State Space Reduction of Rewrite Theories Using Invisible Transitions

Azadeh Farzan José Meseguer
Department of Computer Science,
University of Illinois at Urbana-Champaign.
{afarzan,meseguer}@cs.uiuc.edu

Abstract. State space explosion is the hardest challenge to the effective application of model checking methods. We present a new technique for achieving drastic state space reductions that can be applied to a very wide range of concurrent systems, namely any system specified as a rewrite theory. Given a rewrite theory $\mathcal{R} = (\Sigma, E, R)$ whose equational part (Σ, E) specifies some state predicates P, we identify a subset $S \subseteq R$ of rewrite rules that are P-invisible, so that rewriting with S does not change the truth value of the predicates P. We then use S to construct a reduced rewrite theory $\mathcal{R}/S$ in which all states reachable by S-transitions become identified. We show that if $\mathcal{R}/S$ satisfies reasonable executability assumptions, then it is in fact stuttering bisimilar to $\mathcal{R}$ and therefore both satisfy the same CTL^*_{-X} formulas. We can then use the typically much smaller $\mathcal{R}/S$ to verify such formulas. We show through several case studies that the reductions achievable this way can be huge in practice. Furthermore, we also present a generalization of our construction that instead uses a stuttering simulation and can be applied to an even broader class of systems.

1 Introduction

Although model checking is one of the most successful automated verification techniques, there are real limitations to its applicability in practice. These limitations are mostly related to the *state space explosion problem*. For example, as the number of processes considered in a distributed system grows, the associated state space may easily grow exponentially, particularly due to the system's concurrency. This can make it unfeasible to model check a system except for very small initial states, sometimes not even for those.

For this reason, a host of techniques to tame the state space explosion problem, which could be collectively described as *state space reduction techniques*, have been investigated: bisimulation techniques, partial order reduction (POR) techniques, abstraction techniques, and so on (see for example [20, 33, 22, 4, 9, 11, 19, 34, 32, 1, 21, 16]). The general idea is to transform the original system into a simpler one (typically bisimilar or at least similar to the original one) whose state space is small enough to model check properties. Transfer results then ensure that the same property holds in the original system.

This paper proposes a new such state space reduction technique within the rewriting logic semantic framework, in which concurrent systems are formally specified as rewrite theories [27]. In such specifications, the set of states is specified as an algebraic data type by an equational theory (Σ, E), and the system's

transitions are specified by rewrite rules R that are applied *modulo* the equations E. The rewrite theory specifying the system is then the triple $\mathcal{R} = (\Sigma, E, R)$. The fact that rewriting logic has been shown to be a very general and expressive semantic framework to specify concurrent systems [28, 25] makes our proposed state space reduction technique applicable to a very wide range of concurrent systems. Achieving a state space reduction typically requires discharging *proof obligations* to verify that the reduction is correct. In this regard, the fact that the state space is itself axiomatized by an equational theory (Σ, E) makes the tool-assisted discharging of such proof obligations using equational theorem proving techniques and tools much easier than if a non-logical specification formalism had been used instead.

Our technique is based on the idea of *invisible transitions*, that generalize a similar notion in POR techniques (see for example [5]). The basic setting is that we assume a rewrite theory $\mathcal{R} = (\Sigma, E, R)$ in which a certain set P of *state predicates* has been equationally axiomatized by some of the equations in E. $\mathcal{R}$ then has an associated Kripke structure, whose labeling function associates to each state (represented as an E-equivalence class of terms $[t]$ in the initial algebra $T_{\Sigma/E}$) all those predicates in P that hold in $[t]$ according to the equations E. We then call a rewrite rule r in R P-*invisible* if in any rewrite step $[t] \longrightarrow [t']$ using r the states $[t]$ and $[t']$ satisfy the *same* state predicates, i.e., they are labeled in the same way. Our state space reduction technique is then very simple: we identify a subset $S \subseteq R$ of rules such that all rules in S are invisible. We then define the S-*reduction* of $\mathcal{R} = (\Sigma, E, R)$ as the rewrite theory $\mathcal{R}/S = (\Sigma, E \cup S, R \backslash S)$, that is, we *turn all rules in* S into *equations*, thus collapsing the set of states from $T_{\Sigma/E}$ to the quotient $T_{\Sigma/E \cup S}$. The intuitive idea, therefore, is that all states that can be reached from a given state by repeated S-transitions can be *collapsed* into a single one. In practice, as we show by means of several case studies in Section 4, the reductions obtained this way can be huge.

However, the above technique must meet an important *executability requirement*. The point is that, for E an arbitrary set of equations, rewriting modulo E, which is the way transitions take place in the Kripke structure associated to $\mathcal{R} = (\Sigma, E, R)$, is in general undecidable. Therefore, to be able to execute and model check a rewrite theory in a rewriting logic language implementation such as Maude [6, 7] we must require that the equations E are confluent and terminating (perhaps modulo some axioms A) and that the rules R are strongly *coherent* with respect to the equations E [35]. Intuitively, the coherence requirement means that we can identify a state $[t]$ with the canonical form $can_E(t)$ of t by the equations E, and that rewriting with equations E and with rules R *commutes* in an appropriate sense, so that we can safely restrict our computations with R to only rewrite E-canonical forms. Therefore, the executability requirement for our technique is that $\mathcal{R}/S = (\Sigma, E \cup S, R \backslash S)$ should be executable, that is, that $E \cup S$ should be confluent and terminating, and that the rules $R \backslash S$ should be strongly coherent with respect to $E \cup S$ (perhaps modulo axioms A).

We show in Section 3 that the above-mentioned executability requirements on $\mathcal{R}/S$, besides being absolutely essential to model check $\mathcal{R}/S$ in practice, ensure a further very important property, namely that $\mathcal{R}$ and $\mathcal{R}/S$ are *stuttering*

bisimilar, and therefore they satisfy exactly the same CTL^*_{-X} formulas. Furthermore, to make our technique applicable to cases where a suitable set S may not be available, we generalize it to allow enlarging a set of invisible rules S by adding new invisible rules not in R to get a superset $\widehat{S} \supseteq S$. This gives rise to a state space reduction $\widehat{\mathcal{R}}/\widehat{S}$ that is no longer stuttering bisimilar to $\mathcal{R}$ but is nevertheless similar to it. This still allows us to verify $ACTL^*_{-X}$ formulas for $\mathcal{R}$ if we can model check them for $\widehat{\mathcal{R}}/\widehat{S}$, but such model checking can now give rise to spurious counterexamples. We illustrate how this more general technique is also quite useful in practice by means of a client-server protocol in Section 4.3.

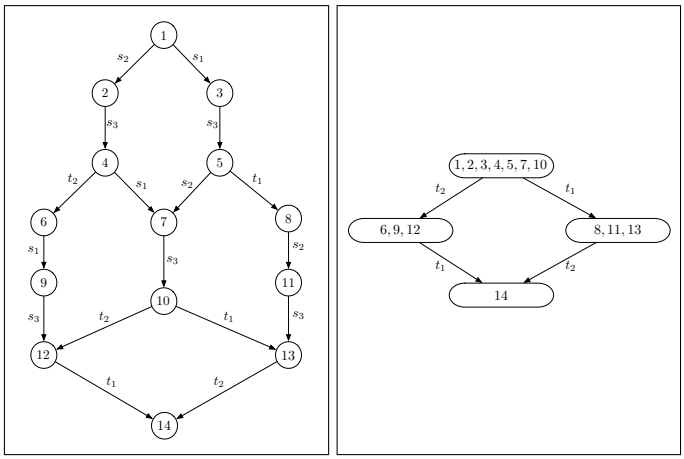


Fig. 1. Restaurant State Space

We can make all these ideas concrete by means of an example which models the workflow in a simplistic restaurant with one waiter and two customers. Customers have a flag indicating their status (waiting, ordered, or eating), so a customer is represented as a pair $C(id, f)$ with id an identifier and f the flag. The waiter has also a status flag (free or order-taken). Therefore, the waiter is represented by a term of the form $W(f)$. The restaurant state is a set with a waiter and two customers, with set union represented by a binary associative and commutative juxtaposition operator "__". We have the following rewrite rules R in our theory $\mathcal{R} = (\Sigma, A, R)$, where A consists of the associativity and commutativity axioms for "__":

$$
\begin{array}{ll}
s_1 : W(\text{free})C(1,\text{waiting}) & \longrightarrow W(\text{order-taken})C(1,\text{ordered}) \\
s_2 : W(\text{free})C(2,\text{waiting}) & \longrightarrow W(\text{order-taken})C(2,\text{ordered}) \\
s_3 : W(\text{order-taken}) & \longrightarrow W(\text{free}) \\
t_1 : W(\text{free})C(1,\text{ordered}) & \longrightarrow W(\text{free})C(1,\text{eating}) \\
t_2 : W(\text{free})C(2,\text{ordered}) & \longrightarrow W(\text{free})C(2,\text{eating})
\end{array}
$$

Figure 1 (a) shows the state space induced by the above rewrite rules from an initial state with the waiter free and the two customers waiting.

Let us assume that the property ϕ that we are interested in is: "eventually both customers eat". This property can be expressed as formula $\diamond(e_1 \wedge e_2)$ where e_i is true if the ith customer's status is "eating" and false otherwise. Rewrite rules s_1, s_2, and s_3 do not change the truth value of the predicates e_1 and e_2. One can observe that the rules in $S = \{s_1, s_2, s_3\}$ are confluent and terminating and $R \backslash S$ is strongly locally coherent [35] with respect to S modulo axioms. The reduced theory $\mathcal{R}/S$ (see the state space in Figure 1 (b), where each state represents an S-equivalence class) is then stuttering bisimilar to the theory $\mathcal{R}$.

Besides the small example used above to illustrate the main ideas, in Section 4 we show that our technique yields very drastic state space reductions in three more substantial case studies involving well-known algorithms and applications. Furthermore, in Section 5 we discuss in detail the discharging of the necessary proof obligations ensuring that a proposed S-reduction $\mathcal{R}/S$ is both correct and executable, and the kind of tool support necessary to facilitate such discharging activities. We end with a discussion of related work and some concluding remarks in Section 6.

2 Preliminaries

2.1 Termination, Confluence and Coherence in Rewrite Theories

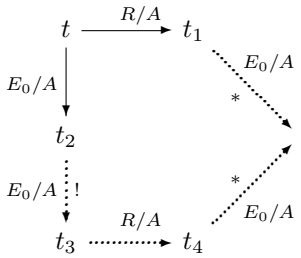
A rewrite theory [27] is a triple $\mathcal{R} = (\Sigma, E, R)$ where (Σ, E) is an equational theory with signature Σ and equations E, and where R is a set of conditional rewrite rules of the form $l \longrightarrow r$ if C. In this paper we assume that C is always an equational condition. Intuitively, if a concurrent system is modeled as a rewrite theory $\mathcal{R} = (\Sigma, E, R)$, then the equational theory (Σ, E) defines the system *states* (terms in $T_{\Sigma/E}$) and the set of rewrite rules R specify the system's concurrent transitions.

Given two terms $u, v \in T_\Sigma$, a one-step rewrite $u \xrightarrow{\tau} v$ means that there is a rule $\tau : l \to r$ if C in R that can be applied to a subterm of u with a ground substitution θ such that $E \models \theta C$ and u rewrites to v by replacing the subterm $\theta(l)$ by the subterm $\theta(r)$. We write $u \xrightarrow{R} v$ to mean that there is a rule $\tau \in R$ such that $u \xrightarrow{\tau} v$. The notation $\xrightarrow{R}_*$ denotes the reflexive and transitive closure of the relation $\xrightarrow{R}$. Set Can_S includes all elements $x \in T_\Sigma$ such that no rule in S is enabled at x. We define $\xrightarrow{S}_! = \{(x, y) | x \xrightarrow{S}_* y \ \wedge \ y \in \mathrm{Can}_S\}$ and $x \downarrow_S y \Leftrightarrow \exists z : x \xrightarrow{S}_* z \ \wedge y \xrightarrow{S}_* z$. Rewriting over equivalence classes modulo equations E is defined as follows: $[t]_E \xrightarrow{r} [t']_E$ if and only if there are terms u and v such that $u \in [t]_E$ and $v \in [t']_E$ and $u \xrightarrow{r} v$. We define $\xrightarrow{R/E}$ by the equivalence $[t]_E \xrightarrow{R} [t']_E \ \Leftrightarrow \ t \xrightarrow{R/E} t'$.

A set $S \subseteq R$ of rewrite rules is *confluent* modulo E in the theory (Σ, E, R) if and only if $\forall t, t', t'' \in T_\Sigma : \ ([t]_E \xrightarrow{S}_* [t']_E \ \wedge [t]_E \xrightarrow{S}_* [t'']_E) \ \Rightarrow \ (\exists w : [t']_E \xrightarrow{S}_* [w]_E \ \wedge [t'']_E \xrightarrow{S}_* [w]_E)$. S is *terminating* if for all t there exists no infinite chain of rewriting $[t]_E \xrightarrow{S} \ldots \xrightarrow{S} \ldots$.

Definition 1. *[35] In a rewrite theory $\mathcal{R} = (\Sigma, E, R)$, where $E = E_0 \cup A$ with E_0 a (terminating) set of equations and A a set of equational axioms, R is called* locally strongly coherent *with respect to E_0 modulo A if*

$$(t \xrightarrow{R/A} t_1 \ \wedge \ t \xrightarrow{E_0/A} t_2) \Rightarrow (\exists t_3, t_4 : t_2 \xrightarrow{E_0/A}_! t_3 \ \wedge \ t_3 \xrightarrow{R/A} t_4 \ \wedge \ t_4 \downarrow_{E_0/A} t_1)$$



Strong local coherence is the main property to check to ensure executability of a rewrite theory $\mathcal{R} = (\Sigma, E_0 \cup A, R)$ when we have matching algorithms for the equational axioms A. Viry shows that if the equations E_0 are confluent and terminating modulo A, then strong local coherence implies a more general strong coherence property [35]. Strong coherence ensures that we can achieve the effect of rewriting with R in $E_0 \cup A$-equivalence classes by first computing the $E_0 \cup A$-canonical form modulo A, and then rewriting that canonical form with R modulo A.

2.2 Stuttering Simulations

Let us assume that the equational part (Σ, E) of a rewrite theory $\mathcal{R} = (\Sigma, E, R)$ defines, among other things a set P of *state predicates* on the initial algebra $T_{\Sigma/E}$[1]. We can then associate to $\mathcal{R}$ a *Kripke structure* [5] whose states are the set $T_{\Sigma/E,State}$ for some designated sort *State* of states, whose labeling function assigns to each state the predicates $p \in P$ that provably hold in it using E, and whose transition relation is the total closure $\xrightarrow{R}_\bullet$ of $\xrightarrow{R}$, that is, we make $\xrightarrow{R}$ into a total relation by adding identity transitions for each deadlock state. We can then interpret any temporal logic formula, say in CTL^* in $\mathcal{R}$, namely by interpreting it in its associated Kripke structure. For a more detailed presentation on the relations between rewrite theories, Kripke structures and temporal logic, with applications to model checking in Maude see [14].

We present some basic notions and results, used later on, about transition systems, Kripke structures, and stuttering (bi-)simulations between them that will apply in particular to the Kripke structures associated to rewrite theories.

Definition 2. *Let $\mathcal{A} = (A, \xrightarrow{A})$ and $\mathcal{B} = (B, \xrightarrow{B})$ be transition systems and let $H \subseteq A \times B$ be a relation. Given a path π in $\mathcal{A}$ and a path ρ in $\mathcal{B}$, we say that ρ H-matches π if there are strictly increasing functions $\alpha, \beta : \mathbb{N} \to \mathbb{N}$ with $\alpha(0) = \beta(0) = 0$ such that, for all $i, j, k \in \mathbb{N}$, if $\alpha(i) \leq j < \alpha(i+1)$ and $\beta(i) \leq k < \beta(i+1)$, it holds that $\pi(j) H \rho(k)$.*

Definition 3. *Given transition systems $\mathcal{A}$ and $\mathcal{B}$, a stuttering simulation of transition systems $H : A \longrightarrow B$ is a binary relation $H \subseteq \mathcal{A} \times \mathcal{B}$ such that if*

[1] Note that all the equivalent states modulo E satisfy the same set of predicates.

aHb, then for each path π in $\mathcal{A}$ starting at a there is a path ρ in $\mathcal{B}$ starting at b that H- matches π.

Definition 4. *Given Kripke structures $\mathcal{A} = (A, \xrightarrow{A}, L_A)$ and $\mathcal{B} = (B, \xrightarrow{B}, L_B)$ over a set of predicates P, a stuttering P-simulation $H : A \to B$ is a stuttering simulation of transition systems $H : (A, \xrightarrow{A}) \to (B, \xrightarrow{B})$ such that if aHb then $L_B(b) \subseteq L_A(a)$. We call the stuttering P-simulation strict if aHb implies $L_B(b) = L_A(a)$. H is called a stuttering P-bisimulation if both H and H^{-1} are stuttering P-simulations.*

In [31], it is shown that (strict) stuttering simulations preserve the satisfaction of $ACTL^*_{-X}(P)$ formulas. Also, [24, 3] state that stuttering bisimulations preserve the satisfaction of $CTL^*_{-X}(P)$ formulas which can be derived by generalizing the results from [31].

3 Invisible Transitions and the $\mathcal{R}/S$ Reduction

Definition 5. *Given a rewrite theory $\mathcal{R} = (\Sigma, E, R)$ and having an equationally-defined set of atomic predicates P, a rewrite rule $\tau : l \to r$ if C in R is called **P-invisible** if for any $[t] \in T_{\Sigma/E}$ and any $u \in [t]$ such that $u \xrightarrow{\tau} v$, then for each $p \in P$ we have $[t] \models p \Leftrightarrow [v] \models p$. We denote by $Inv^P(R)$ the set of all P-invisible rewrite rules of R.*

We call $\mathcal{R}/S = (\Sigma, S \cup E_0 \cup A, T = R \backslash S)$ the S-reduced theory of $\mathcal{R} = (\Sigma, E_0 \cup A, R)$. We are particularly interested in the S-reduced theory of $\mathcal{R}$ when $S \subseteq \mathrm{Inv}^P(R)$, $S \cup E_0$ is confluent and terminating modulo A, and T is coherent with respect to $S \cup E_0$ modulo A.

Theorem 1. *Let $\mathcal{R} = (\Sigma, E_0 \cup A, R)$ be a rewrite theory with P a set of equationally defined atomic predicates. Let $S \subseteq R$ be a set of P-invisible rules such that $S \cup E_0$ is confluent and terminating modulo A, and $T = R \backslash S$ is coherent with respect to $S \cup E_0$ modulo A. Then $\mathcal{R}$ and $\mathcal{R}/S$ are stuttering bisimilar.*

Proof. (sketch) The relation H on which the bisimilarity is based is defined by the quotient homomorphism $H : T_{\Sigma/E} \twoheadrightarrow T_{\Sigma/E \cup S}$. We need to prove that: (a) H is a stuttering simulation; and (b) that H^{-1} is so too. Since H maps deadlock states to deadlock states, and H^{-1} of a deadlock state always contains a deadlock state, we can disregard deadlocks.

(a) It suffices to show that for each path π in the underlying Kripke structure of the theory $\mathcal{R}$, $(T_{\Sigma/E}, \xrightarrow{R}_\bullet)$, there exists a stuttering equivalent path π' in the underlying Kripke structure of the S-reduced theory $\mathcal{R}/S$, $(T_{\Sigma/E \cup S}, \xrightarrow{S/\widehat{S}}_\bullet)$.

π must be of the following general form:

$$\pi : [s_0]_E \xrightarrow{S}_* [t_0]_E \xrightarrow{T} [s_1]_E \xrightarrow{S}_* [t_1]_E \xrightarrow{T} \ldots \xrightarrow{T} [s_n]_E \xrightarrow{S}_* [t_n]_E \xrightarrow{T} \ldots$$

Since the rules in S are P-invisible, we know that $L(s_i) = L(t_i)$ for all i. Also, observe that by collapsing the $\xrightarrow{S}_*$, we have $[s_i]_{E \cup S} = [t_i]_{E \cup S}$. Then the following path

$$\pi' : [t_0]_{S \cup E} \xrightarrow{T} [t_1]_{S \cup E} \xrightarrow{T} \ldots \xrightarrow{T} [t_n]_{S \cup E} \xrightarrow{T} \ldots$$

is stuttering equivalent to π and of course, by construction, it is a path in the underlying Kripke structure of $\mathcal{R}/S$.

(b) It suffices to show that for each path ρ in the underlying Kripke structure of the theory $\mathcal{R}/S$, $(T_{\Sigma/E\cup S}, \rightarrow_{R/S})$, there exists a stuttering equivalent path ρ' in the underlying Kripke structure of the reduced theory $\mathcal{R}$, $(T_{\Sigma/E}, \xrightarrow{R}_{\bullet})$.

Assume that ρ is of the following general form:

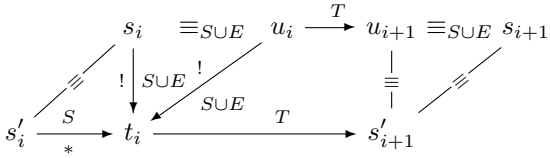
$$\rho : [s_0]_{S\cup E} \xrightarrow{T} [s_1]_{S\cup E} \xrightarrow{T} \ldots \xrightarrow{T} [s_n]_{S\cup E} \xrightarrow{T} \ldots$$

We show by construction that there exists a stuttering equivalent path ρ' of the following form:

$$\rho' : [s'_0]_E \xrightarrow{S}_* [t_0]_E \xrightarrow{T} [s'_1]_E \xrightarrow{S}_* [t_1]_E \xrightarrow{T} \ldots \xrightarrow{T} [s'_n]_E \xrightarrow{S}_* [t_n]_E \xrightarrow{T} \ldots$$

where $s_0 = s'_0$ and for all i, $s_i \equiv_{S\cup E} s'_i$, and therefore $L(s_i) = L(s'_i)$. H^{-1} then relates the state $[s_i]_{S\cup E}$ to all the states on $[s'_i]_E \xrightarrow{S}_* [t_i]_E$ which by invisibility of S all satisfy the same set of predicates.

$[s_i]_{S\cup E} \xrightarrow{T} [s_{i+1}]_{S\cup E}$ implies that there are terms u_i and u_{i+1} such that $s_i \equiv_{S\cup E} u_i \xrightarrow{T} u_{i+1} \equiv_{S\cup E} s_{i+1}$. If $s'_i \equiv_{S\cup E} s_i$ (meaning $s'_i H s_i$), then there is a term t_i such that $s_i \xrightarrow{S\cup E}_! t_i$ and $s'_i \xrightarrow{S\cup E}_! t_i$. Since $u_i \equiv_{S\cup E} s_i$, by confluence of $S \cup E$, we have $u_i \xrightarrow{S\cup E}_! t_i$. Therefore, by T being coherent with respect to $S \cup E_0$ modulo A, there exists a term s'_{i+1} such that $t_i \xrightarrow{T} s'_{i+1}$ and $s'_{i+1} \downarrow_{S\cup E} u_{i+1}$. Since $u_{i+1} \equiv_{S\cup E} s_{i+1}$, we have $s'_{i+1} \equiv_{S\cup E} s_{i+1}$ (meaning that $s'_{i+1} H s_{i+1}$).



Start by letting $s'_0 = s_0$. Since $s'_0 = s_0$, it trivially holds that $s'_0 \equiv_{S\cup E} s_0$. Inductively construct the path according to the above diagram. Note that by viewing S steps as τ-transitions, the above argument also shows that $\equiv_{S\cup E}$ is a branching bisimulation relation [30]. □

We have shown that, under the theorem hypothesis, the reduced rewrite theory $\mathcal{R}/S$ is stuttering P-bisimilar with the original theory $\mathcal{R}$. Therefore, (see Section 2) for any $\phi \in CTL^*_{-X}(P)$[2] , and any initial state $[t]_E$ we have

$$\mathcal{R}, [t]_E \models \phi \;\Leftrightarrow\; \mathcal{R}/S, [t]_{E\cup S} \models \phi$$

In practice, the reduced theory $\mathcal{R}/S$ can have a drastically smaller state space than $\mathcal{R}$, making model checking of $\mathcal{R}/S$ feasible when model checking of $\mathcal{R}$ is unfeasible.

[2] Note that the simulation relations are *strict* in the sense that $aHb \Rightarrow L(a) = L(b)$ and therefore negation does not have to be excluded.

In cases where the $\mathcal{R}/S$ construction cannot be carried out for lack of a suitable S satisfying the confluence condition in Theorem 1, we can nevertheless achieve a similar state space reduction with a relation H that is a stuttering *simulation*. For example the client-server reduction in Section 4.3 is achieved in this manner. The general method is as follows: we assume that we have a set of rules $S \subseteq R$ which are P-invisible ($S \subseteq \mathrm{Inv}^P(R)$), and $T = R \backslash S$ is coherent with respect to $S \cup E_0$ modulo A, and $S \cup E_0$ is terminating but not confluent modulo A. We then extend S to a set of rules $\widehat{S}$ with $S \subseteq \widehat{S}$, $\widehat{S} \not\subseteq R$, and where $\widehat{S}$ is still P-invisible, and $(R \backslash \widehat{S})$ is coherent with respect to $\widehat{S} \cup E_0$ modulo A, and furthermore, $E_0 \cup \widehat{S}$ is terminating and confluent modulo A. Consider now the rewrite theory $\widehat{\mathcal{R}} = (\Sigma, E_0 \cup A, R \cup \widehat{S})$. Since $\widehat{\mathcal{R}}$ has more rules than $\mathcal{R}$, if $\mathcal{R}$ is deadlock-free[3], that is, if any state $[t]_E$ can always be rewritten by R to a new state $[t']_E$, then the following proposition is easy to prove:

Proposition 1. *The identity homomorphism* $1_{T_{\Sigma/E_0 \cup A}} : T_{\Sigma/E_0 \cup A} \to T_{\Sigma/E_0 \cup A}$ *induces a* P*-simulation map from the underlying Kripke structure of* $\mathcal{R}$ *to that of* $\widehat{\mathcal{R}}$.

We can now apply Theorem 1 to $\widehat{\mathcal{R}}$ to obtain a stuttering P-bisimilar $\widehat{S}$-reduced theory $\widehat{\mathcal{R}}/\widehat{S}$. Since any simulation is a special case of a stuttering simulation, and stuttering simulations are closed under composition [31, 24], by composing the above simulation from $\mathcal{R}$ to $\widehat{\mathcal{R}}$ with the stuttering bisimulation from $\widehat{\mathcal{R}}$ to $\widehat{\mathcal{R}}/\widehat{S}$ generated by Theorem 1, we obtain a stuttering *simulation* from $\mathcal{R}$ to $\widehat{\mathcal{R}}/\widehat{S}$ and therefore we have

Theorem 2. *Under the above assumptions for any* $\phi \in ACTL^*_{-X}(P)$ *and any initial state* $[t]_{E_0 \cup A}$ *in* $\mathcal{R}$*, we have* $\widehat{\mathcal{R}}/\widehat{S}, [t]_{E_0 \cup \widehat{S} \cup A} \models \phi \;\Rightarrow\; \mathcal{R}, [t]_{E_0 \cup A} \models \phi$.

Therefore, if we can model check the property ϕ using the reduced theory $\widehat{\mathcal{R}}/\widehat{S}$, we are then guaranteed that ϕ holds in $\mathcal{R}$. See Section 4.3 for an example.

4 Case Studies

We present three case studies showing how the $\mathcal{R}/S$ and $\widehat{\mathcal{R}}/\widehat{S}$ constructions can be achieved in practice for real applications, leading to massive reductions in the state space. All the experiments have been performed with the Maude LTL model checker running on an Intel machine with a 2.6GHz processor and 4GB of memory running Linux.

4.1 Leader Election Protocol

We consider the simple case where the network is a ring consisting of n nodes, numbered from 1 to n in the clockwise direction. We want to investigate the LCR algorithm to select a leader. The informal description of this algorithm is as follows [23]:

[3] Given a rewrite theory $\mathcal{R}$, we can always transform it into a bisimilar deadlock-free theory (see [29]). Therefore, there is no real loss of generality imposed by this requirement.

> Each process sends its identifier around the ring. When a process receives an incoming identifier, it compares that identifier to its own. If the incoming identifier is greater than its own, it keeps passing the identifier; if it is less than its own, it discards the incoming identifier; if it is equal to its own the process declares itself the leader.

We can specify a rewrite theory modeling this algorithm by means of objects and messages, where the distributed state is a multiset of objects and messages built by an associative and commutative multiset union operator "_ _":

$$\begin{array}{lll} s_0 : \langle I \rangle & \longrightarrow [I]\ (I \rightarrow I + 1 \bmod N) & \\ s_1 : [I]\ (J \rightarrow I) & \longrightarrow [I] & \text{if } J < I \\ s_2 : [I]\ (J \rightarrow I) & \longrightarrow [I](J \rightarrow I + 1 \bmod N) & \text{if } J > I \\ t : \ [I]\ (I \rightarrow I) & \longrightarrow \text{Leader}(I) & \end{array}$$

where N is the number of processes on the ring and $\langle I \rangle$ is the initial state of process I. In the first phase (rewrite rule s_0), each process I sends its identifier to its neighbor and changes its format $[I]$ so that this is done only once. As soon as a process I receives its own identifier through the ring, the computation is over; it removes all the object and the message and outputs Leader(I). The messages are of the general form $(I \rightarrow J)$ where J is the identifier of the receiver and I is the integer content of the message.

The set $S = \{s_0, s_1, s_2\}$ can be shown to be confluent and terminating modulo associativity and commutativity. Let us assume that the property that we are interested in is that eventually some process will be elected as leader. This is expressed by means of a single atomic predicate, p, that is true in any state containing Leader(I). The rules in S are p-invisible, and t is coherent with respect to S modulo the associativity and commutativity axioms. Therefore, by Theorem 1, we can use the stuttering bisimilar reduction $\mathcal{R}/S$ to model check our property. Note that reducing $\mathcal{R}$ with the rewrite rule s_0 above (which can easily be shown to be confluent and terminating) collapses an N-dimensional cube (generated by rule s_0) into a path of length N, meaning that the number of states in $\mathcal{R}/\{s_0\}$ is reduced from 2^N to N, and the number of paths reduces from 2^N to 1. Table 1 shows the performance evaluation of model checking this problem before and after reduction using the Maude LTL model checker.

4.2 Distributed Spanning Tree

A *spanning tree* of an undirected graph $G = (V, E)$ is a tree (i.e., a connected acyclic graph) that consists entirely of undirected edges and contains every vertex of G. The distributed spanning tree problem tries to find a spanning tree for a given set of network nodes V that are connected by E. The asynchronous algorithm from [23] solves this as follows:

> There is a distinct node r that is initially marked and acts as the root. A marked node v asynchronously sends a message to each of its neighbors once and for all. An unmarked node v nondeterministically chooses one of the nodes who have sent it a message as its parent in the spanning tree, becomes marked, and discards all the other messages.

One possible way of specifying the above algorithm is by the following rewrite rules:

$$
\begin{array}{ll}
s_1 : [\ N \mid P,\ M\ NL] & \longrightarrow [\ N \mid P,\ NL](M \leftarrow N) \\
t_1 : [N \mid \texttt{none},\ NL](N \leftarrow M) & \longrightarrow [\ N \mid M,\ NL] \\
s_2 : [N \mid M,\ NL](N \leftarrow K) & \longrightarrow [\ N \mid M,\ NL] \\
s_3 : [N \mid \texttt{root},\ NL](N \leftarrow K) & \longrightarrow [\ N \mid \texttt{root}, NL]
\end{array}
$$

where the state is represented as a multiset (modulo associativity, commutativity, and identity) of nodes and messages. Each node is of the form $[N \mid P,\ L]$ where N is its unique identifier, P is its parent node (initially `none`), and L is the list of its neighbors (their identifiers to be exact). Variable M is of type integer which denotes a *known* parent and consequently cannot be `none` or `root`. The node with "root" as its parent is the root of the spanning tree. Let us assume that the property of interest is "to eventually reach a state in which every node has a parent". This property can be expressed using a single atomic predicate, p, that is false if there is a node with "none" as the parent. One can easily check that the set of rules $S = \{s_1, s_2, s_3\}$ is p-invisible, confluent, terminating modulo associativity, commutativity, and identity, and t_1 is coherent with respect to S modulo the same axioms. Since there are no equations (excluding the axioms) in the theory, one can turn these rules into equations and gain a huge reduction in the state space for model checking. Table 1 shows the performance evaluation of model checking this problem before and after this $\mathcal{R}/S$ reduction using the Maude LTL model checker.

Problem	Number of Nodes	Time	Space	Time (reduced)	Space (reduced)
Leader Election	10	3.6s	27633	0	2
	13	2.7m	506037	0	2
	14	19.3m	1329885	0	2
	15	–	–	0	2
Spanning Tree	3	0.02s	417	0	9
	4	10.2s	120183	0.01s	64
	5	–	–	0.17s	625
	6	–	–	0.5s	1296
	7	–	–	110.22s	117649
	8	–	–	99m	2097152
Client-Server	6	4.0s	125248	0.01	64
	7	81.4s	1753600	0.01s	128
	8	–	–	0.01s	256
	15	–	–	1.8s	32768
	20	–	–	5.3m	1048576

Table 1. Performance Results.

4.3 A Distributed Client-Server System

Consider a system consisting of several clients and one server. The server has a log (a list) for incoming requests. The clients send a message to the server to request a service. When the server receives a request message, it sends the relevant client a message containing the requested material, and adds an entry

to its log (B) to keep track of this communication. The following set of rewrite rules model a simple version of this system:

$$\begin{array}{ll} s_1 : [N \mid M] & \longrightarrow \{N \mid M\}(\texttt{server} \leftarrow (N, M)) \\ s_2 : (\texttt{server} \leftarrow (N, M))[\texttt{server} \mid B] & \longrightarrow [\ \texttt{server} \mid B\ (N, M)](N \leftarrow \texttt{serv}(M)) \\ t_1 : (N \leftarrow \texttt{serv}(M))\{N \mid M\} & \longrightarrow \{N\} \end{array}$$

where the state is a multiset (modulo associativity, commutativity, and identity of multiset union operator "__") of a server, clients, and messages. The server is indicated by identifier `server`. Clients each have an integer identifier N and another integer index M indicating the service they require from the server. Each client sends a message including its identifier and the index of the service to the server. The server replies back and logs the communication in its local list B. Assume that the property of interest is "a client that requires a service will eventually receive it". This property can be expressed by a set P of two atomic predicates, of which one indicates the requirement of the service and the other indicates the receipt. The set $\{s_1, s_2\}$ is P-invisible and a very good candidate for S, but because of the list nature of the buffer, these rules are not confluent. For the property of interest, it does not matter in what order the messages are buffered; but since the resulting buffer is different, confluence does not hold. If one assumes a lexicographical ordering on the buffer (pairwise comparison of the pairs (M, N)), then adding the following rule which always sorts the buffer

$$\begin{array}{rl} s_3 : [\texttt{server} \mid B\ (N, M)\ (N', M')\ B'] \longrightarrow [\ \texttt{server} \mid B\ (N', M')\ (N, M)\ B'] \text{ if} \\ (N' > N) \vee ((N = N') \wedge (M' > M)) \end{array}$$

and makes the set $\widehat{S} = \{s_1, s_2, s_3\}$ confluent and terminating. It is also invisible, and t_1 is coherent with respect to $\widehat{S}$ modulo axioms. Therefore, one can reduce this theory to a theory of the form $\widehat{\mathcal{R}}/\widehat{S}$. Table 1 shows the performance evaluation of model checking this problem before and after reduction using the Maude LTL model Checker.

5 Discharging Proof Obligations

Typically, formal verification efforts using state space reduction techniques involve two separate tasks: (i) model checking the desired properties in the reduced model; and (ii) discharging *proof obligations* ensuring that the proposed reduction is indeed a correct reduction of the original system. We discuss here the proof obligations that must be verified to ensure the correctness of an S-reduction $\mathcal{R}/S$, and ways in which the discharging of such obligations can be assisted by formal tools. For $\mathcal{R}/S$ to be a correct reduction of $\mathcal{R}$ the following proof obligations must be discharged:

1. the rules S must be proved P-invisible;
2. $S \cup E_0$ must be shown confluent and terminating modulo A; and
3. the rules in $R \backslash S$ must be proved locally strongly coherent with respect to the equations $S \cup E_0$ modulo A.

Proving (1) is an inductive theorem proving task. Specifically, it amounts to proving that each state predicate $p \in P$ and also its negation $\neg p$ are both *invariants* for the rewrite theory $(\Sigma, E_0 \cup A, S)$. This can be reduced to proving a series of first-order formulas that must be shown to hold inductively in the equational specification $(\Sigma, E_0 \cup A)$; that is, to be satisfied in the initial model $T_{\Sigma/E_0 \cup A}$. Proofs can be assisted by any first-order inductive theorem prover. For Maude specifications Maude's ITP [8] can be used. The proof obligations for this task become considerably easier if the rules in S are *topmost*, that is, if all rewriting happens at the top of a term. Many rewrite theories whose state is a set or multiset of objects and messages, such as those in the case studies presented in this paper, can be transformed into bisimilar topmost rewrite theories.

Proving (2) can be done mechanically using standard termination and confluence checking tools that support reasoning modulo axioms A such as associativity and commutativity, and can in some cases handle conditional rules. Tools of this kind include, for example, CiME [10] (for both tasks) AProVE [17] (for termination), and for Maude specifications the Maude Termination Tool (MTT) [13] and the Maude Church-Rosser Checker [8].

There is a discussion on proving (3) in [35]. For most combinations of associativity, commutativity and identity axioms in A this task can be checked algorithmically when the rules are linear and unconditional. To the best of our knowledge the only tool available is Maude's Coherence Checker [12], which currently can only reason modulo commutativity axioms.

We now discuss briefly the proof obligations for the $\widehat{\mathcal{R}}/\widehat{S}$ reductions. To begin with, the same proof obligations (1)–(3) must be discharged, but now for $\widehat{\mathcal{R}}/\widehat{S}$ instead of $\mathcal{R}/S$. But that still leaves open the task of coming up with the rules $\widehat{S}$ in the first place. Two approaches are possible for this. On the one hand, as done in the case study of Section 4.3, one can use insight about the given specification to find a suitable $\widehat{S}$. On the other, it is also possible to automatically search for such a set $\widehat{S}$ by performing Knuth-Bendix (KB) completion modulo A on the equations $E_0 \cup S$ using any KB completion tool (modulo A) such as, for example, CiME [10].

6 Related Work and Conclusions

Broadly speaking, our work is related to all other state space reduction and abstraction techniques (see for example [20, 33, 22, 4, 9, 11, 19, 34, 32, 1, 21, 16]). We discuss below several approaches that are most closely related to our own.

Several *partial order reduction* (POR) techniques achieve a reduction to a representative subset of all states while preserving various types of bisimilarity. Some of these techniques [19, 34, 32, 1, 21, 16] exploit the notion of *invisibility*, an idea that is generalized here to arbitrary rewrite theories. A first main difference with the POR approach is that POR techniques are typically *dynamic* (all except [1, 21]), in the sense that the reduction is performed on-the-fly during the model checking and requires substantial changes to the underlying model checking algorithm (see [15, 18] for an exception to this); by contrast, our technique is a *static* method, since we generate the reduced rewrite theory and then model check it. Furthermore, it does not require any changes in the model checker. A second

important difference is in the different levels of generality: POR techniques typically assume a conventional concurrent language with processes and consider invisible process transitions, whereas our approach is much more general: it does not rely on these assumptions, and applies to arbitrary rewrite theories.

Our method has also some similarities with a reduction technique presented in [2]. However, the settings are quite different, because [2] works in the framework of process algebras, whereas our technique works for arbitrary rewrite theories. Furthermore, the notion of invisibility used in [2] is not based on a certain set of predicates. Instead, in our case the invisibility depends on what state predicates are involved in the property that we want to model check. Also, the notion of confluence used in [2] is completely different from ours: we use the standard term-rewriting notion. The notion of coherence used in this work has some similarities with notion of *weak confluence* in [36] if one views the rules in S as τ-transitions. Moreover, their approach is dynamic, while ours is static. The symbolic prioritization in [2] is relevant to our work in two senses: (1) it is static, and (2) it is giving priority to some transitions over the rest, while we also in some sense give priority to some rules over the rest.

Our reduction technique is also closely related to other notions of abstraction and simulation used for reduction purposes in rewriting logic. In the case of *equational abstractions* [29] one begins with a rewrite theory $\mathcal{R} = (\Sigma, E_0 \cup A, R)$ and *adds extra equations* G to it to obtain an abstract theory $\mathcal{R}/G = (\Sigma, E_0 \cup G \cup A, R)$, so that we have a rewrite theory inclusion $\mathcal{R} \subseteq \mathcal{R}/G$. This technique is generalized in [26] to much more general *rewrite theory morphisms* $H : \mathcal{R} \longrightarrow \mathcal{R}'$ that need not be theory inclusions, give rise to simulations, and can be used for model checking purposes when $\mathcal{R}'$ is more abstract than $\mathcal{R}$. Our proposed technique is different from those in [29] and [26]. In our case the relationship between $\mathcal{R}$ and $\mathcal{R}/S$ cannot be understood as a theory *morphism*: it is only a theory *transformation*. This means that we now have a new state space reduction technique for rewrite theories that nicely complements those proposed in [29, 26].

Our technique makes essential use of Viry's notion of coherence [35] in rewrite theories. But we use the notion in precisely the *opposite way* than in Viry's work. The original purpose of coherence is to make a rewrite theory $\mathcal{R} = (\Sigma, E_0 \cup A, R)$ executable by *turning the equations* E_0 *into rules*. Strong coherence then guarantees that $\mathcal{R}$ and the resulting theory $(\Sigma, A, E_0 \cup R)$ are semantically equivalent. We do somehow the opposite: beginning with a rewrite theory $\mathcal{R} = (\Sigma, E_0 \cup A, R)$ we select a subset of rules $S \subseteq R$ and *turn those rules into equations* to obtain our reduced theory $\mathcal{R}/S = (\Sigma, E_0 \cup S \cup A, R \backslash S)$. We then check strong coherence of $\mathcal{R}/S$ for executability and stuttering bisimilarity purposes.

We can summarize our contributions as follows: we have presented a general method to reduce the state space of a concurrent system specified as a rewrite theory $\mathcal{R}$ by selecting a set S of P-invisible transition rewrite rules that, when turned into equations, yield a reduced theory $\mathcal{R}/S$. We have shown that if $\mathcal{R}/S$ satisfies reasonable executability assumptions it is stuttering bisimilar to $\mathcal{R}$ and therefore satisfies the same CTL^*_{-X} formulas under this bisimilarity. Several case studies presented show that $\mathcal{R}/S$ can have a drastically smaller state space

in practice, making it feasible to model check properties for $\mathcal{R}$ by using $\mathcal{R}/S$ instead. We have also presented a method to obtain reductions of this kind using extra invisible rules not present in the original theory $\mathcal{R}$. The proof obligations that must be discharged to guarantee the correctness of our proposed reductions have also been discussed. Discharging them involves reasonable proof tasks that for the most part can be supported by existing formal tools.

This work is part of a broader effort to develop state space reduction techniques of wide applicability for concurrent systems specified as rewrite theories. In this sense, it complements earlier efforts to develop reduction techniques of this kind for rewrite theories [29, 26, 15]. It is however a new technique, different from earlier ones. In future work we plan to further develop the ideas presented here in two opposite directions. In a more general direction, we plan to investigate *weaker conditions* under which invisible transitions S can be used to reduce the state space. In a more specific direction, we plan to apply these techniques to *distributed object systems*, where we hope to exploit the more specific nature of those systems to obtain even more drastic reductions. Two other aspects that need to be further developed are: (i) building a stronger tool environment for checking proof obligations, particularly for checking coherence modulo more general axioms A; and (ii) developing a broader experimental base of case studies.

Acknowledgment. Research funded by ONR grant N00014-02-1-0715.

References

1. R. Alur, R. K. Brayton, T. A. Henzinger, S. Qadeer, and S. K. Rajamani. Partial-order reduction in symbolic state exploration. In *CAV*, volume 1254 of *LNCS*, pages 340 – 351, 1997.
2. Stefan Blom and Jaco van de Pol. State space reduction by proving confluence. In *CAV*, volume 2404 of *LNCS*, pages 596–609, 2002.
3. M.C. Browne, E. M. Clarke, and O. Grumberg. Characterizing finite kripke structures in propositional temporal logic. *Theoretical Computer Science*, 59:115 – 131, 1988.
4. E. M. Clarke, O. Grumberg, and D. E. Long. Model checking and abstraction. *ACM Transactions on Programming Languages and Systems*, 16(5):1512–1542, 1994.
5. Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking*. MIT Press, 2001.
6. M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and J. Quesada. Maude: specification and programming in rewriting logic. *Theoretical Computer Science*, 285:187–243, 2002.
7. M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. Talcott. Maude Manual (Version 2.2). December 2005, http://maude.cs.uiuc.edu.
8. M. Clavel, F. Durán, S. Eker, and J. Meseguer. Building equational proving tools by reflection in rewriting logic. In *Proc. of the CafeOBJ Symposium*, April 1998.
9. M. A. Colón and T. E. Uribe. Generating finite-state abstractions of reactive systems using decision procedures. In *Computer Aided Verification*, volume 1427 of *Lecture Notes in Computer Science*, pages 293–304, 1998.
10. E. Contejean and C. Marché. CiME: Completion modulo E. In *RTA*, volume 1103 of *LNCS*, 1996.
11. D. Dams, R. Gerth, and O. Grumberg. Abstract interpretation of reactive systems. *ACM Transactions on Programming Languages and Systems*, 19:253–291, 1997.

12. F. Durán. Coherence checker and completion tools for Maude specifications. Manuscript, http://maude.cs.uiuc.edu/papers, 2000.
13. F. Durán, S. Lucas, J. Meseguer, C. Marché, and X. Urbain. Proving termination of membership equational programs. In *PEPM'04*, pages 147–158, 2004.
14. S. Eker, J. Meseguer, and A. Sridharanarayanan. The Maude LTL model checker and its implementation. In *SPIN'03*, volume 2648 of *LNCS*, pages 230 – 234, 2003.
15. A. Farzan and J. Meseguer. Partial order reduction for rewriting semantics of programming languages. In *WRLA06*, pages 56–75, 2006.
16. C. Flanagan and P. Godefroid. Dynamic partial order reduction for model checking software. In *Proceedings of POPL*, 2005.
17. J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. Automated termination proofs with AProVE. In *RTA*, volume 3091 of *LNCS*, pages 210–220, 2004.
18. P. Godefroid. Model checking for programming languages using VeriSoft. In *POPL*, volume 174–186, 1997.
19. P. Godefroid and P. Wolper. A partial approach to model checking. In *Proceedings of Logic in Computer Science*, pages 406 – 415, 1991.
20. Y. Kesten and A. Pnueli. Control and data abstraction: The cornerstones of practical formal verification. *International Journal on Software Tools for Technology Transfer*, 4(2):328–342, 2000.
21. R. Kurshan, V. Levin, M. Minea, D. Peled, and H. Yenigun. Static partial order reduction. In *TACAS*, volume 1384, pages 345 – 357, 1998.
22. C. Loiseaux, S. Graf, J. Sifakis, A. Bouajjani, and S. Bensalem. Property preserving abstractions for the verification of concurrent systems. *Formal Methods in System Design*, 6:1–36, 1995.
23. N. A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.
24. P. Manolios. *Mechanical Verification of Reactive Systems*. PhD thesis, University of Texas at Austin, August 2001.
25. N. Martí-Oliet and J. Meseguer. Rewriting logic: roadmap and bibliography. *Theoretical Computer Science*, 285:121–154, 2002.
26. N. Martí-Oliet, J. Meseguer, and M. Palomino. Theoroidal maps as algebraic simulations. In *WADT*, pages 126–143, 2004.
27. J. Meseguer. Conditional rewriting logic as a unified model of concurrency. *Theoretical Computer Science*, 96(1):73–155, 1992.
28. J. Meseguer. Research directions in rewriting logic. In *Computational Logic, NATO Advanced Study Institute, Marktoberdorf*. 1999.
29. J. Meseguer, M. Palomino, and N. Martí-Oliet. Equational abstractions. In *CADE*, volume 2741 of *LNCS*, pages 2–16, 2003.
30. R. De Nicola and F. Vaandrager. Three logics for branching bisimulation. *Journal of ACM*, 42(2), 1995.
31. M. Palomino, J. Meseguer, and N. Martí-Oliet. A categorical approach to simulations. In *CALCO*, pages 313–330, 2005.
32. D. Peled. Combining partial order reduction with on-the-fly model checking. In *CAV*, volume 818 of *LNCS*, pages 377 – 390, 1994.
33. H. Saïdi and N. Shankar. Abstract and model check while you prove. In *Computer Aided Verification*, volume 1633 of *LNCS*, pages 443–454, 1999.
34. A. Valmari. A stubborn attack on state explosion. In *CAV*, volume 531 of *LNCS*, pages 156 – 163, 1990.
35. P. Viry. Equational rules for rewriting logic. *Theoretical Computer Science*, 285:487–517, 2002.
36. M. Ying. weak confluence and tau-inertness. *Theoretical Computer Science*, 238:465–475, 2000.