_R(s) = H(\$, s)\) and fixed point \((FP_R(s) = H^*(\$, s))\) are terminating?

From section 2.2, initial tree automata \(A_{1,a}\) is depicted in the figure 1.a. We have \(s \subseteq L(A_{1,a})\) since \(a(0, b(c(1, \$), d(2, \$))) \in L(A_{1,a})\):

\[
q_0(a(0, b(c(1, \$), d(2, \$)))) \\
\rightarrow a(0(0), q_3(b(c(1, \$), d(2, \$)))) \\
\rightarrow a(0, b(q_3(c(1, \$)), q_4(d(2, \$)))) \\
\rightarrow a(0, b(c(1(1), \$)), d(2(2), \$))) \\
\rightarrow a(0, b(c(1, \$), d(2(2))))
\]

and since \(b(d(0, \$), a(0, \$)) \in L(A_{1,a})\):

\[
q_0(b(d(0, \$), a(0, \$))) \\
\rightarrow b(q_2(d(0, \$), q_1(a(0, \$))) \\
\rightarrow b(d(0(0), \$), a(0(0), \$)) \\
\rightarrow b(d(0, \$), a(0, \$))
\]

The simplification is as the following:

\[
\begin{align*}
& s' = H(\$, \$) \\
& \xrightarrow{r_4,r_7,r_8} \\
& a(0, H(\$, b(c(1, \$), d(2, \$)))) \\
& + b(H(\$, d(0, \$)), a(0, \$)) + b(d(0, \$), H(\$, a(0, \$))) \\
\end{align*}
\]

and then we have:

\[
\begin{align*}
& a(0, H(\$, b(c(1, \$), d(2, \$)))) \\
& \xrightarrow{r_7,r_8} a(0, b(H(\$, c(1, \$)), d(2, \$))) + a(0, b(c(1, \$), H(\$, d(2, \$)))) \\
& \xrightarrow{r_6,r_8} a(0, b(c(H(\$, 1), \$), d(2, \$))) + a(0, b(c(1, \$), d(2, H(\$, \$)))) \\
& \xrightarrow{r_4,r_8} a(0, b(c(b(2, \$), d(2, \$))) + a(0, b(c(1, \$), d(2, \$)))
\end{align*}
\]

and \(b(H(\$, d(0, \$), a(0, \$)) + b(d(0, \$), H(\$, a(0, \$))) \) provide a canonical representation:

\[
\begin{align*}
& \xrightarrow{r_9} b(d(0, H(\$, \$), a(0, \$)) + b(d(0, \$), a(0, \$)) + b(d(0, \$), a(0, \$)) \}
\end{align*}
\]

The factorization \((a(0, b(c(2, \$), d(2, \$))) + a(0, b(c(1, \$), d(2, \$))))\) and the redundancy elimination \((b(d(0, \$), a(0, \$)) + b(d(0, \$), a(0, \$)))\) provide a canonical representation:
Tree Data Decision Diagrams

FIGURE 1: Tree automaton representation: a) Initial tree automaton b) Fixed point tree automaton

\[ s' = a(0, b(c(1,5) + 2,5), d(2,8)) + b(d(0,8), a(0,8)) \]

Similarly: \[ s'' = H(8, s') = a(0, b(c(1,5) + 2,5 + 3,5), d(2,8)) + b(d(0,8), a(0,8)) \]
And next: \[ s''' = H(8, s'') = a(0, b(c(1,5) + 2,5 + 3,5), d(2,8)) + b(d(0,8), a(0,8)) = s''. \]

So we have: \[ s'''' = H(8, s''') = s'''' = H(8, s''') = s'', etc. \]

Finally the fixed point will be computed as the following:
\[ FP_R = a(0, b(c(1,5) + 2,5 + 3,5), d(2,8)) + b(d(0,8), a(0,8)) \]

Fixed point tree automata \( A_{1, b} \) is depicted in the figure 1.b. We have \( FP_R \subseteq L(A_{1, b}) \) since
\[ a(0, b(c(1,5), d(2,8))) \in L(A_{1, b}), \]
\[ a(0, b(c(2,5), d(2,8))) \in L(A_{1, b}), \]
\[ a(0, b(c(3,5), d(2,8))) \in L(A_{1, b}), \]
and since \[ b(d(0,8), a(0,8)) \in L(A_{1, b}) \]

The second question about termination proving will be discussed in the next section.

3.2. Termination proving

We consider in this section some researches about termination of TRS. This is an undecidable problem even if \( F, R \) finited. A rewrite relation that is also a (strict) partial order is called a rewrite order. An order \( \succ \) is called well-founded if there is no infinite descending sequence \( t_1 \succ t_2 \succ ... \).

A TRS \( \lessdot F, R \) is compatible with a rewrite order \( \succ \) on \( T(F, V) \) if \( l \succ r \) for every rewrite rule \( l \rightarrow r \) of \( R \). It is easy to show that a TRS is terminating if and only if it is compatible with a well-founded rewrite order.

Example 2 We come back to example 1: Let \( \gg \) be an order on \( F \): \( H \gg a \gg b \gg c \gg d \gg 3 \gg 2 \gg 1 \gg 0 \gg \$ \).

This is well-known that \( H(\$, a(x,y)) \succ_{rpo} a(x, H(\$, y)) \) since:
\[ H \gg a, H(\$, a(x,y)) \succ_{rpo} x \] and
Tree Data Decision Diagrams

\[ H(\$, a(x, y)) \succ_{rpo} H(\$, y) \text{ since } H = H \text{ and } \{\$, a(x, y)\} \succ_{rpo} \{\$, y\}, \ldots \]

So we can say that \( r_4 \) is compatible with \( \succ_{rpo} \).

Similarly, for other rules, \( R \) is compatible with \( \succ_{rpo} \), i.e. \( \triangle_R \) is simply terminating.

It should be noted that the above order is not unique, e.g. another one can be: \( H \succ d \succ c \succ b \succ a \succ 0 \succ 1 \succ 2 \succ 3 \succ \$ \). Also notice that we sometime can not find any order. In this case, we try to search a map \( \pi_f \) like semantic labelling technique presented at the end of section 2.1 providing termination on \( \langle F_{lab}, R_{lab} \rangle \).

We do not discuss here the fixed point termination because fixed point termination proving is an undecidable problem, even harder than root reduction problem, there often exists a fixed point is not terminating while its root reduction termination is proved, e.g: let \( F(\$, x) \rightarrow a(\$, x), F^r(\$) = \$ + a(\$, \$) + a(a(\$, \$), a(\$, \$)) + a(a(a(\$, \$), a(\$, \$)), a(a(\$, \$), a(\$, \$))) \ldots \) will be infinite. Intuitively, we find that fixed point termination is a special termination problem in infinite system [10].

4. EXPERIMENTAL IMPLEMENTATIONS AND CASE STUDY

4.1. TRS and tree automaton representation

In this section, we focus on the dining philosophers problem for either DDD simulation or TDDD representation.

For DDD simulation, a single philosopher has left hand \('l'\) and right hand \('r'\) with their values are 0 and 1 (Corresponding with or without fork). DDD system is described as a sequence of \('l'\) and \('r'\), e.g. the DDD encoding of four philosophers system can be represented by:

\[ l(0, r(1, l(0, r(1, l(0, r(1, dollar))))))) \]

It should be noted that at the same moment, two neighbors never take the same fork, i.e. \( l(0, r(1, l(0, r(1, l(0, r(1, dollar))))))) \) is not an accepted configuration.

And DDD TRS simulates the homomorphisms of DDD (Synchronization between two philosophers, the left hand of \( i^{th} \) and the right hand of \( i+1^{th} \)):

\begin{align*}
\text{Init:} & \quad H(\$, l(x, y)) \rightarrow l(x, y); \\
& \quad L(z, r(x, y)) \rightarrow l(z, R(x, y)); \\
\text{Synchronize:} & \quad R(z, l(x, y)) \rightarrow r(0, L(1, y)); \\
& \quad R(z, l(x, y)) \rightarrow r(0, L(0, y)); \\
& \quad R(z, l(x, y)) \rightarrow r(1, L(0, y)); \\
\text{End of configuration handle:} & \quad R(z, \$) \rightarrow r(z, \$); \\
& \quad L(z, \$) \rightarrow l(z, \$); \\
\end{align*}

Synchronization between philosopher \( n^{th} \) and \( 1^{st} \) (They are neighbors in the round table):

\begin{align*}
\text{Change and memorize the left hand of first philosopher:} & \quad H(\$, l(x, y)) \rightarrow r(0, M(0, y)); \\
& \quad H(\$, l(x, y)) \rightarrow r(1, M(1, y)); \\
\text{Go down until the last philosopher:} & \quad M(z, l(x, y)) \rightarrow l(x, M(z, y)); \\
& \quad M(z, r(x, y)) \rightarrow r(x, M(z, y)); \\
& \quad N(z, l(x, y)) \rightarrow l(x, N(z, y)); \\
& \quad N(z, r(x, y)) \rightarrow r(x, N(z, y)); \\
\text{Synchronize with the right hand of last philosopher:} & \quad M(0, r(x, \$)) \rightarrow l(0, \$); \\
& \quad M(0, r(x, \$)) \rightarrow l(1, \$); \\
& \quad N(1, r(x, \$)) \rightarrow l(0, \$); \\
& \quad N(1, r(x, \$)) \rightarrow l(1, \$); \\
\end{align*}

This TRS is not proved terminating on account of the cycle below (i.e we can not determine whether \( L \gg R \) or \( R \gg L \)):

\begin{align*}
& \quad L(z, r(x, y)) \rightarrow l(z, R(x, y)); \\
& \quad R(z, l(x, y)) \rightarrow r(0, L(1, y)); \\
& \quad R(z, l(x, y)) \rightarrow r(0, L(0, y)); \\
& \quad R(z, l(x, y)) \rightarrow r(1, L(0, y)); \\
\end{align*}
But we can use semantic labelling with labels are order of philosophers in the system, e.g.:

\[
\begin{align*}
\text{Li} (z, r(x, y)) &\rightarrow l(z, \text{Ri} (x, y)) \\
\text{Ri} (z, l(x, y)) &\rightarrow r(0, \text{L}(i+1)(1, y)) \\
\text{Ri} (z, l(x, y)) &\rightarrow r(0, \text{L}(i+1)(0, y)) \\
\text{Ri} (z, l(x, y)) &\rightarrow r(1, \text{L}(i+1)(0, y))
\end{align*}
\]

with \( i = 1, \ldots, n \) and \( \gg \) be an order on \( F : H \gg L_1 \gg R_1 \gg L_2 \gg R_2 \gg \ldots \gg L_n \gg R_n \gg M \gg N \gg 1 \gg 0 \gg \$.

This TRS is terminating owing to the compatibility with RPO. When a model is proved terminating (called well-designed), the model-checker will generate all of reachability states if the system resources are enough.

We have also built a TDDD representation for dining philosophers problem. Hierarchical model contains:

- \('s'\) is a philosophers group (having a different number of philosophers, 's' higher for group more philosophers)
- \('p'\) is a philosopher having left hand and right hand with their values 0 and 1 (Corresponding with or without fork).

For example, a system of four philosophers is like follow (The first 's' presents the group having four philosophers while two others in its sub-terms describe the group of two philosophers):

\[
\begin{align*}
s(s(p(1,0), p(1,0)), s(p(1,0), p(0,0)))
\end{align*}
\]

TDDD rewriting rule set 1:

**Synchronization between two philosophers \(i^{th}\) and \(i+1^{th}\):**

\[
\begin{align*}
\text{H } ( \text{dollar}, s (x, y) ) &\rightarrow A (x, y) \\
\text{A} (s(x, y), s(u, v)) &\rightarrow s(\text{A} (x, y), s(u, v)) \\
\text{A} (s(x, y), s(x, y)) &\rightarrow s(s(x, y), \text{A} (u, v)) \\
\text{A} (p(x, y), p(u, v)) &\rightarrow s(p(x, 0), p(0, v)) \\
\text{A} (p(x, y), p(u, v)) &\rightarrow s(p(x, 1), p(0, v)) \\
\text{A} (p(x, y), p(u, v)) &\rightarrow s(p(x, 0), p(1, v))
\end{align*}
\]

TDDD rewriting rule set 2:

**Synchronization between philosopher \(n^{th}\) and \(1^{st}\):**

\[
\begin{align*}
\text{H } ( \text{dollar}, s (x, y) ) &\rightarrow B (x, y) \\
\text{B} (s(w, s(u, v)), s(x, y)) &\rightarrow s(\text{B} (w, s(u, v)), s(x, y)) \\
\text{B} (s(w, s(u, v)), s(x, y)) &\rightarrow s(s(w, s(u, v)), \text{B} (x, y)) \\
\text{B} (s(w, s(u, v)), s(x, y)) &\rightarrow s(s(w, s(u, v)), s(x, y)) \\
\text{B} (s(w, p(u, v)), s(x, y)) &\rightarrow s(s(w, p(u, 0)), s(p(0, y), x)) \\
\text{B} (s(w, p(u, v)), s(x, y)) &\rightarrow s(s(w, p(u, 1)), s(p(0, y), x)) \\
\text{B} (s(w, p(u, v)), s(x, y)) &\rightarrow s(s(w, p(u, 0)), s(p(1, y), x))
\end{align*}
\]

International Workshop on Verification and Evaluation of Computer and Communication Systems  
VECoS 2008  
8
Let \( \gg \) be an order on \( F \): \( H \gg A \gg C \gg D \gg G \gg R \gg L \gg M \gg N \gg U \gg V \gg s \gg p \gg 1 \gg 0 \gg \$.

This TRS is terminating because of the compatibility with RPO. As we previously mentioned, this model is well-designed, the model-checker will generate all of reachability states if the system resources are enough.

Tree automaton representation of fixpoint \( FP_R \) with 18 states for 81 reachability configurations (In other words, term set \( FP_R \) contains 81 different terms where each term corresponds to a system configuration) is shown in the figure 2. We can recognize the compact data from the sharing states.

![Figure 2: Fixpoint of Dining Philosophers with n = 4](image)

### 4.2. Result analysis

We aim at comparing DDD and TDDD in the same data structure and test them in the same environment. The code is written in JAVA and ran in a normal PC 1.7GHz, 1G RAM.

First, we prove successfully the termination of models for DDD and TDDD, i.e. the model-checker either for DDD or TDDD will return all of reachability states if the system resources are enough.

The result in the table shows that TDDD has a better sharing internal data structure than DDD owing to the hierarchical architecture. So DDD has a smaller time complexity but it fails rapidly when the size of test models increases exponentially.
The TRS system design is very important. Because of the redundant rules elimination and our others experiences in implementation, we can reduce to an equivalent rewriting rule set 2 with 
\[ R = 19 \]
and 
\[ V = \{ x, y \} \] (Instead of rewriting rule set 1 with 
\[ R = 37 \]
and 
\[ V = \{ x, y, u, v \} \]):

Synchronization between two philosophers \( i^{th} \) and \( i + 1^{th} \):

\[
\begin{align*}
H (\text{dollar}, x) & \to A (\text{dollar}, x); \\
A (\text{dollar}, s (x, y)) & \to s (G (\text{dollar}, x), D (\text{dollar}, y)); \\
A (\text{dollar}, s (x, y)) & \to s (L (\text{dollar}, x), R (\text{dollar}, y));
\end{align*}
\]

Synchronization between two philosophers \( n^{th} \) and \( 1^{st} \):

\[
\begin{align*}
H (\text{dollar}, x) & \to B (\text{dollar}, x); \\
B (\text{dollar}, s (x, y)) & \to s (B (\text{dollar}, x), y); \\
B (\text{dollar}, s (x, y)) & \to s (x, B (\text{dollar}, y)); \\
B (\text{dollar}, s (x, y)) & \to s (R (\text{dollar}, x), L (\text{dollar}, y)); \\
B (\text{dollar}, s (x, y)) & \to s (D (\text{dollar}, x), G (\text{dollar}, y));
\end{align*}
\]

And the sharing primitive rule:

\[
\begin{align*}
L (\text{dollar}, s (x, y)) & \to s (L (\text{dollar}, x), y); \\
R (\text{dollar}, s (x, y)) & \to s (x, R (\text{dollar}, y)); \\
L (\text{dollar}, p (x, y)) & \to p (V (\text{dollar}, x), y); \\
R (\text{dollar}, p (x, y)) & \to p (x, U (\text{dollar}, y)); \\
G (\text{dollar}, s (x, y)) & \to s (G (\text{dollar}, x), y); \\
D (\text{dollar}, s (x, y)) & \to s (x, D (\text{dollar}, y)); \\
G (\text{dollar}, p (x, y)) & \to p (U (\text{dollar}, x), y); \\
D (\text{dollar}, p (x, y)) & \to p (x, V (\text{dollar}, y)); \\
U (\text{dollar}, x) & \to 1; \\
U (\text{dollar}, x) & \to 0; \\
V (\text{dollar}, x) & \to 0;
\end{align*}
\]

Let \( \gg \) be an order on \( F \): 
\[ H \gg A \gg B \gg L \gg R \gg G \gg D \gg U \gg V \gg 1 \gg 0 \gg \$. 
This TRS is terminating on account of the compatibility with RPO.

<table>
<thead>
<tr>
<th>Automata</th>
<th>TDDD with RuleSet 1</th>
<th>TDDD with RuleSet 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>States</td>
<td>Time(s)</td>
</tr>
<tr>
<td>4</td>
<td>18</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>26</td>
<td>59</td>
</tr>
<tr>
<td>16</td>
<td>34</td>
<td>407</td>
</tr>
<tr>
<td>32</td>
<td>45</td>
<td>2718</td>
</tr>
<tr>
<td>64</td>
<td>60</td>
<td>59724</td>
</tr>
<tr>
<td>128</td>
<td>121</td>
<td>-</td>
</tr>
</tbody>
</table>

Obviously, the reduction of rule set and variable set (RuleSet 2) brings the better time complexity than RuleSet 1, we call it informally the better-designed among well-designed models. It should be noted that tree automaton is the same for both RuleSet 1 or RuleSet 2, that can be explained by the fact that the better-designed improves only the time complexity but it can not change the space complexity.

In fact, a model-checker of distributed systems verification [7] based on a formal language LfV (Language for Verification) [6] using DDD in MORSE [5], an industry and academic cooperation RNTL project can compute the local fixpoints generating the result more rapid.

Actually, the global fixpoint in TRS increases the time complexity of TDDD, each step we must start the fixpoint computation from root of the automaton and move downward by storing a large temporary results. The integration of
local fixpoints technique (like in DDD and SDD) into TDDD will progress the time complexity of this model-checker in this case.

5. CONCLUSION

We proposed a new extension of DDD for symbolic verification in TRS. It is not only compact as SDD (more than DDD) but its operators are also very simpler and more flexibly than the others on account of TRS interface.

Moreover, we prove successfully the termination of models for DDD and TDDD in some particular cases and support model design orientation for developers. On the other hand, presenting terms by tree automata technique provides the capability to maintain the internal representation of data in canonical form though the time complexity of the factorization is still a challenge.

For future works, first, we aim at integrating into TDDD the local fixpoints technique which is implemented successfully in DDD and SDD to improve the time complexity of TDDD model-checker. Finally, we plan to develop a TDDD model-checker for the infinite TRS $< F, R >$. In this case, we must use minimization algorithm for non-deterministic tree automaton like bisimulation minimization [14, 15].

REFERENCES