State Space Reduction using Partial τ-Confluence

Jan Friso Groote
CWI
P.O. Box 94079, 1090 GB Amsterdam, The Netherlands
Department of Mathematics and Computing Science, Eindhoven University of Technology
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
Email: JanFriso.Groote@cwi.nl

Jaco van de Pol
CWI
P.O. Box 94079, 1090 GB Amsterdam, The Netherlands
Email: Jaco.van.de.Pol@cwi.nl

ABSTRACT

We present an efficient algorithm to determine the maximal class of confluent τ-transitions in a labelled transition system. Confluent τ-transitions are inert with respect to branching bisimulation. This allows to use τ-priorisation, which means that in a state with a confluent outgoing τ-transition all other transitions can be removed, maintaining branching bisimulation. In combination with the removal of τ-loops, and the compression of τ-sequences this yields an efficient algorithm to reduce the size of large state spaces.

2000 Mathematics Subject Classification: 68Q20, 68Q50.
Keywords and Phrases: Confluence, State Space, Reduction, Branching Bisimulation, Labelled Transition System.

1. Introduction

A currently common approach towards the automated analysis of distributed systems is the following. Specify an instance of the system with a limited number of parties that are involved, and with a limited number of data that is exchanged. Subsequently, generate the state space of this system and reduce it using an appropriate equivalence, for which weak or branching bisimulation generally serves quite well. The reduced state space can readily be manipulated, and virtually all questions about it can be answered with ease, using appropriate, available tools (see e.g. [5, 4, 8] for tools to generate and manipulate state spaces). Of course correctness of the full distributed system does not follow in this way. But by taking the number of involved parties as well as the data domains as large as possible, a good impression of the behaviour can be obtained and many of its problems are exposed.

A problem of the sketched route is that the state spaces that are generated are as large as possible, which, giving the growing memory capacities of contemporary computers is huge. So, as the complexity of reduction algorithms is generally super linear, the time required to reduce these state spaces increases even more. Let n be the number of states and m be the number of transitions of a state space. The complexity of computing the minimal branching bisimilar state space is $\mathcal{O}(nm)$ [10] and computing the minimal weakly bisimilar one is approximately $\mathcal{O}(n^{\alpha})$ where $\alpha = 2.376$ is the typical constant required for matrix multiplication [11].

When the size of state spaces that can be handled grows, the time required to reduce these grows with an extra factor. Currently, state spaces that can be generated and handled on common computers contain in the order of 10^6 states. In the coming years we expect this number to grow to 10^8 and beyond. Assuming that the number of transitions are of the same order as the number of states and considering those cases where the complexity measure of the algorithm is tight, an increase in time of a factor 10^4 for branching bisimulation can be expected. This is not easily matched by an increase of the speed of processors.

We introduce a state space reduction algorithm of complexity $\mathcal{O}(m\, \mathit{Fanout}_{\tau}^{3})$ where Fanout_{τ} is the maximal number of outgoing τ-transitions of a node in the transition system. Assuming that for certain classes of transition systems Fanout_{τ} is constant, our procedure is linear in the size of the transition system.

The reduction procedure is based on the detection of τ-*confluence*. Roughly, we call a τ-transition from a state s confluent if for each a-transition starting in s, we can get again to the same state via an a and a confluent τ-transition, respectively. If the maximal class of confluent τ-transitions has been determined in this way, τ-priorisation is applied. This means that from each state with an outgoing confluent τ, all other outgoing transitions may be removed, giving "priority" to the τ-transition. In this way large parts of the transition system can become unreachable, which in some cases reduces the size of the state space with an exponential factor. For convergent systems, this reduction preserves branching bisimulation, so it can serve as a preprocessing step to computing the branching bisimulation minimization.

Related Work. Confluence has always been recognized as an important feature of the behaviour of distributed communicating systems. In both [15] and [16] a chapter is devoted to determinate and confluent processes, showing that certain operators preserve confluence, and showing how confluence can be used to verify certain processes. In [13] these notions have been extended to the π-calculus. In [9] an extensive investigation into various notions of global confluence for processes is given, where it is shown that by applying τ-priorisation, state spaces could be reduced substantially. In particular the use of confluence for verification purposes in the context of linear process operators was discussed. Linear process operators can be viewed as a symbolic representation of state spaces, into which huge or infinite state spaces can easily be encoded. The advantage of this approach is that confluence and τ-priorisation can be employed before the generation of state spaces, avoiding the generation of state spaces that are unnecesarily large. In [17] it is shown how using a typing system on processes it can be determined which actions are confluent, without generating the transition system. The goal of this is also to enable τ-priorisation before the generation of state spaces. In [12] such typing schemes are extended to the π-calculus.

Partial order reductions [7, 20] on the one hand, and τ-confluence and τ-priorisation on the other are strongly related. The idea behind partial order reductions is that a priori some property is established that must be checked on a transition system, together with an independence relation between certain actions. Based on the property it is established which actions can be viewed as internal, or τ. The independence relation on actions then determines a subset of τ-transitions for which the system is τ-confluent. Now the property can be checked by carrying out τ-priorisation, which in other words can be phrased as a reduction based on the partial ordering of actions.

Our primary contribution consists of providing an algorithm that determines the maximal set of confluent τ-transitions for a given transition system. This differs from the work in [9] which is only applicable if all τ-transitions are confluent, which is generally not the case. It also differs from approaches that use type systems or independence relations, in order to determine a subset of the confluent τ-transitions, given a high level description of the transition system. These methods are incapable of determining the maximal set of confluent τ-transitions in general.

In order to assess the effectiveness of our state space reduction strategy, we implemented it and compared it to the best implementation of the branching bisimulation reduction algorithm that we know [3]. In combination with τ-loop elimination, and τ-compression, we found that in the worst case the time that our algorithm required was in the same order as the time for the branching bisimulation algorithm. This happens when the number of classes of equivalent states in a transition system is small (typically up to 100 classes) and there are many hidden τ-transitions in the transition system. Under these circumstances the branching bisimulation algorithm performs well, whereas our algorithm is relatively weak.

But under more favourable conditions, with many equivalence classes, and many visible transitions, our algorithm reduced state spaces with factors varying from 10 to 1000 within seconds, being a substantial improvement over the time branching bisimulation requires.

In section 2 we introduce elementary notions, including the definition of confluence. In section 3 we define the removal of τ-loops as a necessary preprocessing step of our algorithm. In section 4 we explain our algorithm to determine the maximal class of τ-confluent transitions, and prove its correctness and complexity. In section 5 we explain τ-priorisation and τ-compression and in section 6 we combine all parts into a full algorithm and present some benchmark results.

Acknowledgements. We thank Holger Hermanns for making available for comparison purposes a new implementation of the branching bisimulation algorithm devised by him and others [3].

2. Preliminaries

In this section we define elementary notions such as labelled transition systems, confluence, branching bisimulation and T-convergence. Definition A *labelled transition system* is a three tuple $A = (S, Act, \longrightarrow$

) where

- S is a set of *states*;
- *Act* is a set of *actions*. We assume there is a special *internal action* τ that is always included in *Act*;
- $\longrightarrow \subseteq S \times Act \times S$ is a transition relation.

We write $\xrightarrow{a}$ for the binary relation $\{\langle s,t\rangle \mid \langle s,a,t\rangle \in \longrightarrow\}$. We write $s \xrightarrow{\tau^*} t$ iff there is a sequence $s_0, \ldots, s_n \in S$ with $n \geq 0$, $s_0 = s$, $s_n = t$ and $s_i \xrightarrow{\tau} s_{i+1}$. We write $t \overset{\tau^*}{\rightleftarrows} s$ iff $t \xrightarrow{\tau^*} s$ and $s \xrightarrow{\tau^*} t$, i.e. s and t lie on a τ-loop. Finally, we write $s \xrightarrow{\overline{a}} s'$ if either $s \xrightarrow{a} s'$, or $s = s'$ and $a = \tau$.

A set $T \subseteq \xrightarrow{\tau}$ is called a *silent transition set of* A. We write $s \xrightarrow{\tau}_T t$ iff $\langle s,t\rangle \in T$. With $s \xrightarrow{\overline{\tau}}_T t$ we denote $s = t$ or $s \xrightarrow{\tau}_T t$. We define the set $Fanout_\tau(s)$ for a state s by: $Fanout_\tau(s) = \{s \xrightarrow{\tau} s' | s' \in S\}$.

We say that A is *finite* if S and *Act* have a finite number of elements. In this case, n denotes the number of states of A, m is the number of transitions in A and m_τ denotes the number of τ-transitions. Furthermore, we write $Fanout_\tau$ for the maximal size of the set $Fanout_\tau(s)$.

Definition Let $A = (S, Act, \longrightarrow)$ be a labelled transition system and T be a silent transition set of A. We call A *T-confluent* iff for each transition $s \xrightarrow{\tau}_T s'$ and for all $s \xrightarrow{a} s''$ ($a \in Act$) there exists a state $s''' \in S$ such that $s' \xrightarrow{\overline{a}} s'''$, $s'' \xrightarrow{\overline{\tau}}_T s'''$. We call A *confluent* iff A is $\xrightarrow{\tau}$-confluent.

Definition Let $A = (S_A, Act, \longrightarrow_A)$ and $B = (S_B, Act, \longrightarrow_B)$ be labelled transition systems. A relation $R \subseteq S_A \times S_B$ is called a *branching bisimulation relation* on A and B iff for every $s \in S_A$ and $t \in S_B$ such that sRt it holds that

1. If $s \xrightarrow{a}_A s'$ then there exist t' and t'', such that $t \xrightarrow{\tau^*}_B t' \xrightarrow{\overline{a}}_B t''$ and sRt' and $s'Rt''$.
2. If $t \xrightarrow{a}_B t'$, then there exist s' and s'', such that $s \xrightarrow{\tau^*}_A s' \xrightarrow{\overline{a}}_A s''$ and $s'Rt$ and $s''Rt'$.

For states $s \in S_A$ and $t \in S_B$ we write $s \underline{\leftrightarrow}_b t$, and say s and t are *branching bisimilar*, iff there is a *branching bisimulation relation* R on A and B such that sRt. In this case, $\underline{\leftrightarrow}_b$ itself is the maximal branching bisimulation, and it is an equivalence relation. In the special case that A and B are the same transition systems, we say that R is a *branching auto-bisimulation* on A. For $s, s' \in S_A$ we write $s \underline{\leftrightarrow}_b^A s'$ iff there is a branching auto-bisimulation R on A such that sRs'.

Definition Let $A = (S, Act, \longrightarrow)$ be a labelled transition system. A transition $s \xrightarrow{\tau} s'$ is called *inert* iff $s \underline{\leftrightarrow}_b^A s'$.

Theorem Let $A = (S, Act, \longrightarrow)$ be a labelled transition system and let T be a silent transition set of A. If A is T-confluent, every $s \xrightarrow{\tau}_T s'$ is inert. Proof We show that the relation $R = \xrightarrow{\overline{\tau}}_T$ is a branching bisimulation relation. Note that the theorem is a direct consequence of this result. So, let sRs'. The case where $s = s'$ is trivial. Consider the case where sRs' because $s \xrightarrow{\tau}_T s'$. We prove the two cases of branching bisimulation:

Assume $s \xrightarrow{a} t$. So, as A is T-confluent, there exists some $t' \in S$ such that $t \xrightarrow{\overline{\tau}}_T t'$, $s' \xrightarrow{\overline{a}} t'$. Clearly, $s' \xrightarrow{\tau^*} s' \xrightarrow{\overline{a}} t'$ and sRs' and tRt'.

Now assume $s' \xrightarrow{a} t'$. Then $s \xrightarrow{\tau^*} s' \xrightarrow{a} t'$ and sRs' and $t'Rt'$.

Lemma Let A be a labelled transition system. There exists a largest silent transition set T_{conf} of A, such that A is T_{conf}-confluent. Proof Consider the set $\mathcal{T}$, being the set of all silent transition sets T such that A is T-confluent. Define $T_{conf} = \bigcup \mathcal{T}$. We first show that A is T_{conf}-confluent. So, let $s \xrightarrow{\tau}_{T_{conf}} s'$ and $s \xrightarrow{a} s''$, then by definition of T_{conf}, for some $T \in \mathcal{T}$, $s \xrightarrow{\tau}_T s'$. By confluence of T, we find s''', such that $s' \xrightarrow{\overline{a}} s'''$ and $s'' \xrightarrow{\overline{\tau}}_T s'''$. By definition of T_{conf}, $s'' \xrightarrow{\overline{\tau}}_{T_{conf}} s'''$.

Now consider any T such that A is T-confluent. By definition of T_{conf}, $T \subseteq T_{conf}$. So T_{conf} is indeed the largest.

Definition Let $A = (S, Act, \longrightarrow)$ be a labelled transition system. We call A (τ-)*convergent* iff there does not exist an infinite sequence of τ steps, i.e. there does not exist $s_1, s_2 \ldots \in S$ such that $s_1 \xrightarrow{\tau} s_2 \xrightarrow{\tau} \cdots$. We will employ well-founded recursion and induction over $\xrightarrow{\tau}$ for convergent systems.

3. Elimination of τ-cycles

In this section we define the removal of τ-loops from a transition system. The idea is to collapse each loop to a single state. This can be done, because $\overset{\tau^*}{\rightleftarrows}$ is an equivalence relation on states. Definition

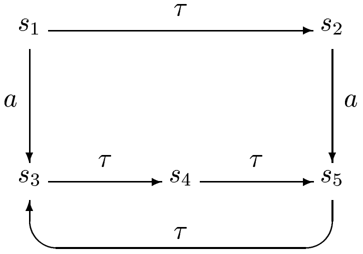
Let $A = (S, Act, \longrightarrow)$ be a labelled transition system. We write for any state $s \in S_A$:

$$[s]_A = \{t \in S \mid t \overset{\tau^*}{\rightleftarrows} s\}$$

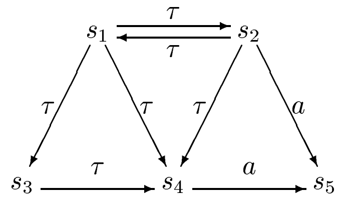
Moreover, we define the relation $[\longrightarrow]_A$ by:

$$S[\xrightarrow{a}]_A S' \text{ iff } \exists s \in S,\ s' \in S' \text{ such that } s \xrightarrow{a} s',\ \text{and } s \neq s' \text{ or } a \neq \tau.$$

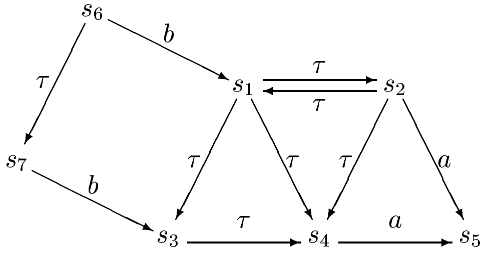
We write $[S]_A$ for $\{[s]_A | s \in S\}$ and $[T]_A$ for $\{[s]_A[\xrightarrow{a}]_A[t]_A | s \xrightarrow{a} t \in T\}$. Definition Let $A = (S, Act, \longrightarrow)$ be a labelled transition system. The *τ-cycle reduction* of A is the labelled transition system $A_\otimes = ([S]_A, Act, [\longrightarrow]_A)$. Using the algorithm to detect strongly connected components [1] it is possible to construct the τ-cycle reduction of a labelled transition system A in linear time. Lemma Let $A = (S, Act, \longrightarrow)$ be a labelled transition system and let $A_\otimes$ be its τ-cycle reduction. Then for every state $s \in S$, s in A is branching bisimilar to $[s]_A$ in $A_\otimes$. We show below that taking the τ-cycle reduction can change the confluence structure of a process. The first example shows that it can happen that some τ-transitions not being confluent, can become confluent after τ-cycle reduction.



The transition $s_1 \xrightarrow{\tau} s_2$ is not confluent before τ-cycle reduction, but it is afterwards. Of course, as the number of τ-transitions in a τ-cycle reduction of A can be much smaller than the number of τ-transitions in A, the absolute number of confluent τ's can be strongly reduced. But the following examples show that the situation is actually worse. Some τ-transitions that are confluent may have lost that property after τ-cycle reduction. Consider for instance



Observe that all τ-transitions are confluent. If τ-cycle reduction is applied, then states s_1 and s_2 are taken together, and we see that transitions $\{s_1, s_2\} \xrightarrow{\tau}_\otimes \{s_3\}$ and $\{s_1, s_2\} \xrightarrow{a}_\otimes \{s_5\}$ exist, but there is no way to complete the diagram. So, in the τ-cycle reduction, $\{s_1, s_2\} \xrightarrow{\tau}_\otimes \{s_3\}$ is not confluent. We can extend the example slightly, showing that states that have outgoing confluent transitions before τ-cycle reduction, do not have these afterwards:



So, before τ-cycle reductions all τ-transitions are confluent. In particular state s_6 has an outgoing τ-transition. After τ-cycle reduction, $\{s_1, s_2\} \xrightarrow{\tau}_\otimes \{s_3\}$ is not confluent anymore, and consequently, $\{s_6\} \xrightarrow{\tau}_\otimes \{s_7\}$ is not, too. So, state s_6 has lost the property of having an outgoing confluent τ-transition. Nevertheless, τ-cycle reduction is unavoidable in view of Example 5.

4. Algorithm to calculate T_{conf}

We now present an algorithm to calculate T_{conf} for a given labelled transition system $A = (S, Act, \longrightarrow)$. First the required data structures and their initialization are described:

1. Each transition $s \xrightarrow{\tau} s'$ is equipped with a boolean `candidate` that is initially set to true, and which indicates whether this transition is still a candidate to be put in T_{conf}.

2. For every state s we store a list of all ingoing transitions, as well as a list with outgoing τ-transitions.

3. Each transition $s \xrightarrow{a} s'$ is stored in a hashtable, such that given s, a and s', it can be found in constant time, if it exists.

4. Finally, there is a stack on which transitions are stored, in such a way that membership can be tested in constant time. A transition $s \xrightarrow{a} s'$ is on the stack if confluence of $s \xrightarrow{a} s'$ must be checked with regard to τ-transitions $s \xrightarrow{\tau} s''$. Initially, all transitions are put on the stack.

The algorithm works as follows:

> As long as the stack is not empty, remove a transition $s \xrightarrow{a} s'$ from it. Check that $s \xrightarrow{a} s'$ is still confluent with respect to all τ-transitions outgoing from state s that have the variable `candidate` set. Checking confluence means that for each candidate transition $s \xrightarrow{\tau} s''$ it must be verified that either
>
> - there exist an $s''' \in S$ such that $s' \xrightarrow{\tau} s'''$ with the variable `candidate` set to true, and there is a transition $s'' \xrightarrow{a} s'''$. Note that as there is a list of outgoing τ-transitions from s' and all transitions $s'' \xrightarrow{a} s'''$ are stored in the hashtable, carrying out this check takes time proportional to $Fanout_\tau$. Or,
> - $s'' \xrightarrow{a} s'$. This can be checked in constant time using the hash table. Or
> - $a = \tau$ and $s' \xrightarrow{\tau} s''$ with the variable `candidate` set, or finally,
> - $a = \tau$ and $s' = s''$.
>
> For all transitions $s \xrightarrow{\tau} s''$ for which the confluence check with respect to $s \xrightarrow{a} s'$ fails, the boolean `candidate` is set to false. If there is at least one transition $s \xrightarrow{\tau} s''$ for which the check fails, then all transitions $t \xrightarrow{a} s$ that are not on the stack must be put on it. This can be done conveniently, using the list of incoming transitions of node s.

After the algorithm has terminated, i.e. when the stack is empty, the set T_{alg} is formed by all τ-transitions for which the variable `candidate` is still true. Termination of the algorithm follows directly from the following observations:

- either, the size of the stack decreases, while the number of candidate transitions remains constant;

- or, the number of candidate transitions decreases, although in this case the stack may grow.

Correctness of the algorithm follows from the theorem below, showing that $T_{alg} = T_{conf}$.

Lemma A is T_{alg}-confluent. Proof Consider transitions $s \xrightarrow{a} s'$ and $s \xrightarrow{\tau}_T s''$. Consider the last step, with index say n, in the algorithm where $s \xrightarrow{a} s'$ is removed from the stack. As the variable `candidate` of $s \xrightarrow{\tau} s''$ was never set to false, it was established either that

- $a = \tau$ and $s' = s''$, or

- $s'' \xrightarrow{a} s'$, or

- $a = \tau$ and $s' \xrightarrow{\tau} s''$ with the variable `candidate` set (at step n), or

- there exists an $s''' \in S$ such that $s' \xrightarrow{\tau} s'''$ that was a candidate at step n, and $s'' \xrightarrow{a} s'''$.

In the first two cases it is obvious that $s \xrightarrow{a} s'$ and $s \xrightarrow{\tau} s''$ are T_{alg}-confluent w.r.t. each other. In the last two cases confluence is straightforward, if respectively $s' \xrightarrow{\tau} s''$ or $s' \xrightarrow{\tau} s'''$ are still candidate transitions when the algorithm terminates. This means that these transitions are put in T_{alg}. If, however, this is not the case, then there is a step $n' > n$ in the algorithm where the `candidate` variable of $s' \xrightarrow{\tau} s''$ or $s' \xrightarrow{\tau} s'''$, respectively, has been reset. In this case each transition ending in s' is put back on the stack. In particular $s \xrightarrow{a} s'$ is put on the stack to be removed at some step $n'' > n' > n$. But this means that n was not the last step of the algorithm where $s \xrightarrow{a} s'$ was removed from the stack, contradicting the assumption.

THEOREM $T_{conf} = T_{alg}$ PROOF From the previous lemma, it follows that $T_{alg} \subseteq T_{conf}$. We now prove the reverse. Assume towards a contradiction that $s \xrightarrow{\tau} s'$ in T_{conf} is the first transition in the algorithm, whose variable `candidate` is erroneously marked false.

This only happens when confluence wrt. some $s \xrightarrow{a} s''$ fails. By T_{conf}-confluence, for some s''', $s'' \xrightarrow{\overline{\tau}}_{T_{conf}} s'''$ and $s' \xrightarrow{\overline{a}} s'''$. As $s \xrightarrow{\tau} s''$ is marked false, it must be the case that $s' \xrightarrow{\tau} s'''$, and its `candidate` bit has been reset earlier in the algorithm.

But this contradicts the fact that we are considering the first instance where a boolean `candidate` was erroneously set to false.

LEMMA The algorithm terminates in $\mathcal{O}(m\, \mathit{Fanout}_\tau^3)$ steps. PROOF Checking that a single transition $s \xrightarrow{a} s'$ is confluent, requires $\mathcal{O}(\mathit{Fanout}_\tau^2)$ steps (for each $\xrightarrow{\tau}$-successor s'' of s we have to try the $\xrightarrow{\tau}$-successors of s'').

Every transition $s \xrightarrow{a} s'$ is put at most $\mathit{Fanout}_\tau + 1$ times on the stack: initially and each time a the variable `candidate` of a τ-successor s'' of s' is reset. As there are m transitions, this leads to a total amount of time of: $\mathcal{O}(m\, \mathit{Fanout}_\tau^3)$. Note that it requires $\mathcal{O}(m\, \mathit{Fanout}_\tau^2)$ to check whether a labelled transition system is τ-confluent with respect to all its τ-transitions. As determining the set T_{conf} is more expensive than determining global τ-confluence, and we only require a factor Fanout_τ to do so, we expect that the complexity of our algorithm cannot be improved.

We have looked into establishing other forms of partial τ-confluence (cf. [9]), especially forms where if there are transitions $s \xrightarrow{a} s'$ and $s \xrightarrow{\tau} s''$, there exists some state s''' such that $s' \xrightarrow{\tau^*} s'''$ and $s'' \xrightarrow{\tau^* a \tau^*} s'''$. However, doing this requires the dynamic maintenance of the transitive τ-closure relation, which we could not perform in a sufficiently efficient manner to turn it into an effective preprocessing step for branching bisimulation.

5. τ-PRIORISATION AND τ-COMPRESSION

After the set T_{conf} for a labelled transition system A has been determined we can "harvest" by applying τ-priorisation and calculating a form of τ-compression. Both operations can be applied in linear time, and moreover, reduce the state space.

The τ-priorisation operation allows to give precedence to silent steps, provided they are confluent. This is defined as follows: DEFINITION Let $A = (S, \mathit{Act}, \longrightarrow_A)$ be a labelled transition system and let T be a set of τ-transitions of A. We say that a transition system $B = (S, \mathit{Act}, \longrightarrow_B)$ is a *τ-priorisation* of A with respect to T iff for all $s, s' \in S$ and $a \in \mathit{Act}$

- if $s \xrightarrow{a}_B s'$ then $s \xrightarrow{a}_A s'$, and
- if $s \xrightarrow{a}_A s'$ then either $s \xrightarrow{a}_B s'$, or for some $s'' \in S$ it holds that $s \xrightarrow{\tau}_B s'' \in T$.

The following theorem implies that τ-priorisation maintains branching bisimulation. LEMMA Let $A = (S, \mathit{Act}, \longrightarrow_A)$ be a convergent, labelled transition system, which is T-confluent for some silent transition set T. Let $B = (S, \mathit{Act}, \longrightarrow_B)$ be a τ-priorisation of A wrt. T. Let $s \underline{\leftrightarrow}_b^A t$. If $s \xrightarrow{\tau^*}_A s' \xrightarrow{a}_A s''$ and $s \underline{\leftrightarrow}_b^A s'$, then for some t' and t'', $t \xrightarrow{\tau^*}_B t' \xrightarrow{a}_B t''$, $t \underline{\leftrightarrow}_b^A t'$ and $s'' \underline{\leftrightarrow}_b^A t''$. PROOF Well-founded induction on $\xrightarrow{\tau}$. Assume $s \underline{\leftrightarrow}_b^A t$ and $s \xrightarrow{\tau^*}_A s' \xrightarrow{a}_A s''$ and $s \underline{\leftrightarrow}_b^A s'$. By transitivity, $s' \underline{\leftrightarrow}_b^A t$. Hence for some $u' \underline{\leftrightarrow}_b^A t$ and $u'' \underline{\leftrightarrow}_b^A s''$, we have $t \xrightarrow{\tau^*}_A u' \xrightarrow{\overline{a}}_A u''$. We get two cases according to $t \xrightarrow{\tau^*}_A u'$:

- $t = u'$. As B is a τ-priorisation of A, we have two new cases:
  - either $u' \xrightarrow{\overline{a}}_B u''$. Then the theorem holds with $t' = t$ and $t'' = u''$.
  - or $u' \xrightarrow{\tau}_{B,T} u'''$. By Theorem 2, $u' \underline{\leftrightarrow}_b^A u'''$, so $s \underline{\leftrightarrow}_b^A u'''$. As $t \xrightarrow{\tau}_A u'''$, the induction hypothesis applies, so for some $t' \underline{\leftrightarrow}_b^A u'''$ and $t'' \underline{\leftrightarrow}_b^A s''$, $t = u' \xrightarrow{\tau}_B u''' \xrightarrow{\tau^*}_B t' \xrightarrow{\overline{a}}_B t''$.
- $t \xrightarrow{\tau}_A t''' \xrightarrow{\tau^*}_A u'$. As B is a τ-priorisation of A, we have again two cases:
  - Either $t \xrightarrow{\tau}_B t'''$. By the stuttering lemma [6], we know $t \underline{\leftrightarrow}_b^A t'''$.
  - Or $t \xrightarrow{\tau}_{B,T} u'''$ for some u'''. By Theorem 2, $t \underline{\leftrightarrow}_b^A u'''$.

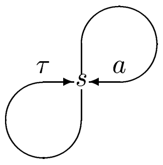
In both cases we have some v with $s\underline{\leftrightarrow}_b^A v$ and $t \xrightarrow{\tau}_B v$, so the induction hypothesis applies, which yields $t'\underline{\leftrightarrow}_b^A v$ and $t''\underline{\leftrightarrow}_b^A s''$ such that $t \xrightarrow{\tau}_B v \xrightarrow{\tau^*}_B t' \xrightarrow{\overline{a}}_B t''$.

THEOREM Let $A = (S, Act, \longrightarrow_A)$ be a convergent, labelled transition system, which is T-confluent for some silent transition set T. Let $B = (S, Act, \longrightarrow_B)$ be a τ-priorisation of A wrt. T. Then the auto-bisimulation $\underline{\leftrightarrow}_b^A$ is a branching bisimulation relation on A and B. PROOF Let $s\underline{\leftrightarrow}_b^A t$. We have to prove the obligations of branching bisimulation between A and B:

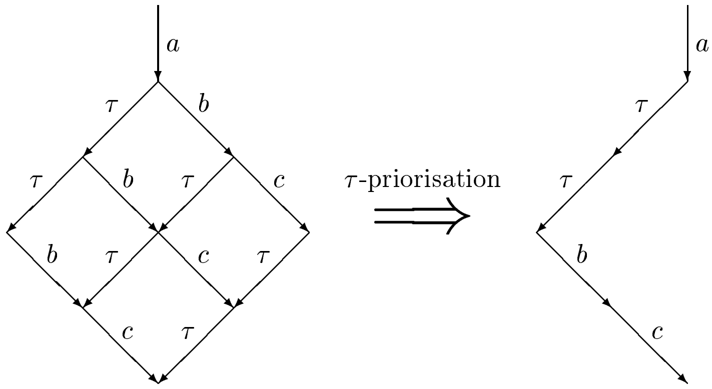
- If $s \xrightarrow{a}_A s'$, then for some $t'\underline{\leftrightarrow}_b^A s$ and $t''\underline{\leftrightarrow}_b^A s'$, we have $t \xrightarrow{\tau^*}_B t' \xrightarrow{\overline{a}}_B t''$; and
- If $t \xrightarrow{a}_B t'$, then for some $s'\underline{\leftrightarrow}_b^A t$ and $s''\underline{\leftrightarrow}_b^A t'$, we have $s \xrightarrow{\tau^*}_A s' \xrightarrow{\overline{a}}_A s''$.

The first one follows from the previous lemma, as $s \xrightarrow{\tau^*} s$. For the second one, let $t \xrightarrow{a}_B t'$, then as B is a τ-priorisation of A also $t \xrightarrow{a}_A t'$. Now (2) trivially follows from the fact that $s\underline{\leftrightarrow}_b^A t$.

EXAMPLE Convergence of a labelled transition system is a necessary precondition. Consider the following $\{s \xrightarrow{\tau} s\}$-confluent transition system, which is clearly not convergent.



The τ-priorisation with respect to $\{s \xrightarrow{\tau} s\}$ consists of a single τ-loop. Clearly, state s before and after τ-priorisation are not branching bisimilar. There is a straightforward linear algorithm that applies τ-priorisation wrt. a given set T of transitions to a labelled transition system. Just traverse all states. For each state check whether there is an outgoing τ-transition in T. If so, select one such τ-transition and remove all other transitions that start in this state. EXAMPLE Consider the labelled transition system below. All τ-transitions are confluent, and τ-priorisation removes more than half of the transitions.



Note also that a typical pattern in τ-prioritised transition systems are long sequences of τ-steps that can easily be removed. We call the operation doing so τ-compression.

DEFINITION Let $A = (S, Act, \longrightarrow)$ be a convergent labelled transition system. For each state $s \in S$ we define with well-founded recursion $\xrightarrow{\tau}$ the *τ-descendant* of s, notation $\tau^*(s)$, as follows:

$$\tau^*(s) = \begin{cases} \tau^*(s') & \text{if } s \xrightarrow{\tau}_A s' \text{ and for all } a \in Act,\ s'' \in S, \text{ if } s \xrightarrow{a} s'' \text{ then } a = \tau \text{ and } s' = s'', \\ s & \text{otherwise.} \end{cases}$$

The *τ-compression* of A is the labelled transition system $A_F = (S, Act, \longrightarrow_{A_F})$ where

$$\longrightarrow_{A_F} = \{\langle s, a, \tau^*(s')\rangle | s \xrightarrow{a}_A s'\}.$$

THEOREM Let $A = (S, Act, \longrightarrow)$ be a labelled transition system and A_F the τ-compression of A. Then $R = \{\langle s, s\rangle \mid s \in S\} \cup \{\langle s, \tau^*(s)\rangle \mid s \in S\}$ is a branching bisimulation relation on A and A_F. PROOF Let sRt. We have two cases:

- $t = s$. First, assume $s \xrightarrow{a}_A s'$, then by definition, $s \xrightarrow{a}_{A_F} \tau^*(s')$ and $s'R\tau^*(s')$. Conversely, if $s \xrightarrow{a}_{A_F} s'$, then by definition of A_F, for some s'' we have $s' = \tau^*(s'')$ and $s \xrightarrow{a}_A s''$. This proves both obligations of branching bisimulation in this case.

- $t = \tau^*(s)$. First, assume $s \xrightarrow{a}_A s'$. We again have two cases:
  - $a = \tau$ and this is the only τ-step from s. Then $\tau^*(s) = \tau^*(s')$ by definition of τ^*, hence we have $\tau^*(s) \xrightarrow{\overline{a}}_{A_F} \tau^*(s')$, viz. in zero steps.
  - In the other case $\tau^*(s) = s$, so the case $t = s$ applies.

  Conversely, assume $\tau^*(s) \xrightarrow{a}_{A_F} s'$. Then by definition of A_F, for some s'' we have $\tau^*(s'') = s'$ and $\tau^*(s) \xrightarrow{a}_A s''$. Of course, $s \xrightarrow{\tau^*}_A \tau^*(s)$. As $\tau^*(s) R \tau^*(s)$ and $s'' R s'$, we finished the proof.

Note that the τ-compression can be calculated in linear time by strictly following the definition. During a depth first sweep $\tau^*(s)$ can be calculated for each state s. Then by traversing all transitions, each transition $s \xrightarrow{a} s'$ can be replaced by $s \xrightarrow{a} \tau^*(s')$.

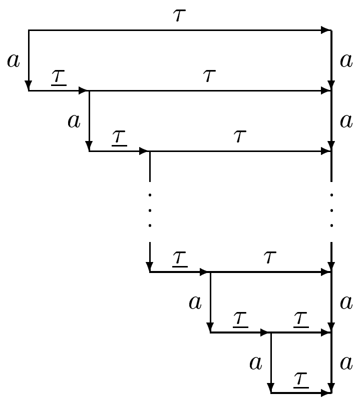
6. The full algorithm and benchmarks

In this section we summarize all parts that have hitherto been described in a complete algorithm and give some benchmarks to indicate how useful the algorithm is. When we had implemented the algorithm, our first observation was that iterative application of the algorithm led to repeated reduction of the state space. The reason is of course that τ-compression can make new diagrams confluent. Therefore, we present our algorithm as a fixed point calculation, starting with a transition system $A = (S, Act, \longrightarrow)$:

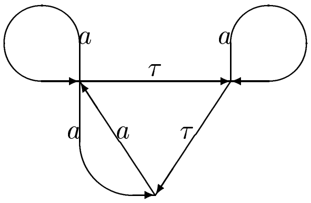
```
$B := A_{\otimes}$;
repeat
    $n$:=#states of $B$;
    calculate $T_{conf}$ for $B$;
    apply $\tau$-priorisation to $B$ wrt. $T_{conf}$;
    apply $\tau$-compression to $B$;
while $n \neq$ #states of B;
```

The following example shows that in the worst case the loop in the algorithm must be repeated $\Omega(n)$ times for a labelled transition system with n states.

Example In the labelled transition system below the underlined τ-transitions are the confluent ones. Therefore, only the left a-transition at the bottom will be removed by τ-priorisation. An application of τ-compression will connect the two τ-transitions at the one but last horizontal arrows. This will yield a new confluent diagram that will be removed by the subsequent iteration of the algorithm.



A further improvement, which we have not employed, might be to apply strong bisimulation reductions, in this algorithm. We have seen examples where this gives a substantial further reduction. As shown by [18] strong bisimulation can be calculated in $\mathcal{O}((m+n)\log n)$, which although not being linear, is quite efficient. An interesting question that arises immediately, is whether the combination of partial confluence checking, τ-priorisation, τ-reduction and strong bisimulation would yield a minimal transition system with respect to branching bisimulation. Unfortunately, this is not the case, as the following example shows, which is not minimal with respect to branching bisimulation, but which cannot be reduced using any of the reductions mentioned above, including the application of strong bisimulation.



In order to understand the effect of partial confluence checking we have applied our algorithm to a number of examples. We can conclude that if the number of internal steps in a transition system is relatively low, and the number of equivalence classes is high, our algorithm performs particularly well compared to the best implementation [3] of the standard algorithm for branching bisimulation [10]. Under less favourable circumstances, we see that the performance is comparable with the implementation in [3].

In table 1 we summarize our experience with 5 examples. In the rows we list the number of states "n" and transitions "m" of each example. The column under "n (red1)" indicates the size of the state space after 1 iteration of the algorithm. "#iter" indicates the number of iterations of the algorithm to stabilize, and the number of states of the resulting transition system is listed under "n (red tot)". The time to run the algorithm for partial confluence checking is listed under "time conf". The time needed to carry out branching bisimulation reduction and the size of the resulting state space are listed under "time branch" and "n min", respectively. The local confluence reduction algorithm was run on a SGI Powerchallenge with 250/300MHz R12000 processors. The branching bisimulation reduction algorithm was run on a 300MHz SUN Ultra 10.

The Firewire benchmark is the firewire or IEEE 1394 link protocol with 2 links and a bus as described in [14, 19]. The SWP1.2 and SWP1.3 examples are sliding window protocols with window size 1, and with 2 and 3 data elements, respectively. The description of this protocol can be found in [2]. The processes PAR2.12 and PAR6.7 are specially constructed processes to see the effect of the relative number of τ-transitions and equivalence classes on the relative speed of the branching bisimulation and local confluence checking algorithms. They are defined as follows:

$$\mathrm{PAR2.12} = \mathop{\|}_{i=1}^{12} \tau\, a_i \delta$$

$$\mathrm{PAR6.7} = \mathop{\|}_{i=1}^{7} \tau\, a_i\, b_i\, c_i\, d_i\, e_i\, \delta$$

where δ is a constant representing deadlock.

	n	m	n (red1)	#iter	n (red tot)	time conf	n min	time branch
Firewire	372k	642k	46k	4	23k	3.6s	2k	132s
SWP1.2	320k	1.9M	32k	6	960	13s	49	9s
SWP1.3	1.2M	7.5M	127k	6	3k	57s	169	136s
PAR2.12	531k	-	4k	2	4k	98s	4k	64s
PAR6.7	824k	4.9M	280k	2	280k	55s	280k	369s

Table 1: Benchmarks showing the effect of partial τ-confluence checking

References

1. A.V. Aho, J.E. Hopcroft and J.D. Ullman. Data structures and algorithms. Addison-Wesley. 1983.
2. M.A. Bezem and J.F. Groote. A correctness proof of a one bit sliding window protocol in μCRL. The Computer Journal, 37(4): 289-307, 1994.
3. M. Cherif and H. Garavel and H. Hermanns. The bcg_min user manual Version 1.1. `http://www.inrialpes.fr/vasy/cadp/man/bcg_min.html`, 1999.
4. D. Dill, C.N. Ip and U. Stern. Murphi description language and verifier. `http://sprout.stanford.edu/dill/murphi.html`, 1992-2000.
5. H. Garavel and R. Mateescu. The Caesar/Aldebaran Development Package. `http://www.inrialpes.fr/vasy/cadp/`, 1996-2000.
6. R.J. van Glabbeek and W.P. Weijland. Branching time and abstraction in bisimulation semantics (extended abstract). In *Information Processing 89*, (G.X. Ritter, ed.), North-Holland, Amsterdam, pp. 613–618.
7. P. Godefroid and P. Wolper. A partial approach to model checking. *Information and Computation*, 110(2):305-326, 1994.
8. J.F. Groote and B. Lisser. The μCRL toolset. `http://www.cwi.nl/~mcrl`, 1999-2000.
9. J.F. Groote and M.P.A. Sellink. Confluence for Process Verification. In Theoretical Computer Science B (Logic, semantics and theory of programming), 170(1-2):47-81, 1996.
10. J.F. Groote and F.W. Vaandrager. An efficient algorithm for branching bisimulation and stuttering equivalence. In M.S. Paterson, editor, Proceedings 17th ICALP, Warwick, volume 443 of Lecture Notes in Computer Science, pages 626-638. Springer-Verlag, 1990.
11. P.C. Kanellakis and S.A. Smolka. CCS expressions, finite state processes, and three problems of equivalence. Information and Computation, 86(1):43-68, 1990.
12. N. Kobayashi, B.C. Pierce, and D.N. Turner. Linearity and the π-calculus. In: *Proceedings of the 23rd POPL*, pages 358-371. ACM press, January 1996.
13. X. Liu and D. Walker. Confluence of Processes and Systems of Objects. In *Proceedings of TAPSOFT'95*, pages 217-231, LNCS 915, 1995.
14. S.P. Luttik. Description and formal specification of the link layer of P1394. Technical Report SEN-R9706, CWI, Amsterdam, 1997.
15. R. Milner. *A calculus of Communicating Systems.* LNCS 92, Springer-Verlag, 1980.
16. R. Milner. *Communication and Concurrency.* Prentice Hall International. 1989.
17. U. Nestmann and M. Steffen. Typing Confluence. In: Proceedings of FMICS'97, pages 77-101. CNR Pisa, 1997.
18. R. Paige and R. Tarjan. Three partition refinement algoriths. SIAM Journal on Computing 16(6):973-989, 1987.

19. M. Sighireanu and R. Mateescu. Proceedings of the 2nd COST 247 International Workshop on Applied Formal Methods in System Design (Zagreb, Croatia), June 1997.

20. A. Valmari. A stubborn attack on state explosion. In R.P. Kurshan and E.M. Clarke, editors, Proceedings of Computer Aided Verification, LNCS 531, pages 25-42, Springer-Verlag, 1990.