Calculating τ-Confluence Compositionally*

Gordon J. Pace[1], Frédéric Lang[2], Radu Mateescu[2]

[1] University of Malta (gordon.pace@um.edu.mt)
[2] INRIA Rhône-Alpes, France ({Frederic.Lang, Radu.Mateescu}@inrialpes.fr)

Abstract. τ-confluence is a reduction technique used in enumerative model-checking of labeled transition systems to avoid the state explosion problem. In this paper, we propose a new on-the-fly algorithm to calculate partial τ-confluence, and propose new techniques to do so on large systems in a compositional manner. Using information inherent in the way a large system is composed of smaller systems, we show how we can deduce partial τ-confluence in a computationally cheap manner. Finally, these techniques are applied to a number of case studies, including the rel/REL atomic multicast protocol.

1 Introduction

An important area of research in model checking is the generation of restricted models using intuition and insight in the system in question to produce smaller state spaces — small enough to enumerate and manipulate. In practice, different techniques have been developed. Of interest to this paper, we note: *on-the-fly* model generation, where only the 'interesting' part of the model is generated; *partial-order reduction* [14] and the related *τ-confluence* [8, 17] reduction techniques which exploit independence of certain transitions in the system to discard unnecessary parts; and *compositional techniques* [7] where a model is decomposed into smaller parts, partially generated using knowledge about future interface components to avoid intermediate explosion.

In this paper we are mainly interested in deriving techniques which use structural information of the system to perform τ-confluence reduction. Extracting general τ-confluence of a flattened system can be costly and impractical. However, the user usually also provides the system in the form of a symbolic description, which we attempt to exploit at a low cost to calculate τ-confluence. The information we use is the connection pattern of the network of communicating transition systems — composition expressions. At the leaves, we have transition system components, usually various magnitudes of size smaller than the whole system (especially if techniques such as projection [11] are first applied). Using the structure of the network, we can immediately deduce certain independence between transitions to be used for model reduction. We propose a new algorithm

* This work has been partially supported by the European Research Consortium in Informatics and Mathematics (ERCIM).

to calculate partial τ-confluence on-the-fly — similar in spirit to [2], but optimised in particular for flat transition systems. We then prove correct a number of laws which allow us to deduce τ-confluence in a composition expression without the need of expensive calculations and show its performance when applied on a number of case studies, including the rel/REL atomic multicast protocol.

2 Related Work

An extensive and thorough study of τ-confluence in process algebrae and LTS verification can be found in [8]. In [17], the results are developed further, extending weak confluence conditions for divergent transition systems.
The ideas we develop in this paper heavily borrow from [9], in which a global (not on-the-fly) algorithm is given for calculating maximal τ-confluence sets. The algorithmic complexity is of the order $O(m * \mathit{fanout}_\tau^3)$, where m is the number of transitions in the LTS, and fanout_τ is the maximum number of τ transitions exiting from a state. The paper also uses τ-prioritisation and τ-compression (where chains of τ transitions are replaced by a single one), used to reduce a LTS, once a τ-confluence set has been calculated. We use the same notion of τ-confluence as in this paper mainly since discovering τ-confluence sets under this definition is well-tractable. Our alternative algorithm to evaluate a maximal τ-confluence set has complexity of the order $O(m*\mathit{fanout}*\mathit{fanout}_\tau)$ and works on-the-fly. Furthermore, our τ-confluence detection algorithm works equally well for LTSs which are divergent, and we use deduction to partially identify τ-confluence in large systems by analysing their components.
[3, 2] build upon the results of [17] and are closely related to our work, except that they concentrate on weak confluence. The algorithms work equally well with the stronger confluence condition we use. To calculate a τ-confluent set, they use the symbolic description of the LTS (as guarded action/event systems) and feed conditions to an automated theorem prover to prove the independence of certain guards. In a certain sense, our algorithm to calculate the maximal τ-confluent set can be seen as an extreme case of this approach — the LTS expanded to the actual description of the LTS transitions, and given the trivial nature of the resulting guards and transitions, we replace the automated theorem prover by a BES solver. Our symbolic description, based on composition expressions, differs from theirs, and allows for certain independence to be concluded easily, but does not allow symbolic reduction as is possible in their case.
τ-confluence is closely connected to partial-order reduction techniques [14]. The fact that τ transitions are 'partially' invisible under branching and other weak bisimulations, means that independence of τ transitions preserving bisimulation is possible, and can be useful in practice. In [16] is an analysis of partial-order methods applied to process algebrae, that includes a set of conditions sufficient to guarantee branching bisimulation equivalence after reduction. As remarked in [2], these conditions are stronger than weak τ-confluence. The conditions are not comparable to the notion of partial confluence we use, since we allow for confluence, but closing up to one step ahead. [16] allows for multiple invisible

transitions, but not for confluence. The conditions, however, closely relate to the conditions used in this and other τ-confluence papers.
Several partial-order reduction techniques applied to compositions of LTSs have been proposed. Of interest are the τ-diamond elimination technique presented in [4] (implemented for CSP in the FDR 2 tool) and a technique based on the detection of so-called τ-inert transitions presented in [15] (implemented for CCS in the Concurrency Factory). Both consist in identifying τ-transitions that do not need be interleaved with concurrent transitions, since the obtained behaviour would be equivalent (for some relation) to the one in which the τ-transition is taken first. The difference relies on the properties being preserved under bisimulation in the case of behaviour equivalence preserved under reduction: weak bisimulation in the case of [15], and failure/divergence in the case of [4]), both of which do not preserve branching properties of the system. Additionally, our approach works on-the-fly, in combination with any verification tool of CADP, and for any language with a front-end for CADP.

3 Basic Definitions

Definition: A *labeled transition system* (LTS) is a quadruple $\langle Q, Act, \rightarrow, q0 \rangle$ where Q is the set of *states* of the system, Act is the set of possible *actions* the system may take (including a special invisible action τ), $\rightarrow \subseteq Q \times Act \times Q$ is the set of *transitions* and $q0 \in Q$ is the *initial state* of the system.
Using standard conventions, we will write $q \xrightarrow{a} q'$ to say that $(q, a, q') \in \rightarrow$, and for a set of actions $G \subseteq Act$, $\xrightarrow{G}$ is the transition relation $\rightarrow$ restricted to actions in G. $actions(q)$ is the set of actions possible from state q. If we may want to 'ignore' invisible transitions, $q \xrightarrow{\overline{a}} q'$, means that either $q \xrightarrow{a} q'$, or $q = q'$ and $a = \tau$ (note that this case does not necessarily imply that $q \xrightarrow{\tau} q'$). $\xrightarrow{\tau^*}$ is the reflexive transitive closure of $\xrightarrow{\{\tau\}}$. Finally, we say that an LTS is *divergent* if there exists an infinite sequence of states q_i such that for all i, $q_i \xrightarrow{\tau} q_{i+1}$.
Definition: Given two LTSs S_1 and S_2 ($S_i = \langle Q_i, Act, \rightarrow_i, q0_i \rangle$) a relation between the states of the two LTSs $\simeq \subseteq Q_1 \times Q_2$ is said to be a *branching bisimulation* if for any $q_1 \simeq q_2$, the following two properties are satisfied:

1. for any $q_1 \xrightarrow{a} q_1'$, there exist q_2', q_2'' with $q_2 \xrightarrow{\tau^*} q_2' \xrightarrow{\overline{a}} q_2''$ and $q_1 \simeq q_2'$, $q_1' \simeq q_2''$.
2. for any $q_2 \xrightarrow{a} q_2'$, there exist q_1', q_1'' with $q_1 \xrightarrow{\tau^*} q_1' \xrightarrow{\overline{a}} q_1''$ and $q_1' \simeq q_2$, $q_1'' \simeq q_2'$.

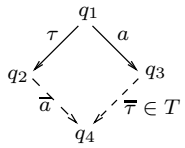
The maximal branching bisimulation is a well-defined equivalence relation ($\simeq_b$). We say that two LTSs are branching bisimilar ($S_1 \simeq_b S_2$) if their initial states are branching bisimilar $q0_1 \simeq_b q0_2$.

3.1 τ-Confluence

τ-confluence corresponds to the intuition that certain silent transitions do not change the set of transitions we can undertake now or in the future. If we can calculate a set of silent transitions with this property, we can then reduce the LTS to obtain a smaller system.

Different levels of τ-confluence have been defined in the literature. Some encompass more τ transitions (and hence allow more powerful reductions), but are more expensive to calculate an appropriate confluent set. Others are more restrictive, but allow cheap τ-confluence set deduction. In this paper we will concentrate on so-called *strong confluence* which we will refer to in the rest of the paper simply as confluence. The interested reader is referred to [8, 17] for a whole hierarchy of τ-confluence notions.

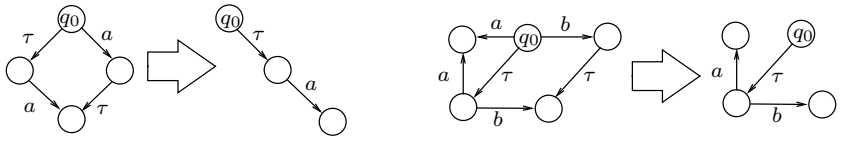
Definition: Given an LTS $S = \langle Q, Act, \rightarrow, q0 \rangle$, and $T \subseteq \stackrel{\{\tau\}}{\rightarrow}$, we say that T is *τ-confluent* in S if: for every $q_1 \stackrel{\tau}{\rightarrow} q_2 \in T$ and $q_1 \stackrel{a}{\rightarrow} q_3$, there exists a state q_4 such that $q_2 \stackrel{\overline{a}}{\rightarrow} q_4$ and $q_3 \stackrel{\overline{\tau}}{\rightarrow} q_4 \in T$.
The intuition is that every other outgoing transition of q_1 can be emulated after the τ-confluent transition. Graphically, the τ-confluence can be seen in the following figures. Normal line transitions are given (universally quantified), whereas dashed transitions indicate that their existence must be proved:



Proposition 1: If $q \stackrel{\tau}{\rightarrow} q'$ is a τ-confluent transition in S, then $q \simeq_b q'$.
Proposition 2: The union of two τ-confluent sets of an LTS S is itself a τ-confluent set of S. We call the union of all τ-confluent sets the *maximal τ-confluent set,* and write it as $\mathbb{T}(S)$.
The proofs of these propositions can be found in [9].

3.2 Reduction techniques

Definition: Given two LTSs S_1 and S_2 ($S_i = \langle Q_i, Act, \rightarrow_i, q0_i \rangle$), we say that S_2 is a *τ-prioritisation* of S_1 with respect to a τ-confluent set T, if (i) $\rightarrow_2 \subseteq \rightarrow_1$ and (ii) for every $q \stackrel{a}{\rightarrow}_1 q'$, either $q \stackrel{a}{\rightarrow}_2 q'$ or for some q'', $q \stackrel{\tau}{\rightarrow}_2 q'' \in T$.
The following figures show two examples of τ-prioritisation (with unreachable states removed):



Proposition 3: If S_1 is a τ-prioritisation of a non-divergent LTS S_2 with respect to T, then $S_1 \simeq_b S_2$.
The proof can be found in [9]. τ-prioritisation thus allows reduction of non-divergent systems with respect to a τ-confluent set, maintaining equivalence modulo branching bisimulation. The main problem with τ-prioritisation is that it is restricted to non-divergent systems. However, one can augment the prioritisation to calculate and eliminate on-the-fly τ cycles. Alternatively, other reduction techniques [3, 13] have been defined in the literature (see Section 2) and can be used.

4 Calculating τ-Confluence Using Boolean Equations

In this section, we present an on-the-fly algorithm to calculate the maximal τ-confluent set.
Definitional boolean equation systems without negation are a well-known and studied field. The following is a short resumé of definitions and results to set the picture for the translation algorithm we propose for on-the-fly τ-confluence calculation.

4.1 Boolean Equation Systems.

Definition: A *boolean equation system* (BES) is a set of variables V split into two disjoint subparts V_d and V_c, with their definition $\delta \in V \to 2^V$. Variables in V_d are seen as defined in terms of a disjunction over the definition set, while those in V_c as a conjunction:

Definition: An *interpretation* I of a BES is a subset of variables V of the equation system, $I \subseteq V$. Variable v is said to be *satisfied* in I if $v \in I$. An interpretation is said to be *valid* if the definition function holds:

$$(\forall v : V_d \,.\ \delta(v) \cap I \neq \emptyset) \ \wedge (\forall v : V_c \,.\ \delta(v) \subseteq I)$$

Proposition 4: The union of all valid interpretations of a BES Eq is itself a valid interpretation. This is called the *greatest fixed point solution*: $(\nu V.\ Eq)$.
Standard algorithms exist to evaluate the greatest fixed point of a boolean equation system. In particular, we are mainly interested in an on-the-fly algorithm — a local one resolving only the necessary variables we may require. Such algorithms can be found in [12, **?**] and work in both a breadth-first and depth-first fashion. This problem can be solved in time proportional to the number of variables and the size of the definition sets.

4.2 Translating τ-Confluence of LTSs into Boolean Equations.

It is rather straightforward to translate the definition of τ-confluence in Section 3.1 into a BES whose validity implies the confluence of individual τ transitions.

Definition: Given an LTS S, we introduce a conjunctive variable for every τ transition, and a disjunctive variable for every half-terminated diamond in the τ-confluence diagram:

$$V_c \stackrel{df}{=} \{c_{q_1,q_2} \mid q_1 \xrightarrow{\tau} q_2\}, \qquad V_d \stackrel{df}{=} \{d^a_{q_1,q_2,q_3} \mid q_1 \xrightarrow{\tau} q_2, q_1 \xrightarrow{a} q_3\}$$

The intuitive interpretation we will use is that (i) every confluent τ transition has to be able to close *all* half-diamonds (conjunction) and (ii) every half-diamond has to be closed by *some* other confluent τ transition (disjunction). The boolean variables c_{q_1,q_2} will be satisfiable if and only if $q_1 \xrightarrow{\tau} q_2$ is confluent, while $d^a_{q_1,q_2,q_3}$ is satisfiable if and only if the half-diamond can be satisfactorily closed.

Conjunctive variables: c_{q_1,q_2} should be satisfiable if and only if all extended half-diamonds which are not trivially closed (via a direct a transition from q_2 to q_3) can be closed:
$\delta(c_{q_1,q_2}) \stackrel{df}{=} \{d^a_{q_1,q_2,q_3} \mid q_1 \xrightarrow{\tau} q_2,\ q_1 \xrightarrow{a} q_3,\ q_2 \not\xrightarrow{a} q_3\}$

Disjunctive variables: $d^a_{q_1,q_2,q_3}$ is satisfiable if and only if there is some τ transition from q_3 to some q_4 which (i) closes the diamond, and (ii) is τ-confluent:
$\delta(d^a_{q_1,q_2,q_3}) \stackrel{df}{=} \{c_{q_3,q_4} \mid q_2 \stackrel{a}{\rightarrow} q_4, q_3 \stackrel{\tau}{\rightarrow} q_4\}$
$\delta(d^\tau_{q_1,q_2,q_3}) \stackrel{df}{=} \{c_{q_3,q_4} \mid q_2 \stackrel{\tau}{\rightarrow} q_4, q_3 \stackrel{\tau}{\rightarrow} q_4\} \cup \{c_{q_3,q_2} \mid q_3 \stackrel{\tau}{\rightarrow} q_2\}$
Note that in the case of $a = \tau$, the diamond may be closed as a triangle (see the figures depicting how τ-confluence diagrams can be closed.)

Proposition 5: The translation is sound and complete: Given a valid interpretation I of a translated LTS S, $\{q_1 \stackrel{\tau}{\rightarrow} q_2 \mid c_{q_1,q_2} \in I\}$ is a τ-confluent set (soundness), and for any τ-confluent set T, there is a valid interpretation I such that $I \cap V_c = \{c_{q_1,q_2} \mid q_1 \stackrel{\tau}{\rightarrow} q_2 \in T\}$ (completeness).
From this proposition, it then follows that:

Theorem 1: Calculating the greatest fixed point of the BES obtained by translating an LTS gives the maximal τ-confluent set.

4.3 Complexity.

Consider variables V_c. We have m_τ (the number of τ transitions) such variables. Furthermore, the definition set of each variable is bounded above by $\mathit{fanout} * \mathit{fanout}_\tau$ (fanout is the maximum number of successors of a state in the LTS, fanout_τ is the maximum number of τ-successors). Now consider the disjunctive variables V_d. We have $m_\tau * \mathit{fanout}$ such variables (for each τ transition, we have an entry for each other transition which can be taken from the source node). The definition sets of these variables never exceeds fanout_τ entries.
Recall that a BES can be solved in time proportional to the number of variables plus the size of the definition sets. The complexity of resolving τ-confluence using our algorithm is thus $O(m_\tau \mathit{fanout} * \mathit{fanout}_\tau)$. This compares favourably with the algorithm given in [9] which has complexity $O(m_\tau * \mathit{fanout}^3_\tau)$.
However, this is pessimistic view of the complexity. Due to the regular nature of the equations (conjunctions of disjunctions), and the fact that we also know that the disjunctive variables are never reused (a disjunctive variable is revisited only through a conjunctive one), we can hone the algorithm to work more efficiently (for example, by not caching disjunctive variables).

5 Definitions and Basic Results: Composition Expressions

Definition: A *composition expression* is an object in the abstract type *Exp*:

$$\mathit{Exp} \ ::= \ \text{LTS} \mid \texttt{hide}\ G\ \texttt{in}\ \mathit{Exp} \mid \mathit{Exp} \parallel_G \mathit{Exp}$$

The basic building blocks are LTSs, together with the hiding operator (renames any label in the action set G to τ) and synchronous composition (actions in G are synchronised, the rest can happen independently). One can add other operators to this family, but these usually suffice for a decomposed view of a system.
A composition expression describes the way a family of LTSs communicate together, but can be seen itself as an LTS.

Hiding: In ($\texttt{hide}\ G\ \texttt{in}\ E_1$), if E_1 is the LTS $\langle Q_1,\ Act_1,\ \rightarrow_1,\ q0_1\rangle$, the resultant LTS is $\langle Q_1,\ Act_1,\ \rightarrow,\ q0_1\rangle$ where $\rightarrow$ is the smallest relation generated by the following structured operational semantics rules:

$$\frac{q_1 \xrightarrow{a}_1 q_1',\ a \notin G}{q_1 \xrightarrow{a} q_1'} \qquad \frac{q_1 \xrightarrow{a}_1 q_1',\ a \in G}{q_1 \xrightarrow{\tau} q_1'}$$

Synchronous composition: In the composition expression ($E_1 \parallel_G E_2$), if E_i corresponds to the LTS $\langle Q_i,\ Act_i,\ \rightarrow_i,\ q0_i\rangle$, the resultant LTS is $\langle Q_1 \times Q_2,\ Act_1 \cup Act_2,\ \rightarrow,\ (q0_1, q0_2)\rangle$ where $\rightarrow$ is the smallest relation generated by the following structured operational semantics rules:

$$\frac{q_1 \xrightarrow{a}_1 q_1',\ a \notin G}{(q_1,q_2) \xrightarrow{a} (q_1',q_2)} \qquad \frac{q_2 \xrightarrow{a}_2 q_2',\ a \notin G}{(q_1,q_2) \xrightarrow{a} (q_1,q_2')} \qquad \frac{q_1 \xrightarrow{a}_1 q_1',\ q_2 \xrightarrow{a}_2 q_2',\ a \in G}{(q_1,q_2) \xrightarrow{a} (q_1',q_2')}$$

For the sake of brevity, in contexts where we speak of expressions, unless otherwise stated, the LTS generated by expression E will be $\langle Q,\ Act,\ \rightarrow,\ q0\rangle$, and that of expression E_i will be $\langle Q_i,\ Act_i,\ \rightarrow_i,\ q0_i\rangle$.

Definition: The subterm relation over composition expressions $\sqsubseteq$ is defined to be the reflexive, transitive closure of the smallest relation $\sqsubset_1$ satisfying:

$$E \sqsubset_1 \texttt{hide}\ G\ \texttt{in}\ E,\ E \sqsubset_1 E \parallel_G E',\ E \sqsubset_1 E' \parallel_G E$$

We say that a transition $q \xrightarrow{a} q'$ of E is *immediately generated from* a transition $q_1 \xrightarrow{a_1}_1 q_1'$ of E_1 ($E_1 \sqsubset_1 E$) if the derivation of the former transition using the operational semantic rules requires the use of the latter. Thus, for example, $q_1 \xrightarrow{\tau} q_2$ in ($\texttt{hide}\ a\ \texttt{in}\ E$) is immediately generated from $q_1 \xrightarrow{a} q_2$ in E.

We are mainly interested in the transitive closure of this relation: $\uparrow^{E_2}_{E_1} \subseteq \rightarrow_1 \times \rightarrow_2$ (where $E_1 \sqsubseteq E_2$), which relates transitions in $\rightarrow_2$ (of E_2) with the transitions in $\rightarrow_1$ (of E_1) contributing to their generation.

For this definition to make sense, we will make the simplifying assumption that an expression will not contain common subexpressions (all the leaf LTSs are different). This is done to simplify the presentation but can be easily remedied either by tagging different leaf nodes (different tag for every leaf) or by reasoning in terms of expression contexts.

The decomposition law states that if $E_1 \sqsubseteq E_2 \sqsubseteq E_3$: $\uparrow^{E_2}_{E_3} \circ \uparrow^{E_1}_{E_2} = \uparrow^{E_1}_{E_3}$ (where $r \circ s$ is the relation composition of r and s).

Similarly, we can talk about a transition generating another: $t \downarrow^{E_1}_{E_2} t'$ means that t is *a generator of* t'. $\downarrow^{E_1}_{E_2}$ is simply the inverse of $\uparrow^{E_1}_{E_2}$.

We define relation application as usual: $R(X) \stackrel{df}{=} \{y \mid \exists x \in X\ .\ x\ R\ y\}$.

Definition: Given a composition expression E, the actions hidden above, and synchronised above a subexpression E_1 are defined as:

$$\mathit{Hidden}_E(E_1) \stackrel{df}{=} \bigcup\{G \mid \texttt{hide}\ G\ \texttt{in}\ E_2 \sqsubseteq E,\ E_1 \sqsubseteq E_2\}$$

$$\mathit{Synchronised}_E(E_1) \stackrel{df}{=} \bigcup\{G \mid E_2 \parallel_G E_3 \sqsubseteq E,\ E_1 \sqsubseteq E_2 \vee E_1 \sqsubseteq E_3\}$$

Given $E_1 \sqsubseteq E$, we define $\mathit{Tau}_E(E_1)$ to be the set of labels such that transitions in E_1 whose label appears in $\mathit{Tau}_E(E_1)$ are guaranteed to be transformed into τ transitions in E:

$$Tau_E(E_1) \stackrel{df}{=} Hidden_E(E_1) \setminus Synchronised_E(E_1)$$

Proposition 6: Given $E_1 \sqsubseteq E$, every transition labelled by $Tau_E(E_1)$ generates at least one τ transition, and nothing but τ transitions:

$$\forall t : \stackrel{Tau_E(E_1)}{\rightarrow} \ . \quad \uparrow^E_{E_1}(t) \neq \emptyset \wedge \ \uparrow^E_{E_1}(t) \subseteq \stackrel{\{\tau\}}{\rightarrow}$$

Proof: The proof follows from structural induction with the inductive hypothesis that in expression E_2 ($E_1 \sqsubseteq E_2 \sqsubseteq E$), a non-empty set of $\{\tau\} \cup Tau_E(E_1)$ transitions generates a non-empty set of $\{\tau\} \cup Tau_E(E_2)$ transitions.
Furthermore, since by definition, $Tau_E(E) = \emptyset$, the conclusion follows. □

We will write the expression obtained by replacing in E the occurrence of sub-expression E_2 by E_1 as $E[E_1/E_2]$.

Proposition 7: Branching bisimilarity is preserved in composition expressions: If $E_1 \sqsubseteq E$ and $E_1 \simeq_b E_2$ then $E \simeq_b E[E_2/E_1]$.

Proposition 8: Actions in $Tau_E(E_1)$ can be hidden immediately in E_1. Given $E_1 \sqsubseteq E$ and $G \subseteq Tau_E(E_1)$: $E[\texttt{hide } G \texttt{ in } E_2/E_1] \simeq_b E[E_2/E_1]$.

Consider a τ transition in a leaf LTS, which is not confluent. Just by looking at the leaf in question, we can sometimes deduce that the transition can never become confluent. Transitions about which we cannot guarantee this will be called *potential* τ-confluent transitions. We identify a set of transitions which we will later prove that all τ transitions generated higher up in the expression tree, will be generated by transitions in this set.

The intuition is the following: a transition is potentially confluent if (i) either it is already invisible, or its action will be hidden higher up in the expression tree, (ii) hidden, it satisfies the τ-confluence conditions on all other outgoing transitions except (iii) it may not satisfy the τ-confluence conditions with respect to transitions which may later disappear (synchronised above).

Definition: Given $E_1 \sqsubseteq E$, $P_1 \subseteq \stackrel{G}{\rightarrow}_1$ (where $G = Hidden_E(E_1) \cup \{\tau\}$) is said to be a *potential τ-confluence set* if, for all $q_1 \stackrel{a}{\rightarrow}_1 q_2 \in P_1$ and $q_1 \stackrel{b}{\rightarrow}_1 q_3$ with $b \notin Synchronised_E(E_1)$, then either $q_3 \stackrel{a?}{\rightarrow}_1 q_2 \in P_1$ or there exists q_4 such that $q_3 \stackrel{a?}{\rightarrow}_1 q_4 \in P_1$ and $q_2 \stackrel{b?}{\rightarrow}_1 q_4$. $q \stackrel{a?}{\rightarrow} q'$ is defined as $q \stackrel{a}{\rightarrow} q' \vee (a \in G \wedge \exists a' \in G \ . \ q \stackrel{\overline{a'}}{\rightarrow} q')$.

Proposition 9: The union of all potential τ-confluence sets of E_1 with respect to E ($E_1 \sqsubseteq E$) is itself a potential τ-confluence set. We call this the *maximal potential τ-confluence set* and write it as $\mathbb{P}_E(E_1)$.

Proposition 10: If T is a τ-confluent set of E_1 ($E_1 \sqsubseteq E$), T is also a potential τ-confluence set of E_1 with respect to E.

Proof: Consider $q_1 \stackrel{\tau}{\rightarrow} q_2 \in T$. Since it is a τ-confluent transition, for any $q_1 \stackrel{a}{\rightarrow} q_3$, there exists q_4 such that $q_2 \stackrel{\overline{a}}{\rightarrow} q_4$ and $q_3 \stackrel{\overline{\tau}}{\rightarrow} q_4 \in T$. Consider the different cases from the $\overline{a}$ and $\overline{\tau}$: (i) $a = \tau$, $q_2 = q_4$, $q_3 = q_4$ (ii) $a = \tau$, $q_2 = q_4$, $q_3 \stackrel{\tau}{\rightarrow} q_2 \in T$ (iii) $q_2 \stackrel{a}{\rightarrow} q_4$, $q_3 = q_4$ (iv) $q_2 \stackrel{a}{\rightarrow} q_4$, $q_3 \stackrel{\tau}{\rightarrow} q_4 \in T$. These satisfy the property required of potential τ-confluence. T is thus a potential τ-confluence set. □

Proposition 11: If $E_1 \sqsubseteq E$, then $\mathbb{T}(E_1) \subseteq \mathbb{P}_E(E_1)$.

Proof: The proof follows immediately from propositions 2, 9 and 10. □

6 Calculating τ-Confluence in Composition Expressions

We now give a number of results to deduce τ-confluence in composition expressions without applying the algorithm on the top-level LTS, which can be very large.

6.1 Discovering τ-confluence in composition expressions.

The basic result we will apply to reduce composition expressions, is that τ-confluent transitions can only generate τ-confluent transitions. This can be very useful, especially if the leaf LTSs are reduced using τ-prioritisation, where in the resultant LTS, the τ-confluent transitions become the only transitions leaving a state, making them trivially recognisable as τ-confluent ones.

Theorem 2: If T_1 is a τ-confluent transition set of E_1 ($E_1 \sqsubseteq E$) then $\uparrow^E_{E_1}(T_1)$, the set of transitions of E generated from T_1, is a τ-confluent transition set of E.

This theorem together with the reduction techniques given in Section 3 provides us with two approaches to reduce an LTS in a compositional manner. One way is to calculate and label confluent transitions in the leaves, and use this information to deduce a confluence set in the top level LTS and perform reduction on-the-fly as the top level LTS is generated (using either τ-prioritisation or any other technique). Another approach is to reduce the leaves using maximal τ-prioritisation (leaving only one confluent outgoing transition, when one is available), thus making sure that as the top level LTS is generated, confluent transitions in the leaves are easily recognisable (unique τ transitions leaving a state) and use this information to generate the reduced LTS. The latter has the advantage that confluence information needs not be stored.

6.2 Doing more than τ transitions.

One way in which new τ-confluence can manifest itself is via new τ transitions appearing from the *hide* operator. In general, we cannot just treat transitions which are eventually hidden as invisible transitions, because if they are synchronised before being hidden, they may disappear due to the other branch not complementing the required transition. In the case of hidden transitions which are not synchronised, we can either push the *hide* operator into the expression to generate τ transitions as early as possible, or treat them as invisible transitions (despite the fact that they are not τ transitions). The second solution is preferable, since it does not destroy the structure of the expression as given by the user, and avoids adding new expression nodes, resulting in slower analysis. The following result justifies their treatment analogous to τ transitions.

Theorem 3: Given $E_1 \sqsubseteq E$ and $T_1 \subseteq \overset{Tau_E(E_1)}{\rightarrow_1}$ which satisfies the confluence conditions if replaced by τ transitions, and E_2, a τ-prioritisation of E_1 with respect to T_1, then $E[E_2/E_1] \simeq_b E$.

6.3 Some τ transitions are not worth the bother.

Finally, we can not only identify transitions which are, and will remain confluent, but also ones which can under no circumstances become confluent. Since within composition expressions we can only partially identify τ-confluent transitions, we may want to apply τ-confluence algorithm at the top-most level once again. If certain transitions can be identified as certainly not being τ-confluent during the expression tree traversal, we can apply the τ-confluence detection algorithm on a smaller set of transitions. The main theorem in this section allows us to do precisely this by using the notion of potential τ-confluence.

Lemma 1: If P_2 is a potential τ-confluent set of E_2 with respect to E ($E_1 \sqsubseteq E_2 \sqsubseteq E$) then $\downarrow_{E_1}^{E_2}(P_2)$ is a potential τ-confluent set of E_1 with respect to E.

Lemma 2: If $E_1 \sqsubseteq E_2 \sqsubseteq E$, then $\mathbb{P}_E(E_2) \subseteq \uparrow_{E_1}^{E}(\mathbb{P}_E(E_1))$

Theorem 4: Some transitions need never be checked for confluence. If $E_1 \sqsubseteq E$:

$$\mathbb{T}(E) \cap (\rightarrow_1 \setminus \uparrow_{E_1}^{E}(\mathbb{P}_E(E_1))) = \emptyset$$

Proof: From lemma 2 and proposition 10 we can now conclude that:

$$\mathbb{T}(E) \subseteq \uparrow_{E_1}^{E}(\mathbb{P}_E(E_1))$$

from which the theorem directly follows. □

Thus, by identifying and marking the complement of the maximal potential τ-confluent set in the leaf nodes, we can mark transitions which they generate at higher levels in the expression tree. Using this theorem, we are guaranteed that these transitions are not confluent, and we can thus reduce the computation required to identify a τ-confluent set of the LTS generated by the whole composition expression.

7 Tools and Applications

We have implemented the techniques described within the CADP toolkit [10] in the OPEN/CÆSAR environment. A collection of front-ends enable the compilation of source languages into C code which includes a function to access the LTS described by the system which is explored on-the-fly by the verification backends. EXP.OPEN is a front-end for composition expressions, while CÆSAR.OPEN is a front-end for the LOTOS language and GENERATOR is a back-end that explicitly generates the reachable state space of a system.

A variant of GENERATOR, named τ-CONFLUENCE, detects and prioritises τ-confluent transitions on-the-fly, using Boolean Equation Systems. EXP.OPEN has been extended to enable τ-confluence detection (**branching** option), by taking an account of the composition expression as stated in Theorems 2 and 3. More precisely, in global LTS of a composition expression E, EXP.OPEN prioritises the transitions that were detected as τ-confluent in the components of E. Additionally, some locally visible transitions are also prioritised, knowing that they will lead to τ-confluent transitions in the global LTS of E.

EXP.OPEN flattens the composition expression into a tuple of LTSs and a set of so-called *synchronization vectors*. If n is the size of the LTS tuple, each synchronization vector is a tuple of size $n + 1$, whose elements are either labels or a special *null* value. The first n elements represent labels of transitions that

must be fireable from the corresponding LTS current state components (none if element is null), whereas the last element (which must not be null) is the label of the resulting transition in the produced LTS. Working globally on the expression also allows us to identify certain locally confluent transitions which do not fall under the framework proposed in this paper. More details will be found in a related technical report. EXP.OPEN also calculates transitive closures of τ-confluent transitions (to avoid entering circuits of τ-confluent transitions), and hence compresses successive τ-confluent transitions into a single one.

These tools have been used to generate the state space of the rel/REL protocol previously studied in [5, 11]. The rel/REL protocol is an atomic multicast protocol between a transmitter and several receivers. This protocol is *reliable* in the sense that it allows arbitrary failures of the stations involved in the communication. The protocol guarantees the following two properties: (1) when a message M is sent by the transmitter, either every functioning station correctly receives M, or M is not received by any of the stations, and (2) messages are received in the same order as they are sent. Two underlying assumptions are needed to guarantee correctness: that crashed stations stop sending and receiving messages, and that functioning stations can always communicate with each other. The overall compositional structure of the system with two receivers is given by the following composition expression:

```
hide R_T1, R_T2, R1, R2, DEPOSE1, DEPOSE2 in
     CRASH_TRANSMITTER |{R_T1, R_T2}| (
        (RECEIVER_THREAD1 |{R_T1, R1, R2, GET, CRASH, DEPOSE1}| FAIL_RECEIVER1)
        |{R1, R2}|
        (RECEIVER_THREAD2 |{R_T2, R1, R2, GET, CRASH, DEPOSE2}| FAIL_RECEIVER2) )
```

The composition of LTSs `RECEIVER_THREAD`n and `FAIL_RECEIVER`n ($n = 1, 2$) defines the behaviour of receiver n, including the possibility of a crash. The LTS `CRASH_TRANSMITTER` describes the behaviour of the transmitter. These LTSs are generated from a LOTOS description of the system, detailed in [5].

In our experiments, performed using SVL scripts [6], we have compared two state-space generation approaches for the rel/REL protocol: (i) *Normal generation*: leaf LTSs and composition expression are generated normally, without optimisation (using respectively CÆSAR.OPEN/GENERATOR and EXP.OPEN/GENERATOR). (ii) *τ-prioritised generation*: leaf LTSs are generated using the CÆSAR.OPEN/τ-CONFLUENCE tools and composition expression is generated using EXP.OPEN `branching` together with GENERATOR.

Experiment results are displayed in Table 1. From these results, τ-prioritisation techniques on composition expressions seem very promising. Various reasons contribute to the success of τ-prioritisation. Although both `FAIL_RECEIVER`s are purely sequential, `RECEIVER_THREAD`s and `CRASH_TRANSMITTER` use parallel composition of processes performing silent transitions. This generates many τ-confluent transitions, which are detected by τ-confluence. Also, as a consequence of successful τ-prioritisation in three of the five leaves of the composition expression, EXP.OPEN avoids the creation of new τ-confluent diamonds. Additionally, a lot of transitions present in leaves are hidden at the top-level of the composition expression, some of which are confluent.

	Normal		τ-prioritised		Difference %	
	states	transitions	states	transitions	states	transitions
`CRASH_TRANSMITTER`	85	108	73	84	14%	22%
`RECEIVER_THREAD`n	16 260	167 829	16 260	115 697	0%	31%
`FAIL_RECEIVER`n	130	1 059	130	1 059	0%	0%

	Normal	τ-prioritised	Difference %
Number of states	249 357	114 621	54%
Number of transitions	783 470	220 754	72%
Exp.Open execution time	$2'23''$	$2'10''$	9%
Exp.Open memory consumption (Kb)	5 776	3 944	32%
Svl execution time	$3'05''$	$3'03''$	1%

Table 1. (a) Leaf Lts sizes using normal and τ-prioritised generation, (b) Cost of normal and τ-prioritised composition expression generation.

	Exp.Open		State Space	
Difference %	time	memory	states	trans
Alternating Bit(1)	9%	0%	4%	25%
Alternating Bit(2)	−4%	0%	6%	27%
Distributed Leader Election(1)	−57%	3%	11%	24%
Distributed Leader Election(2)	−21%	0%	12%	23%
Distributed Leader Election(3)	−88%	5%	5%	11%
Distributed Leader Election(4)	−90%	−1%	0%	8%
Distributed Leader Election(5)	−102%	−1%	0%	0%

Table 2. Difference ratios for several case studies

Note that applying τ-prioritisation at the top-level gives no further reduction showing that we have identified the maximal τ-confluent set.
To see what gain can be obtained on examples less adapted with respect to these observations, we have applied the τ-confluence technique to systems with purely sequential leaf components. We have chosen examples from the Cadp distribution: two versions of the Alternating Bit Protocol and five versions of a Distributed Leader Election Protocol. Table 2 shows the results. Note that in this case, comparing execution times is irrelevant, since τ-prioritisation of sequential components is known to be useless. It is very encouraging to note that in all experiments, the overhead in memory consumption is negligible, since memory more than time is usually the bottleneck in verification.

8 Conclusions

τ-confluence can be an effective technique to reduce transition systems at a reasonable cost. We propose to use composition expressions to help identify independent transitions resulting in τ-confluence at a negligible cost. One question arising is whether we can do better by enriching the set of composition operators.

In this paper we concentrate on results for strong confluence, mainly because we have no efficient way of recognising weak confluence at the leaf nodes. However, it would be useful to extend these results, especially since certain leaf nodes may be small enough to calculate larger sets of more weakly confluent transitions. Overall, we believe that composition structure information can, in various contexts, be used to improve existing algorithms.

References

1. H.R. Andersen. Model checking and boolean graphs. *Theoretical Computer Science*, 126(1):3–30, 1994.
2. S. Blom and J. van de Pol. State space reduction by proving confluence. In *Computer Aided Verification 2002*, number 2404 in LNCS, 2002.
3. S.C.C. Blom. Partial τ-confluence for efficient state space generation. Technical Report SEN–R0123, CWI, Amsterdam, 2001.
4. A.W. Roscoe et al. Hierarchical compression for model-checking CSP *or* how to check 10^{20} dining philosophers for deadlock. In *TACAS'95*, number NS-95-2, 1995.
5. Jean-Claude Fernandez, Hubert Garavel, Laurent Mounier, Anne Rasse, Carlos Rodriguez, and Joseph Sifakis. A toolbox for the verification of LOTOS programs. In *International Conference on Software Engineering*, pages 246–259, 1992.
6. Hubert Garavel and Frédéric Lang. SVL: a scripting language for compositional verification. In *International Conference on Formal Techniques for Networked and Distributed Systems*, pages 377–392. Kluwer Academic Publishers, 2001.
7. Susanne Graf and Bernhard Steffen. Compositional minimization of finite state systems. In *Computer Aided Verification*, volume 531 of *LNCS*, 1990.
8. J. F. Groote and M. P. A. Sellink. Confluence for process verification. *Theoretical Computer Science*, 170(1–2):47–81, 15 December 1996.
9. J. F. Groote and J. van de Pol. State space reduction using partial tau-confluence. In *Mathematical Foundations of Computer Science*, number 1893 in LNCS, 2000.
10. J.C. Fernandez, H. Garavel, A. Kerbrat, L. Mounier, R. Mateescu, M. Sighireanu. CADP: a protocol validation and verification toolbox. *CAV'96*, LNCS 1102, 1996.
11. Jean-Pierre Krimm and Laurent Mounier. Compositional state space generation from LOTOS programs. In *Proceedings of TACAS'97*, volume 1217 of *LNCS*, 1997.
12. Radu Mateescu and Mihaela Sighireanu. Efficient on-the-fly model-checking for regular alternation-free mu-calculus. *Science of Computer Programming*. to appear.
13. R. Nalumasu and G. Gopalakrishnan. An efficient partial order reduction algorithm with an alternative proviso implementation. *Formal Methods in System Design*, 20(3), May 2002.
14. D.A. Peled, V.R. Pratt, and G.J. Holzmann, editors. *Partial Order Methods in Verification*, volume 29 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*. American Mathematical Society, 1997.
15. Y.S. Ramakrishna and S.A. Smolka. Partial-order reduction in the weak modal mu-calculus. In *Proceedings of CONCUR'97*, volume 1243 of *LNCS*, 1997.
16. A. Valmari. Stubborn set methods for process algebras. In *Workshop on Partial Order Methods in Verification*, volume 29 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*. American Mathematical Society, 1997.
17. Mingsheng Ying. Weak confluence and τ-inertness. *Theoretical Computer Science*, 238(1–2):465–475, May 2000.