Model-Checking for a Subclass of Event Structures*

Wojciech Penczek

Institute of Computer Science
Polish Academy of Sciences
01-237 Warsaw, Ordona 21, Poland
email: penczek@ipipan.waw.pl

Abstract. A finite representation of the prime event structure corresponding to the behaviour of a program is suggested. The algorithm of linear complexity using this representation for model checking of the formulas of Discrete Event Structure Logic without past modalities is given. A method of building finite representations of event structures in an efficient way by applying partial order reductions is provided.

1 Introduction

Model checking is one of the most successful methods of automatic verification of program properties. A model-checking algorithm decides whether a finite-state concurrent system satisfies its specification, given as a formula of a temporal logic [3, 10]. Behaviour of a concurrent system can be modeled in two ways. In the *interleaving semantics*, the meaning of a program is an execution tree, temporal-logic assertions are interpreted over paths of this tree. In *partial-order semantics* (or event structure semantics), behaviour is an event structure, where the ordering relations over events reflect the causal dependency and conflict among them [28]. So far model checking algorithms have been suggested for many partial-order temporal logics [23, 26, 1, 14].

There is a long and rich tradition of research that employs the interleaving semantics, resulting in both theoretical and practical results. The main reason for this is the simplicity of the model providing a natural connection with Kripke structures or automata theory. In this framework, a concurrent system P, possibly with fairness requirements, is a Kripke structure M_P that generates the execution tree. The commonly employed specification language is CTL [3, 6]. To check whether the structure M_P satisfies a CTL-formula φ, the model-checking algorithm assigns the subformulas of φ to states of M_P; if the beginning state has been assigned the formula φ, then it holds over M_P.

The use of partial-order semantics is less common, but has attracted researchers in concurrency theory for at least two reasons: It does not distinguish among total-order executions that are *equivalent* up to reordering of independent

* Partially supported by The State Committee for Scientific Research under the grant No. 8 T11C 029 08

transitions, thereby, resulting in a more abstract and faithful representation of concurrency [18, 26, 1, 14]. Logics over partial orders [24, 18, 19, 20, 21] allow a direct representation of properties involving causality, conflict, and concurrency.

The first temporal logic on prime event structures has been put forward by Lodaya and Thiagarajan [9]. Since then several new logics on event structures have been defined [19, 20, 13, 22]. Most of these logics have been proved to be decidable and possessing complete axiomatizations. However, the model checking problem for event structure logics has never been addressed. The reason for this was the lack of two notions: automata on event structures and/or finite representations of event structures.

The main result of this paper is a finite representation of the event structure corresponding to the behaviour of a program and the algorithm using this representation for model checking of the formulas of Discrete Event Structure Logic (DESL) [20] without past modalities. We suggest also a method of building finite representations of event structures in an efficient way by applying partial order reductions [17]. This is the first model checking algorithm for an event structure logic suggested in the literature.

The rest of the paper is organized as follows. In section 2 event structures are defined. Trace systems are introduced in section 3. Logic (DESL) on event structures is given in section 4. Section 5 contains the definition of finite state concurrent programs and its semantics. Finite representation of trace systems and event structures is given in section 6. The correctness of the construction is proved in section 7. Section 8 contains an efficient method of generating finite representations. Model checking of DESL is described in section 9 and the discussion can be found in section 10.

2 Event Structures

We start with the definitions of event structures [28] and trace systems [12], which are used for giving semantics to concurrent systems.

Let S be a countable set, and let $<$ be a binary relation over S. The inverse of $<$ is denoted by $<^{-1}$. For an element $s \in S$, the set $\downarrow_< (s)$ contains the elements $s' \in S$ such that $s' < s$, and $\uparrow_< (s)$ equals $\{s' \in S \mid s < s'\}$.

An element $s \in S$ is *$<$-initial* if $\downarrow_< (s)$ is empty. The relation $<$ is *prefinite* if $\downarrow_< (s)$ is finite for all elements $s \in S$. Let R^* be the reflexive-transitive closure of the relation R. The relation $<$ is *reduced* if for each $s < s'$, $(s, s') \notin (< \setminus (s, s'))^*$. Note that if $<$ is reduced then $<$ is irreflexive, and if $s < s'$ and $s' < s''$ then $s \not< s''$.

Event structures represent a concurrent system by taking occurrences of actions as the starting point. Every occurrence of an action is modelled as a separate event. Two relations are provided that capture, respectively, the (immediate) causality and (immediate) conflict relationship between events.

Definition 1. A four-tuple $(E, E_0, \prec, \#_m)$ is a (discrete prime) event structure (ES, for short) if the following conditions are satisfied:

1. E is a countable set, called a set of *events*,
2. $\prec \subseteq E \times E$ is irreflexive and reduced relation, called *immediate causality relation*,
3. $\leq = \prec^*$ is a prefinite partial order, called *causality relation*,
4. $\#_m \subseteq E \times E$ is an irreflexive and symmetric relation, called *immediate conflict relation*,
5. $\leq \cap \# = \emptyset$ for $\# = (\leq^{-1} \circ \#_m \circ \leq)$, called *conflict relation*,
6. $E_0 \subseteq E$ is the set of $\prec$-initial elements of E.

It follows from the definition that $\# \circ \leq \subseteq \#$. This condition is called *conflict preservation*. Notice that the conflict relation $\#$ is generated by the immediate conflict relation $\#_m$ and the causality relation $\leq$. For event structures giving semantics to concurrent programs, the relation $\#_m$ will be modeled as the minimal relation generating the conflict relation $\#$.

3 Trace Systems

By an *independence alphabet* we mean any ordered pair (Σ, I), where Σ is a finite set of symbols (*operation names*) and $I \subseteq \Sigma \times \Sigma$ is a symmetric and irreflexive binary relation in Σ (the *independence* relation). Let (Σ, I) be an independence alphabet. Define $\equiv$ as the least congruence in the (standard) string monoid $(\Sigma^*, \circ, \epsilon)$ such that $(a,b) \in I \Rightarrow ab \equiv ba$, for all $a, b \in \Sigma$ i.e., $w \equiv w'$, if there is a finite sequence of strings $w_1, \ldots, w_n$ such that $w_1 = w$, $w_n = w'$, and for each $i < n$, $w_i = uabv$, $w_{i+1} = ubav$, for some $(a,b) \in I$ and $u, v \in \Sigma^*$. Equivalence classes of $\equiv$ are called *traces* over (Σ, I). The trace generated by a string w is denoted by $[w]$. We use the following notation: $[\Sigma^*] = \{[w] \mid w \in \Sigma^*\}$. Concatenation of traces $[w], [v]$, denoted $[w][v]$, is defined as $[wv]$.

Now, let T be the set of all traces over (Σ, I). The *successor* relation $\rightarrow$ in T is defined as follows: $[w_1] \rightarrow [w_2]$ iff there is $a \in \Sigma$ such that $[w_1][a] = [w_2]$. The *prefix* relation $\leq$ in T is defined as a reflexive and transitive closure of the successor relation i.e., $\leq = (\rightarrow)^*$. By $<$ we mean $\leq - id_T$. Let $\tau \in T$ and $Q \subseteq T$. We use the following notation: $\downarrow_{\leq} Q = \bigcup_{\tau \in Q} \downarrow_{\leq} \tau$.

We say that a subset Q of T *dominates* another subset R of T, if $R \subseteq \downarrow_{\leq} Q$. Two traces are *consistent*, if there is a trace in T dominating both of them and *inconsistent* otherwise. We shall say that inconsistent traces are in *conflict*.

A set R of traces is said to be *proper* , if any two of its consistent traces are dominated by a trace in R, and *directed* , if arbitrary two traces in R are dominated by a trace in R. A set of traces Q is said to be *prefix-closed*, if $Q = \downarrow_{\leq} Q$.

An ordered pair $(T, \rightarrow)$ is a *trace system* over (Σ, I), if T is a prefix-closed and proper trace language over (Σ, I) and $\rightarrow$ is the prefix relation in T.

For each $w \in \Sigma^*$, let $last(w) = a$ if $w = w'a$. For each trace $\tau \in [\Sigma^*]$ $Max(\tau) = \{last(w) \mid [w] = \tau\}$. A trace τ is called *prime*, if $|Max(\tau)| = 1$, i.e., there is exactly one operation executed last in τ.

4 Logic on Event Structures

We give a definition of Discrete Event Structure Logic [19, 22] without past modalities, for which a model checking algorithm is defined. The language contains modalities corresponding to the relations of (immediate) causality and (immediate) conflict.

4.1 Syntax and Semantics

Let $AP = \{p_1, p_2, \ldots\}$ be a countable set of atomic propositions. The logical connectives $\neg$ and $\wedge$, as well as modalities $\Box$ (causally always), $\otimes$ (all causally next), $\Box_\sharp$ (all in conflict), and $\otimes_\sharp$ (all in immediate conflict) will be used. The set of formulas is built up inductively:

E1. every member of AP is a formula,
E2. if α and β are formulas, then so are $\neg\alpha$ and $\alpha \wedge \beta$,
E3. if α is a formula, then so are $\Box\,\alpha$ and $\Box_\sharp\,\alpha$,
D4. if α is a formula, then so are $\otimes\,\alpha$, and $\otimes_\sharp\,\alpha$,

The following derived logical connectives and modalities are defined:

$$\alpha \vee \beta \stackrel{def}{=} \neg(\neg\alpha \wedge \neg\beta) \qquad \text{(standard)}$$
$$\alpha \Rightarrow \beta \stackrel{def}{=} \neg\alpha \vee \beta \qquad \text{(standard)}$$
$$\Diamond\,\alpha \stackrel{def}{=} \neg\Box\neg\alpha \qquad (\Diamond \text{ - causally sometimes})$$
$$\Diamond_\sharp\,\alpha \stackrel{def}{=} \neg\Box_\sharp\neg\alpha \qquad (\Diamond_\sharp \text{ - some in conflict})$$
$$\bigcirc\,\alpha \stackrel{def}{=} \neg\otimes\neg\alpha \qquad (\bigcirc \text{ - some immediately causal})$$
$$\bigcirc_\sharp\,\alpha \stackrel{def}{=} \neg\otimes_\sharp\neg\alpha \qquad (\bigcirc_\sharp \text{ - some in immediate conflict})$$

Definition 2. An ordered pair $\mathcal{M} = (F, V)$ is a *model*, where $F = (E, E_0, \prec, \#_m)$ is an *ES* and $V : E \longrightarrow 2^{AP}$ is a *valuation function.*

Let $\mathcal{M}$ be a model, $e \in E$ be a state, and α be a formula. $\mathcal{M}, e \models \alpha$ denotes that the formula α is true at the state e in the model $\mathcal{M}$ ($\mathcal{M}$ is omitted, if it is implicitly understood). This notion is defined inductively as follows:

E1. $e \models p$ iff $p \in V(e)$, for $p \in AP$,
E2. $e \models \neg\alpha$ iff not $e \models \alpha$,
$e \models \alpha \wedge \beta$ iff $e \models \alpha$ and $e \models \beta$,
E3. $e \models \Box\,\alpha$ iff $(\forall e' \in E)$ $(e \prec^* e'$ implies $e' \models \alpha)$,
$e \models \Box_\sharp\,\alpha$ iff $(\forall e' \in E)$ $(e\ \sharp_m \circ \prec^*\ e'$ implies $e' \models \alpha)$.
D4. $e \models \otimes\,\alpha$ iff $(\forall e' \in E)$ $(e \prec e'$ implies $e' \models \alpha)$,
$e \models \otimes_\sharp\,\alpha$ iff $(\forall e' \in E)$ $(e\ \sharp_m\ e'$ implies $e' \models \alpha)$.

Notice that $e \models \Box_\sharp\,\alpha$ specifies that α holds at all the states, which are in immediate conlict or in the future of the states that are in immediate conflict with e.

The above semantics is caused by the lack of the past operators. However, our modality $\Box_\sharp$ is still usefull for expressing properties of event structures:

- $\Box\,\alpha$ - α holds in all the states (safety),
- $\Diamond\,\alpha$ - α is possible in the causal future,
- $\Box(\neg \otimes\!\!\!| \; true)$ - the system is conflict-free,
- $\Diamond(\Box(\neg \otimes\!\!\!| \; true))$ - from some state in the future the system is conflict free,
- $\bigcirc(\alpha \wedge \bigcirc\!\!\!| \; \alpha)$ - α will inevitably hold in the next state.

For specifying more properties see [20, 22]. DESL was shown to be decidable and possessing a complete axiomatization [20]. The extension of DESL by an independency operator was considered in [13]. The model checking problem has never been addressed for any of the event structure logics.

5 Finite State Concurrent Programs and their Semantics

Programs are represented by K-sequential agents communicating via executing joint operations.

5.1 Programs

A program is a structure P with the following components:

1. $\mathcal{K}$ is a finite set, called *set of processes*,
2. for each process $i \in \mathcal{K}$, a finite non-empty set S_i, called a *set of local-states of the process* i,
3. for each process $i \in \mathcal{K}$, $s_i^0 \in S_i$ is a distinguished state, called *the initial state of the process* i,
4. for each nonempty process-set $X \subseteq \mathcal{K}$, $\Gamma_X \subseteq (\prod_{i \in X} S_i)^2$ is *transition relation.*

For $s \in \prod_{i \in X} S_i$ and $Y \subseteq X$ let $s|_Y$ denote the projection of s to the Y-components. If $Y = \{i\}$, we write $s|_i$ instead of $s|_{\{i\}}$.

Some of our model checking results will hold only for a restricted class of programs in which the conflicting transitions having at least one beginning state in common belong to the same sequential agents. This class of programs is called *free-choice* (see [4] for the definition for Petri Nets) and it is formally defined as follows:

Definition 3. A program P is said to be *free-choice*, if for each two transitions $t = (s, s_1) \in \Gamma_X, t' = (s', s_1') \in \Gamma_Y$, either $X \cap Y = \emptyset$ or if $s|_i = s'|_i$ for some $i \in X \cap Y$, then $X = Y$ and $s|_i = s'|_i$, for all $i \in X$.

The transition-relation Γ_X models the events in which all processes in X participate. For each transition $t = (s, s_1)$, $proc(t)$ denotes the set of processes involved in executing t, i.e., $proc(t) = X$, if $t \in \Gamma_X$; $in(t) = s$, and $out(t) = s_1$. Let $\Sigma = \bigcup_{X \subseteq \mathcal{K}} \Gamma_X$. The *independency relation* $I \subseteq \Sigma \times \Sigma$ of transitions is defined as follows: $(t, t') \in I$ if $proc(t) \cap proc(t') = \emptyset$. The *dependency relation* $D = \Sigma \times \Sigma \setminus (I \cup id_\Sigma)$. The *immediate dependency relation* $D_m = \{(t, t') \in D \mid$

$in(t)|_i = in(t')|_i$, for some $i \in proc(t) \cap proc(t')\}$. The free choice programs enjoy the following property: if two transitions $(t, t') \in D_m$, then $proc(t) = proc(t')$ and $in(t)|_i = in(t')|_i$, for each $i \in proc(t)$.

Let $GS = \prod_{i \in \mathcal{K}} S_i$ be the set of *global states* of P. A transition $t = (s, s') \in \Gamma_X$ is said to be *enabled* from a global state $g \in GS$ (denoted $t \in enabled(g)$), if $g\,|_X = s$. For the free-choice programs, for each two transitions $(t, t') \in D_m$, if one of them is enabled from a global state, then the other one is also enabled from the global state.

For two global states $g, g' \in GS$, $g \xrightarrow{t} g'$ iff for some $Y \subseteq \mathcal{K}$, $t = (g\,|_Y, g'\,|_Y) \in \Gamma_Y$ and $g\,|_{\mathcal{K}\setminus Y} = g'\,|_{\mathcal{K}\setminus Y}$. An *execution sequence* $w = t_0 \ldots t_n \in \Sigma^*$ of P is a finite sequence of transitions s.t. there is a sequence of global states $\xi = g_0\, g_1\, g_2\, \ldots g_n$ of P with $g_0 = (s_1^0, \ldots, s_K^0)$, and $g_i \xrightarrow{t_i} g_{i+1}$, for each $i < n$. Let $g_0[w > g_n$ denote that the state g_n is reached after executing the execution sequence w from the state g_0. We say that trace $[w]$ *leads* to the state g_n.

Example 1. Program MUTEX is shown in Figure 1. It is composed of three processes, which local states are denoted with circles, whereas the transitions with horizontal bars, e.g. $b = ((3, 8), (5, 10))$. The program ensures the mutual exclusion of access to the local states 5 and 6 being the critical sections. $S_1 = \{1, 3, 5\}$, $S_2 = \{2, 4, 6\}$, $S_3 = \{7, 8, 9, 10, 11\}$, and $s_1^0 = 1$, $s_2^0 = 2$, and $s_3^0 = 7$.

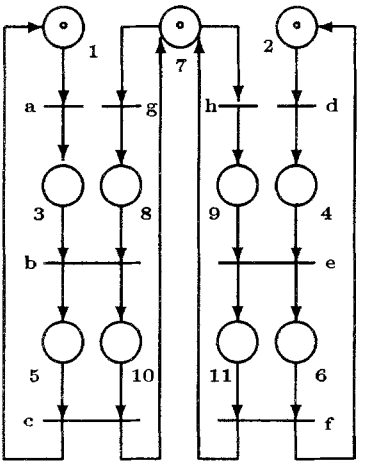


Fig. 1. Program MUTEX

5.2 Trace System Semantics of Programs

The interpreted trace system $TS_P = (T, \rightarrow, \mathcal{I})$ over (Σ, I) and the set of atomic propositions AP is the *trace semantics* of the program P iff the following conditions are satified:

- $[w] \in T$ iff w is an execution sequence of P,
- $\rightarrow$ is the trace successor relation in T,
- $\mathcal{I} : T \longrightarrow 2^{AP}$ is an *interpretation function* such that for each $[w], [w'] \in T$ if $g_0[w > g$ and $g_0[w' > g$, then $\mathcal{I}([w]) = \mathcal{I}([w'])$.

The interpretation function does not distinguish between the traces leading to the same global state.

Example 2. The trace semantics of program MUTEX is shown in Figure 2. The prime traces are printed in bold face.

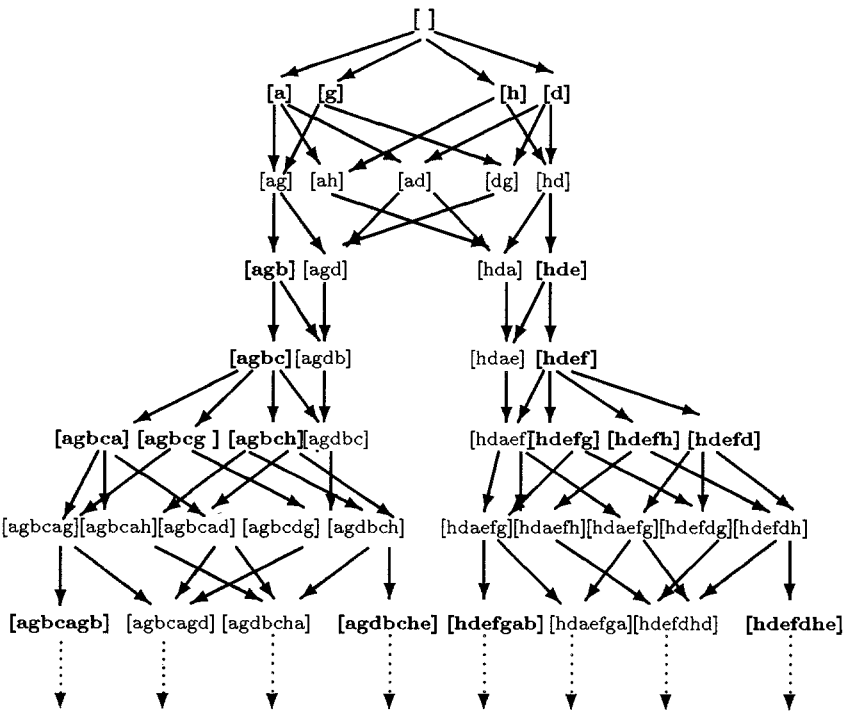


Fig. 2. Trace semantics of MUTEX

5.3 Event Structure Semantics of Programs

Each trace system defines the corresponding event structure [16], where the events are defined as equivalence classes of strings of transitions. For the trace systems that are semantics of the programs defined in Section 5.1, alternatively,

one can identify the events with the prime traces and view the corresponding event structure as a substructure of the original trace system. This approach is taken below.

Definition 4. Let $TS_P = (T, \rightarrow, \mathcal{I})$ be the trace semantics of program P. An interpreted event structure $ES_P = (E, E_0, \prec^E, \#_m^E, V^E)$ is the event structure semantics of program P, where

1. $E = \{e \in T \mid |Max(e)| = 1\}$,
2. $E_0 = \{[\epsilon]\}$,
3. $e \prec^E e'$ if $e \rightarrow^* e'$ and $e \rightarrow^* e'' \rightarrow^* e'$, implies $e = e''$ or $e' = e''$, for each $e'' \in E$,
4. $e \#_m^E e'$ if e, e' are not consistent and for each $\tau \in T$ s.t. $\tau \rightarrow e$ the traces τ, e' are consistent in T and for each $\tau' \in T$ s.t. $\tau' \rightarrow e'$ the traces e, τ' are consistent in T.
5. $V^E(e) = \mathcal{I}(e)$, for $e \in E$.

The condition 3) corresponds to the fact that the relation $\prec^E$ is reduced. The condition 4) specifies that $\#_m^E$ is the minimal relation generating the conflict relation in E.

Lemma 5. *Let P be a free-choice program and let ES_P be the event structure semantics of P. Then, the following condition holds:*

- $e \#_m^E e'$ *iff* $(\exists \tau \in T)\ \tau \rightarrow e$, $\tau \rightarrow e'$, *and* $(Max(e), Max(e')) \in D_m$.

Proof. Consider $e, e' \in E$ that are not consistent (i.e., in conflict) and for each trace $\tau \in T$ s.t. $\tau \rightarrow e$ the traces τ, e' are consistent in T and for each trace $\tau' \in T$ s.t. $\tau' \rightarrow e'$ the traces e, τ' are consistent in T. Then, $proc(Max(e)) \cap proc(Max(e')) \neq \emptyset$. Therefore, there is $e_0 \in E$ such that $e_0 \prec^E e$ and $e_0 \prec^E e'$. This implies that $((Max(e), Max(e')) \in D_m$. Since P is free-choice, $proc(e) = proc(e')$. Thus, $\{f \in E \mid f \prec^E e\} = \{f' \in E \mid f' \prec^E e'\}$. Consider the minimal trace $\tau \in T$ such that $f \rightarrow^* \tau$, for each $f \prec^E e$. Then, $\tau \rightarrow e$ and $\tau \rightarrow e'$. □

Example 3. The event structure semantics of program MUTEX is shown in Figure 3.

Example 4. Consider again our running example. Let the set of propositions contain propositions corresponding to the transitions Σ of program $MUTEX$. For simplicity, we assume the same symbols for these propositions and the transitions, i.e., $\Sigma = \{a, b, c, d, e, f, g, h\} \subseteq AP$. Let $x \in V^E(e)$ if $Max(e) = \{x\}$, for $x \in \Sigma$. Then, the following properties of MUTEX can be specified:

- $\Box \Diamond b \wedge \Box \Diamond e$ -
  each process always can enter its critical section by executing b or e (resp.),
- $\Box(g \Rightarrow \bigcirc(b \wedge \otimes\!\!\!| \; false)) \wedge \Box(h \Rightarrow \bigcirc(e \wedge \otimes\!\!\!| \; false))$,
  after executing g the action b will be inevitably executed in the next step and after executing h the action e will be inevitably executed in the next step,

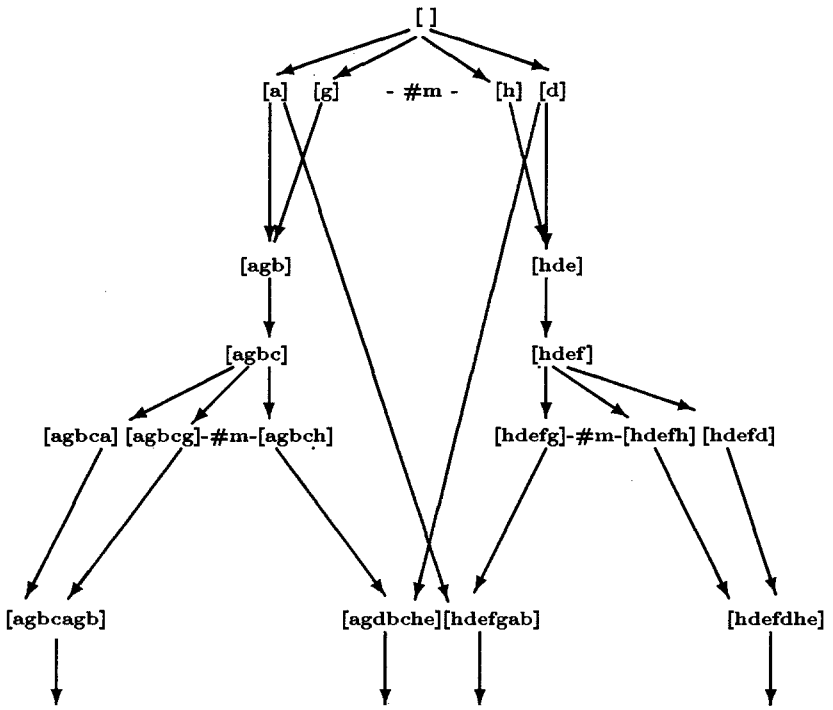


Fig. 3. Event structure semantics of MUTEX

- $\Box(b \Rightarrow \bigcirc c) \wedge \Box(e \Rightarrow \bigcirc f)$,
  each process will always leave its critical section in the next causal step.
- $\Box(g \Rightarrow \Diamond_{\shortparallel} h) \wedge \Box(h \Rightarrow \Diamond_{\shortparallel} g)$,
  the transitions g and h leading to the critical sections are in conflict.

6 Finite Representations of (Prime) Trace Systems

Trace Systems as well as Prime Event Structures are usually infinite objects and as such are not very convenient for model checking. Therefore, one uses quotient structures of TS_P, which identify all the traces that cannot be distinguished by the formulas of a logic of interest.

When one is interested in model checking of CTL formulas, then the finite representation is obtained by applying the equivalence relation $\sim_{CTL}$ on the traces, which identifies the traces leading to the same global states, defined as follows:

$(\forall [w], [w'] \in T)(\forall g \in GS)\ [w] \sim_{CTL} [w']\ iff\ (g_0[w > g \Leftrightarrow g_0[w' > g)$.

When one is interested in model checking of CTL$_P$ formulas, then the finite representation of TS_P is obtained by using a more discriminating equivalence relation (see [21]).

For event structure logics the situation is slightly different since a finite representation should only involve equivalence classes of prime traces. The simplest solution would be, if a substructure of the quotient structure of TS_P by $\sim_{CTL}$ could be applied. This is, unfortunately, not the case since there could be two prime traces leading to the same global state that have different causal futures and therefore can be distinguished by DESL formulas corresponding to the causality relation. In the trace semantics of MUTEX (Figure 2) the traces: $[\,], [agbc], [hdef]$ lead to the same global state $(1,2,7)$, but the sets of global states to which lead their causal successors are different (see Figure 3).

The solution is to sharpen the equivalence relation $\sim_{CTL}$ accordingly. For technical reasons, the new equivalence is defined for all traces. Two traces are ES-equivalent if they lead to the same global states and their sets of maximal transitions are the same. A new equivalence relation $\sim_{ES}$ is formally defined as follows:

- Define $f : T \longrightarrow \Pi_{i\in\mathcal{K}} S_i \times (2^{\Sigma} \cup \{t_0\})$, where $t_0 \notin \Sigma$ is an artificial transition with $proc(t_0) = \mathcal{K}$,
  $f([\epsilon]) = (g_0, t_0)$,
  $f([w]) = (g, X)$ if $g_0[w > g$ and $X = Max([w])$ for $[w] \neq [\epsilon]$, and
- $(\forall [w], [w'] \in T)\ [w] \sim_{ES} [w']$ iff $f([w]) = f([w'])$.

We will write $(g, t_1 ... t_n)$ for $(g, \{t_1, .., t_n\})$ and let $State(\tau)$ stand for g, where $f(\tau) = (g, X)$

As we show later, $\sim_{ES}$ identifies all the prime traces, which cannot be distinguished by DESL formulas without conflict operators. Unfortunately, for unrestricted programs $\sim_{ES}$-equivalent prime traces can be distinguished by conflict modalities. To see this, consider the program Pr shown in Figure 4. The beginning states of the three processes of Pr are marked with dots. The traces $[a], [b], [c], [ac], [ba], [bd], [cd]$ belong to the trace semantics of Pr. All of them except for $[ac]$ are prime. Notice that $f([bd]) = f([cd]) = ((1,7,8), d)$, but $[cd] \#_m^E [a]$, whereas $[bd] \#_m^E [ba]$. Since $f([a]) \neq f([ba])$, the prime traces $[bd]$ and $[cd]$ can be distinguished by a conflict formula, referring to $f([a])$ or $f([ba])$.

There are two possible ways to overcome this problem:

1. Define a more restrictive equivalence relation on T, or
2. Restrict the class of programs such that $\sim_{ES}$ preserves all DESL formulas.

The first solution leads inevitably to an equivalence relation, which index is exponential in the number of the global states and therefore, we do not consider this solution in the present paper.

The second solution requires such a restriction of programs, which makes it possible to define the immediate conflict relation in terms of the preserved by $\sim_{ES}$ causality relation. This is the situation for free-choice programs, which is shown below.

Next, we define the quotient structures of TS_P and ES_P by $\sim_{ES}$.

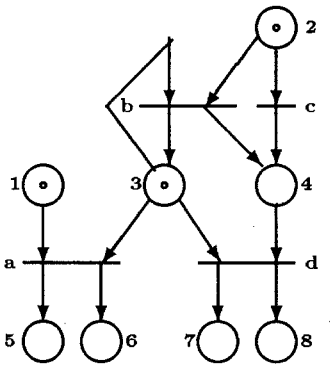


Fig. 4. Program Pr

Definition 6. The quotient structure of TS_P by $\sim_{ES}$ is the triple $F_{TS} = (W, \rightarrow, w_0)$, where:

- $W = \{[\tau]_{\sim_{ES}} \mid \tau \in T\}$ is a set of states,
- $\rightarrow \subseteq W \times \Sigma \times W$ is a transition relation such that $[\tau]_{\sim_{ES}} \xrightarrow{t} [\tau']_{\sim_{ES}}$, if there are traces
  $\tau_1 \in [\tau]_{\sim_{ES}}, \tau_1' \in [\tau']_{\sim_{ES}}$, and $t \in \Sigma$ such that $\tau_1[t] = \tau_1'$,
- $w_0 = [\epsilon]_{\sim_{ES}}$.

Definition 7. The quotient structure of ES_P by $\sim_{ES}$ is the 4-tuple $F_{ES} = (E_F, \prec, \#_m, e_0)$, where:

- $E_F = \{[e]_{\sim_{ES}} \mid e \in E\}$ is a set of states,
- $\prec \subseteq E_F \times E_F$ is a relation such that $[e]_{\sim_{ES}} \prec [e']_{\sim_{ES}}$, if there are prime traces
  $e_1 \in [e]_{\sim_{ES}}, e_1' \in [e']_{\sim_{ES}}$ such that $e \prec^E e'$,
- $\#_m \subseteq E_F \times E_F$ is a relation such that $[e]_{\sim_{ES}} \#_m [e']_{\sim_{ES}}$, if there are prime traces
  $e_1 \in [e]_{\sim_{ES}}, e_1' \in [e']_{\sim_{ES}}$ such that $e_1 \#_m^E e_1'$,
- $e_0 = [\epsilon]_{\sim_{ES}}$.

Example 5. The quotient structure of the trace semantics of MUTEX is shown in Figure 5.

We will refer to the states $[\tau]_{\sim_{ES}}$ of F_{TS} by the values of the function f of their representatives, i.e., $f(\tau)$. Note that F_{TS} can be directly generated from the given program P using the Depth First Search (DFS)-algorithm [25]. The algorithm starts from the beginning state (g_0, t_0) and then, recursively, for a current state (g, X) and enabled transition $t \in \Sigma$ expands the t-successor (g', X') of (g, X). When all the successors of a current state have been already expanded,

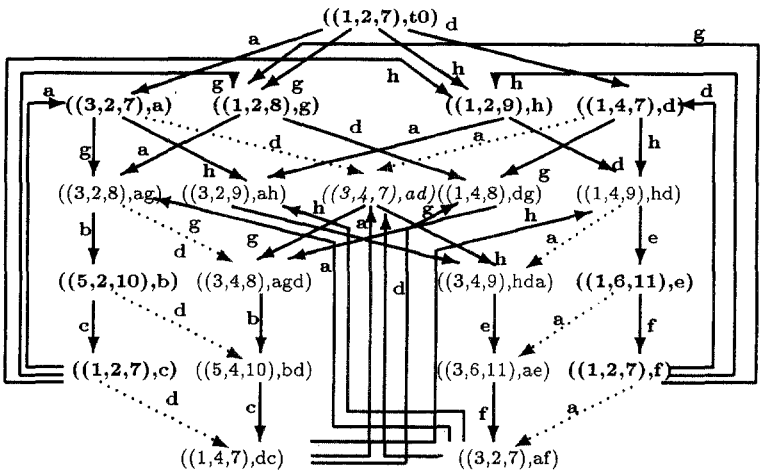


Fig. 5. The quotient structure of the trace semantics of MUTEX

the algorithm backtracks to the nearest state with at least one un-expanded succcessor. (g', X') is computed in the following way: $g' \in GS$: $g[t > g'$ and $X' = (X \cup \{t\}) \setminus \{t' \in \Sigma \mid (t, t') \in D\}$.

The simplest (but ineffective) way of constructing F_{ES} is to extend F_{TS} with the relations $\prec$ and $\#_m$ between the states (g, X) with $|X| = 1$ (using the below lemma) and then to remove from F_{TS} all the states (g, X) with $|X| > 1$.

Lemma 8. *The following two conditions hold:*

*) $(g,t) \prec (g',t')$ *iff there is a sequence* $(g,t) \xrightarrow{t_1} (g_1, X_1) \xrightarrow{t_2} \ldots \xrightarrow{t_{n-1}}$ $(g_{n-1}, X_{n-1}) \xrightarrow{t_n} (g_n, t_n)$ *with* $(g_n, t_n) = (g', t')$ *such that* $(t_i, t) \in I$ *for* $1 \leq i < n$.

**) $(g,t) \#_m (g',t')$ *iff* $\exists (g'', X) \in E_F$ *s.t.* $(g'', X) \xrightarrow{t} (g,t)$, $(g'', X) \xrightarrow{t'} (g', t')$, *and* $(t, t') \in D_m$.

Proof. *). $(g,t) \prec (g',t')$ iff (by definition) there are prime traces $e, e' \in E$ with $f(e) = (g,t)$ and $f(e') = (g',t')$ such that $e \rightarrow^* e'$ and $e \rightarrow^* e'' \rightarrow^* e'$, implies $e = e''$ or $e' = e''$, for each $e'' \in E$, iff there is a sequence of traces $\tau_0 \rightarrow \tau_1 \ldots \rightarrow \tau_n$ with $\tau_0 = e$, $\tau_n = e'$, $\tau_i = \tau_{i-1}[t_i]$ with $t_i \in \Sigma$ and $(t_i, t) \in I$ for $1 \leq i < n$, (Notice that if for some $j < n$ $(t_j, t) \in D$, then it would be a prime trace $f \neq e$ s.t. $e \rightarrow^* f \rightarrow^* \tau_j$ and $Max(f) = \{t_j\}$), iff there is a sequence $(g,t) \xrightarrow{t_1} (g_1, X_1) \xrightarrow{t_2} \ldots \xrightarrow{t_{n-1}} (g_{n-1}, X_{n-1}) \xrightarrow{t_n} (g_n, t_n)$ with $(g_i, X_i) = f(\tau_i)$ and $(g_n, t_n) = (g', t')$ such that $(t_i, t) \in I$ for $1 \leq i < n$.

**) follows directly from Lemma 5. □

7 Correctness of the Construction

We have to show that the formulas of DESL cannot distinguish between two prime traces e and e' with $f(e) = f(e')$. Since the modalities of DESL correspond to the relations expressible in terms of the relations $\prec$ and $\#_m$, this requires proving the following Lemma.

Lemma 9. *For each prime traces $e, e', e'_1 \in E$, the following two conditions hold:*

$\prec^E$) *if $e \sim_{ES} e'$ and $e \prec^E e_1$, then there is $e'_1 \in E$ s.t. $e' \prec^E e'_1$ and $e_1 \sim_{ES} e'_1$,*
$\#^E_m$) *if $e \sim_{ES} e'$ and $e \#^E_m e_1$, then there is $e'_1 \in E$ s.t. $e' \#^E_m e'_1$ and $e_1 \sim_{ES} e'_1$.*

Proof. $\prec^E$). It follows directly from Lemma 8 *).

$\#^E_m$). Let $e \sim_{ES} e'$ and $e \#^E_m e_1$. Then, $(\exists\ \delta, \delta' \in T$ and $t \in \Sigma)$ such that $e = \delta[t]$ and $e' = \delta'[t]$. It is not necessarily the case that $f(\delta) = f(\delta')$, but it is easy to observe that $State(\delta) = State(\delta')$. It follows from the definition of $\#^E_m$ that $e_1 = \delta[t']$ for some $t' \in \Sigma$ with $(t, t') \in D_m$. Let $e'_1 = \delta'[t']$. It is easy to notice that $e'_1 \in T$. Since $\delta'[t] \in E$ and $proc(t) = proc(t')$, so $e'_1 \in E$. It follows from definition of $\#^E_m$ that $e' \#^E_m e'_1$. Next, $State(e_1) = State(e'_1)$ and $Max(e_1) = Max(e'_1) = t'$. Therefore, $e_1 \sim_{ES} e'_1$, which completes the proof. □

Notice that Lemma 9 $\prec^E$) holds for the programs unrestricted to free-choice ones.

Lemma 10. *For each formula $\varphi \in DESL$ and for each $e, e' \in E$ if $f(e) = f(e')$, then $ES_P, e \models \varphi$ iff $ES_P, e' \models \varphi$*

Proof. By induction on the complexity of a formula using Lemma 9. □

Now, we can define the valuation $V_{ES} : [E]_{\sim_{ES}} \longrightarrow 2^{AP}$ of the states of the structure F_{ES} consistent with the valuation of the corresponding traces of ES_P: $V_F([e]_{\sim_{ES}}) = V(e)$ for each $e \in E$. Then, the notion of a formula φ *true* at the state e in the structure $M_{ES} = (F_{ES}, V_{ES})$ (denoted $M_{ES}, e \models \varphi$) is defined inductively as in Definition 2. The following theorem shows that model checking of DESL can be performed over the structure M_{ES}.

Theorem 11. *For each DESL-formula φ and each $e \in E$:*
$ES_P, e \models \varphi$ *iff* $M_{ES}, [e]_{\sim_{ES}} \models \varphi$.

Proof. Follows from the definition of M_{ES} and Lemma 10. □

8 Efficient Method of Generating F_{ES}

The above method of generating F_{ES} can be substantially improved for some programs by applying partial order reduction methods [7, 17].

Despite the equivalence classes of not-prime traces (called *global states*) do not occur in the structure F_{ES}, some of them need to be generated in order to establish whether two equivalence classes of prime traces (called *local states*) are

causally related. The idea of using partial order reductions relies on generating only these necessary global states. For finding out whether $(g,t) \prec (g',t')$, it is sufficient to find a sequence of global states satisfying the condition *) of Lemma 8.

The new algorithm is the adaptation of the DFS-algorithm such that only a subset of transitions enabled at a current state is expanded. This subset (called ample-set) is computed statically at the current state. Let $s = (g,X)$ be a current state with $proc(X) = J$. A transition t is called *J-transition*, if $proc(t) \cap J \neq \emptyset$. Ample(s) has to satisfy the following condition:

C for all non-empty sequences $(g,X) \xrightarrow{t_1} (g_1,X_1) \xrightarrow{t_2} \ldots \xrightarrow{t_{n-1}} (g_{n-1},X_{n-1}) \xrightarrow{t_n} (g_n,t_n)$ such that there is $t \in X$ with $(t_i,t) \in I$ for $1 \leq i < n$, there is $t' \in ample(g,X)$ s.t. $t_i = t'$ for some $i < n$ and $(t',t_j) \in I$ for all $j < i$.

It follows from **C** that t_n is an J-transition and all J-transitions $t \in enabled(s)$ are in $ample(s)$.

Denote the structure generated by the modified DFS-algorithm by $R_{TS} = (R, \rightarrow_r)$. Define the relations $\prec_r, \#_{m,r} \subseteq R \times R$ as follows:

- $(g,t) \prec_r (g',t')$ iff there is a sequence $(g,t) \xrightarrow{t_1}_r (g_1,X_1) \xrightarrow{t_2}_r \ldots \xrightarrow{t_{n-1}}_r (g_{n-1},X_{n-1}) \xrightarrow{t_n}_r (g_n,t_n)$ with $(g_n,t_n) = (g',t')$ such that $(t_i,t) \in I$ for $1 \leq i < n$.
- $(g,t) \#_{m,r} (g',t')$ iff $\exists (g'',X) \in R$ s.t. $(g'',X) \xrightarrow{t}_r (g,t)$, $(g'',X) \xrightarrow{t'}_r (g',t')$, and $(t,t') \in D_m$.

In order to show the correctness of the partial order reduction method we need the following lemma.

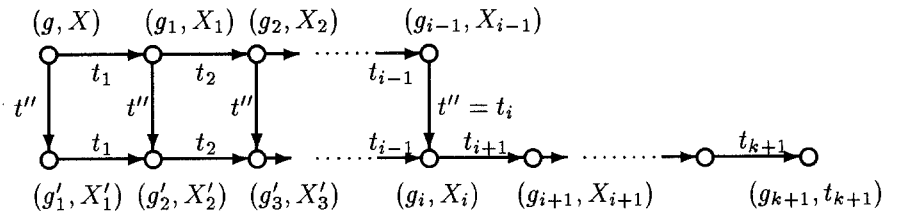
Lemma 12. *If $(g,X) \in R$ with $t \in X$ and there is a sequence $(g,X) \xrightarrow{t_1} (g_1,X_1) \xrightarrow{t_2} \ldots \xrightarrow{t_{n-1}} (g_{n-1},X_{n-1}) \xrightarrow{t_n} (g_n,t_n)$ in F_{ES} such that $(t,t_i) \in I$, for $1 \leq i < n$, then there is a sequence $(g,X) \xrightarrow{t'_1}_r (g'_1,X'_1) \xrightarrow{t'_2}_r \ldots \xrightarrow{t'_{n-1}}_r (g'_{n-1},X'_{n-1}) \xrightarrow{t'_n}_r (g'_n,t'_n)$ in R_{ES} such that $(g'_n,t'_n) = (g_n,t_n)$ and $(t,t'_i) \in I$, for $1 \leq i < n$.*

Proof. By induction on $|n|$. Let $proc(X) = J$.

Base case. $|n| = 1$. Since t_1 is an J-transition, it follows from condition **C** that $t_1 \in ample(g,X)$. So, $(g_1,t_1) \in R$ and $(g,X) \rightarrow_r (g_1,t_1)$.

Induction step. Assume that the lemma holds for all $|n| \leq k$. Let $(g,X) \xrightarrow{t_1} (g_1,X_1) \xrightarrow{t_2} \ldots \xrightarrow{t_{k+1}} (g_{k+1},t_{k+1})$, $t \in X$ with $(t,t_i) \in I$, for $1 \leq i < k+1$. Since t_{k+1} is an J-transition, it follows from condition **C** that there is $t'' \in ample(g,X)$ s.t. $t_i = t''$ for some $i \leq k$ and $(t'',t_j) \in I$ for all $j < i$. Let $(g,X) \xrightarrow{t''}_r (g'_1,X'_1)$, for some (g'_1,X'_1). Since $(t'',t) \in I$ and $t \in X$, $t \in X'_1$. Moreover, there is the sequence $(g'_1,X'_1) \xrightarrow{t_1} (g'_2,X'_2) \xrightarrow{t_2} \ldots (g'_{i-1},X'_{i-1}) \xrightarrow{t_{i-1}} (g_i,X_i) \xrightarrow{t_{i+1}} (g_{i+1},X_{i+1}) \ldots \xrightarrow{t_{k+1}} (g_{k+1},t_{k+1})$ of length k in F_{ES} such that $(t,t_j) \in I$, for

$1 \leq j < i$ and $i < j < k+1$, where $(g_j, X_j) \xrightarrow{t''} (g'_{j+1}, X'_{j+1})$ for $0 < j < i-1$. Thus, the lemma holds by the inductive assumption (see the figure below). □



Theorem 13. *The following three conditions hold:*

1. $(g_0, t_0) \in R$,
2. *if* $(g,t) \in R$ *and* $(g',t') \in F_{ES}$ *such that* $(g,t) \prec (g',t')$, *then* $(g',t') \in R$ *and* $(g,t) \prec_r (g',t')$.
3. *if* $(g,t) \in R$ *and* $(g',t') \in F_{ES}$ *such that* $(g,t) \#_m (g',t')$, *then* $(g',t') \in R$ *and* $(g,t) \#_{m,r} (g',t')$.

Proof. 1) (g_0, t_0) is the beginning state of the modified DFS-algorithm.

2) Follows directly from Lemma 8 *) and Lemma 12 for $X = \{t\}$.

3) If $(g,t) \#_m (g',t')$, then $t \neq t_0$ and $t' \neq t_0$. Thus, there is $(g'', X) \in R$ such that $(g'', X) \xrightarrow{t} (g,t)$. Since $(t,t') \in D_m$, $t' \in enabled(g'')$. Moreover, it follows from $proc(t') \cap proc(X) \neq \emptyset$ that $t' \in ample(g'', X)$. Therefore, $(g'', X) \xrightarrow{t'} (g',t')$, which implies that $(g,t) \#_{m,r} (g',t')$. □

It is easy to see that checking that condition **C** holds for a set of transitions is as hard as checking reachability, hence as hard as the original model-checking algorithm itself (which is in NP-hard for some standard representations of the program).

However, one can benefit from substantial reduction even when using a pessimistic heuristic algorithm that in some cases considers a subset of transitions not to satisfy **C** when it actually does. We suggest the following heuristic method of computing $ample(g, X)$ with $proc(X) = J$.

Define $ample(g, X)$ as the minimal set of transitions satisfying the following conditions:

1. For each J-transition t, if $t \in enabled(g, X)$, then $t \in ample(g, X)$,
2. For each J-transition $t = (s, s') \notin enabled(g, X)$ s.t. $(\exists i \in J)\ s|_i = g|_i$, either there is a transition $t' \in enabled(g, X)$ s.t. $(t, t') \in D$ and $t' \in ample(g, X)$, or $ample(g, X) = enabled(g, X)$,
3. For each transition $t' \in ample(g, X)$, $t'' \in ample(g, X)$ for all transitions t'' s.t. $(t', t'') \in D_m$.

Thus, if there is an J-transition t, which is not enabled at (g, X), but it may become enabled in the future of (g, X), and no transition from the processes to which t belongs is enabled now, then $ample(g, X) = enabled(g, X)$. Notice that condition 3) is correct only for the free-choice programs, but it could be easily adapted to deal with the not-restricted programs.

Lemma 14. *Conditions* $1, 2,$ *and* 3 *imply condition* **C**.

Proof. Consider a non-empty sequence $(g, X) \xrightarrow{t_1} (g_1, X_1) \xrightarrow{t_2} \ldots (g_{n-1}, X_{n-1}) \xrightarrow{t_n} (g_n, t_n)$ such that there is $t \in X$ with $(t_i, t) \in I$ for $1 \leq i < n$. Assume that $ample(g, X) \neq enabled(g, X)$. If $n = 1$, then $t_1 \in ample(g, X)$ (by cond. 1).

Assume that $n > 1$ and let $t_n = (s, s')$. Since t_n is the first transition in the sequence s.t. $(t_n, t) \in D$, $(\exists i \in proc(t) \subseteq J)\ s|_i = g|_i$. If $t_n \notin enabled(g, X)$, then let t' be a transition dependent on t_n such that $t' \in enabled(g, X)$ and $t' \in ample(g, X)$ (by cond. 2). It is easy to notice that it is not possible that all t_i with $i < n$ are independent of t'. Because then, $t' \in enabled(g_{n-1}, X_{n-1})$, which implies that $(t', t_n) \in D_m$. Since $proc(t') = proc(t_n)$, t_n is independent of all t_i with $i < n$, which contradicts with the fact that $t_{n-1} \in X_{n-1}$ and $(g_{n-1}, X_{n-1}) \xrightarrow{t_n} (g_n, t_n)$. If $t_n \in enabled(g, X)$, then $t_n \in ample(g, X)$ (by cond. 1). By repeating the same argument as before, we show that there is $i < n$ s.t. $(t_i, t_n) \in D$. In this case let $t' = t_n$.

Next, consider the smallest index i s.t. $(t', t_i) \in D$. Since $(t', t_j) \in I$ for all $j < i$, $t' \in enabled(g_{i-1}, X_{i-1})$. Thus, $(t', t_i) \in D_m$ and $(t_i, t_j) \in I$ for all $j < i$. This implies that $t_i \in enabled(g, X)$ and $t_i \in ample(g, X)$ (by cond. 3). Therefore, the condition **C** is satisfied. □

Example 6. For our running example, the modified DFS-algorithm would generate the structure shown in Figure 5, without the transitions marked with the dotted lines. In this case, the partial order reduction method did not reduce any states, but only transitions. We give an example showing that substantial reduction of states is also possible.

9 Model Checking of DESL

Assume that the modified DFS-algorithm has generated the reduced structure $R_{TS} = (R, \rightarrow_r)$ for program P. In order to obtain the structure F_{ES}, we have to construct the relations $\prec_r$ and $\#_{m,r}$ between the states $(g, X) \in R$ with $|X| = 1$ and then to remove from R_{TS} all the states (g, X) with $|X| > 1$. Constructing $\#_{m,r}$ is straightforward, whereas constructing $\prec_r$ could be of quadratic complexity in the size of R_{TS}. Therefore, from practical point of view, it is more efficient to perform model checking over the structure R_{TS} itself. Our method relies on translating DESL causality formulas into CTL formulas, interpreted over R_{TS}. Then, the model checking algorithm for CTL [3, 27] applies.

First, we have to extend R_{TS} by a valuation function and make the definition of the set of atomic propositions a bit more precise. So, assume that AP is a

finite set of propositions, which contains the special proposition *prime*, which is used to mark the "prime" states in R_{TS} and propositions corresponding to the transitions Σ of program P. For simplicity, we assume the same symbols for these propositions and the transitions. Therefore, $\{prime\} \cup \Sigma \subseteq AP$. Define $M_R = (R_{TS}, V_R)$ as follows:

- $V_R : R \longrightarrow 2^{AP}$ such that $X \subseteq V_R(g, X)$, for $X \subseteq AP$, and $prime \in V_R(g, X)$ iff $|X| = 1$.

Since F_{ES} is a substructure of R_{TS}, we can define $M_R, (g,t) \models \varphi$ iff $M_{ES}, (g,t) \models \varphi$ for each DESL formula φ. In order to give the traslation from DESL causality formulas to CTL, we have to define the semantics of the CTL operator $EX.$ and $E(.Until.)$:

- $M_R, s \models EX\varphi$ iff there is a state s_1 s.t. $s \rightarrow_r s_1$ and $M_R, s_1 \models \varphi$
- $M_R, s \models E(\psi U \varphi)$ iff there is a sequence of states $s_0, s_1, \ldots$ such that $s_0 = s$, $s_i \rightarrow_r s_{i+1}$, for $i \geq 0$, and $(\exists i \geq 0)\ s_i \models \varphi$ and $(\forall j : 0 \leq j < i)\ s_j \models \psi$.

The translation is defined as follows:

- $M_R, (g,t) \models {}^{\otimes}\varphi$ iff $M_R, (g,t) \models \neg EXE(t\ Until\ (\neg\varphi \wedge prime))$,
- $M_R, (g,t) \models \Box\varphi$ iff $M_R, (g,t) \models \neg E(true\ Until\ (\neg\varphi \wedge prime))$.

The correctness of the translation of ${}^{\otimes}\alpha$ follows from the following observation. Notice that the condition *) of Lemma 8 could be equivalently reformulated by replacing $(t_i, t) \in I$ by $t \in X_i$ i.e., $(g,t) \prec (g',t')$ iff there is a sequence $(g,t) \stackrel{t_1}{\rightarrow} (g_1, X_1) \stackrel{t_2}{\rightarrow} \ldots \stackrel{t_{n-1}}{\rightarrow} (g_{n-1}, X_{n-1}) \stackrel{t_n}{\rightarrow} (g_n, t_n)$ with $(g_n, t_n) = (g', t')$ such that $t \in X_i$ for $1 \leq i < n$. Consider a sequence $(g,t) \stackrel{t_1}{\rightarrow} (g_1, X_1) \stackrel{t_2}{\rightarrow} \ldots \stackrel{t_{n-1}}{\rightarrow} (g_{n-1}, X_{n-1}) \stackrel{t_n}{\rightarrow} (g_n, t_n)$. If $(t_i, t) \in I$ for $1 \leq i < n$, then obviously $t \in X_i$ for $1 \leq i < n$.

On the other hand if $t \in X_i$, then either $(t_i, t) \in I$ or $t_i = t$ for $1 \leq i < n$. In the latter case, $(g_{i-1}, X_{i-1}) = (g_i, X_i)$ as t enables t. Therefore, if we remove from the sequnece $(g,t) \stackrel{t_1}{\rightarrow} (g_1, X_1) \stackrel{t_2}{\rightarrow} \ldots \stackrel{t_{n-1}}{\rightarrow} (g_{n-1}, X_{n-1}) \stackrel{t_n}{\rightarrow} (g_n, t_n)$ all the transitions $t_i = t$ for $1 \leq i < n$, then it would hold for the sequence of remaining states and transitions t_i that $(t_i, t) \in I$.

The correctness of the translation of $\Box\,\alpha$ follows from the definition of $\prec_r^*$.

Given a DESL formula φ. First, replace each occurrence of $\Box_{\|}$ modality by the semantically equivalent combination of ${}^{\otimes_{\|}}\Box$. Next, the model checking method relies on assigning the subformulas of the formula φ to the local states of M_R. The method is inductive, i.e., starting from the shortest and most deeply nested subformula ψ of φ the algorithm labels with ψ these local states of M_R, which are equivalent classes of the prime traces at which ψ holds. Therefore, in case of checking a less nested subformula, it can be assumed that the states have just been labelled with all its subformulas.

The appriopriate algorithms for labelling states with the causality subformulas are standard (see [3, 27, 21]) due to the translation to CTL formulas. The

algorithms for the immediate conflict subformulas are based on the following principle:

- $M_R, (g,t) \models \otimes_{\sharp} \varphi$ iff for all $(g',t') \in R$, if $(g,t) \#_{m,r} (g',t')$, then $M_R, (g',t') \models \varphi$,

Complexity of Model Checking

Notice that checking a formula containing a subformula $\otimes \psi$ requires labelling states of M_R with the formulas $EXE(t\ Until\ (\neg\psi \wedge prime))$ for all $t \in \Sigma$. Therefore, we have the following theorem:

Theorem 15. *The complexity of the model checking algorithm of a formula φ over program P is $O(|M_R| \times (m \times |\Sigma| + (|\varphi| - m)))$, where m is the number of the $\otimes$-subformulas of φ and $|M_R| \leq (|GS| \times |C(I)|) + |\rightarrow|$ with $C(I) = \{A \subseteq \Sigma \mid A \times A \subseteq I\}$ (the set of I-cliques of Σ).*

Experiments with applying partial order reductions show (see [17, 7]) that one can expect M_R to be much smaller than its upper bound.

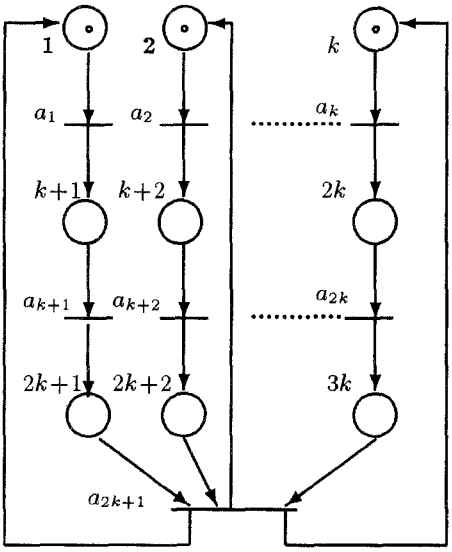


Fig. 6. The global state space contains 3^k states, whereas the reduced structure only $2k^2$ states, for k-processes.

The model checking algorithm for DESL has not yet been implemented. However, it is possible to calculate the number of the global states GS and the

number of the states in the reduced structures R_{TS} for several "toy" programs, for which substantial reductions in the number of states can be obtained. A simple program is given in Figure 6.

10 Discussion

We have suggested a model checking algorithm for Discrete Event Structure Logic without past modalities. So far model checking algorithms have been given for linear time partial order temporal logics: (μ)TrPTL [14, 15] and TLC [1]. Our algorithm is the first one, which is designed for a logic interpreted on event structures. It is also the first model checking algorithm for a partial order logic, which is linear in the number of subformulas of a checked formula. As far as efficient generation of quotient structures of event structures is concerned, it seems possible to apply also net unfolding methods [11]. It is important to mention that for unrestricted programs Lemma 9 ($\prec^E$) still holds making it possible to model check the DESL formulas without conflict operators. Moreover our method can be applied to model checking of more expressive languages, like modal μ-calculus, interpreted over prime event structures. This approach will be described in a forthcoming paper.

References

1. R. Alur, D. Peled, and W. Penczek, Model-Checking of Causality Properties, Proc. of LICS'95, pp. 90–100, 1995
2. L. Bolc, A. Szałas, eds., *Time and Logic: A Computational Approach*, UCL Press Ltd., London, 1995.
3. E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite state concurrent systems using temporal logic specifications: A practical approach. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.
4. J. Desel and J. Esparza, Free choice Petri Nets, Cambridge tracts in TCS 40, Cambridge University Press, 1995.
5. V. Diekert and G. Rozenberg, editors. *Book of Traces.* World Scientific, Singapore. 1995.
6. E.A. Emerson. Temporal and Modal Logic. In J. van Leeuven, editor, Handbook of Theoretical Computer Science, Volume B: Formal Methods and Semantics, The MIT Press Elsevier, pp. 995–1067, 1990.
7. R. Gerth, R. Kuiper, D. Peled, and W. Penczek, A partial order approach to branching time logic model checking, Proc. of the Israeli Conference on Theoretical Computer Science, IEEE Computer Society Press, pp. 130–139, 1995.
8. M. Huhn and P. Niebert, Towards automata for branching time and partial order, Proc. of CONCUR'96, LNCS 1119, pp. 611–626, 1996.
9. K. Lodaya and P.S. Thiagarajan, A modal logic for a subclass of event structures, LNCS 267, Springer-Verlag, pp. 290–303, 1987.
10. O. Lichtenstein, A. Pnueli, Checking that finite-state concurrent programs satisfy their linear specification. *Proc. 11th ACM POPL*, pp. 97–107, 1984.
11. K.L. McMillan, A technique of a state space search based on unfolding. Formal Methods in System Design 6 (1), pp. 45–65, 1995.

12. A. Mazurkiewicz, Basic notions of trace theory, LNCS 354, pp. 285–363, 1988.
13. M. Mukund, P.S. Thiagarajan, An Axiomatization of Well Branching Prime Event Structures. *Theoretical Computer Science* **96**, pp. 35–72, 1992.
14. M. Mukund and P.S. Thiagarajan, Linear time temporal logics over Mazurkiewicz traces, Proceedings of MFCS'96, LNCS 1113, pp. 62–92, 1996.
15. P. Niebert, A μ-calculus with local views for systems of sequential agents, Proc. of MFCS'95, LNCS 969, pp. 563-573, 1995.
16. M. Nielslen and G. Winskel, Trace structures and other models for concurrency, a chapter in [5].
17. D. Peled, Partial order reductions: model-checking using representatives, *Proc. of MFCS'96*, LNCS 1113, pp. 93–112, 1996.
18. D. Peled, A. Pnueli, Proving partial order properties. *Theoretical Computer Science* **126**, 143–182, 1994.
19. W. Penczek, A temporal logic for event structures, *Fundamenta Informaticae* XI, pp. 297–326, 1988.
20. W. Penczek, A Temporal Logic for the Local Specification of Concurrent Systems. *Information Processing IFIP-89*, pp. 857– 862, 1989.
21. W. Penczek, Temporal logics on trace systems: on automated verification, *International Journal of Foundations of Computer Science*, Vol. 4 No. 1, pp. 31–67, 1993.
22. W. Penczek, Branching time and partial order in temporal logics, chapter 4 in [2].
23. W. Penczek and R. Kuiper, "Traces and Logic", a chapter in [5],
24. S. Pinter and P. Wolper, A temporal logic for reasoning about partially ordered computations. *Proc. 3rd ACM PODC*, 28– 37, 1984.
25. R.E. Tarjan, Depth first search and linear graph algorithms, SIAM Journal of Computing, 1(2), pp. 146–160, 1972.
26. P.S. Thiagarajan, A Trace Based Extension of Linear Time Temporal Logic. *Proc. 10th IEEE LICS*, pp. 438–447, 1994.
27. B. Vergauwen and J. Levi, A linear local model checking algorithm for CTL, Proc. of CONCUR'93, LNCS 715, pp. 447–461, 1993.
28. G. Winskel, Event structures. in: W. Brauer, W. Reisig, G. Rozenberg (eds.), *Advances in Petri Nets 1986*, LNCS 255, pp. 279–324, 1987.