Fundamentals of Partial Order Reduction

Presented in University of Tokyo

Subodh Sharma

2019-07-02

Fundamentals of Partial Order Reduction

Fundamentals of Partial Order Reduction
Presented in University of Tokyo
Subodh Sharma

Introduction

- System modeling: common to use interleaving semantics (IS)

Introduction

- System modeling: common to use interleaving semantics (IS)

Introduction

- System modeling: common to use interleaving semantics (IS)
- IS distinguishes the order of execution of instructions
  - leads to explosion in the # of executions

Introduction

- System modeling: common to use interleaving semantics (IS)
- IS distinguishes the order of execution of instructions
  - leads to explosion in the # of executions

1. Intractable because of interleaved executions – give the example of 5 thrs and 5 instructions each

Introduction

- System modeling: common to use interleaving semantics (IS)
- IS distinguishes the order of execution of instructions
  - leads to explosion in the # of executions
- Model checking of concurrent systems often becomes intractable

Question?

Do we really need to distinguish executions on the order of events?

Introduction

- System modeling: common to use interleaving semantics (IS)
- IS distinguishes the order of execution of instructions
  - leads to explosion in the # of executions
- Model checking of concurrent systems often becomes intractable

Question?
Do we really need to distinguish executions on the order of events?

Introduction

- System modeling: common to use interleaving semantics (IS)
- IS distinguishes the order of execution of instructions
  - leads to explosion in the # of executions
- Model checking of concurrent systems often becomes intractable

Question?

Do we really need to distinguish executions on the order of events?

Hint!

A pair of events may be **independent** of each other

Introduction

- System modeling: common to use interleaving semantics (IS)
- IS distinguishes the order of execution of instructions
  - leads to explosion in the # of executions
- Model checking of concurrent systems often becomes intractable

Question?
Do we really need to distinguish executions on the order of events?

Hint!
A pair of events may be **independent** of each other

In a sense, order of execution independent events is immaterial for certain properties. Any order will lead to equivalent executions. Main observation in partial order reduction is that often the property to be checked does not distinguish between executions that only differ in the order of independent events.

Introduction

- System modeling: common to use interleaving semantics (IS)
- IS distinguishes the order of execution of instructions
  - leads to explosion in the # of executions
- Model checking of concurrent systems often becomes intractable

Question?

Do we really need to distinguish executions on the order of events?

Hint!

A pair of events may be **independent** of each other

- Thus, representatives of equivalent executions should suffice!

Introduction

- System modeling: common to use interleaving semantics (IS)
- IS distinguishes the order of execution of instructions
  - leads to explosion in the # of executions
- Model checking of concurrent systems often becomes intractable

Question?
Do we really need to distinguish executions on the order of events?

Hint!
A pair of events may be independent of each other

- Thus, representatives of equivalent executions should suffice!

In a sense, order of execution independent events is immaterial for certain properties. Any order will lead to equivalent executions. Main observation in partial order reduction is that often the property to be checked does not distinguish between executions that only differ in the order of independent events.

Introduction - cont'd

- Partial order reduction is a state reduction technique

Introduction - cont'd

- Partial order reduction is a state reduction technique

Introduction - cont'd

- Partial order reduction is a state reduction technique
  - $TS \models \phi \rightarrow TS' \models \phi$

Introduction - cont'd

- Partial order reduction is a state reduction technique
  - $TS \models \phi \rightarrow TS' \models \phi$

Introduction - cont'd

- Partial order reduction is a state reduction technique
  - $TS \models \phi \rightarrow TS' \models \phi$
- Q: is it applicable for any property?

Introduction - cont'd

- Partial order reduction is a state reduction technique
  - $TS \models \phi \rightarrow TS' \models \phi$
- Q: is it applicable for any property?

Introduction - cont'd

- Partial order reduction is a state reduction technique
  - $TS \models \phi \rightarrow TS' \models \phi$
- Q: is it applicable for any property?
  - A subset of linear time properties (LTL$_X$, etc.)

Introduction - cont'd

- Partial order reduction is a state reduction technique
  - $TS \models \phi \rightarrow TS' \models \phi$
- Q: is it applicable for any property?
  - A subset of linear time properties (LTL$_X$, etc.)

1. LTL-X is a linear time property: i.e. language of infinite concatenation of words in 2^{AP}.
2. Example of an invariant: No state with a data race!

Introduction - cont'd

- Partial order reduction is a state reduction technique
  - $TS \models \phi \rightarrow TS' \models \phi$
- Q: is it applicable for any property?
  - A subset of linear time properties (LTL$_X$, etc.)
  - We shall restrict ourselved to invariants

Introduction - cont'd

- Partial order reduction is a state reduction technique
  - $TS \models \phi \rightarrow TS' \models \phi$
- Q: is it applicable for any property?
  - A subset of linear time properties (LTL$_X$, etc.)
  - We shall restrict ourselved to invariants

1. LTL-X is a linear time property: i.e. language of infinite concatenation of words in 2^{AP}.
2. Example of an invariant: No state with a data race!

System Modeling

M is a LTS $(S, I, A, \rightarrow, AP, L)$

- S is a *finite* set of states
- I is a finite set of *initial* states
- A is a finite set of actions
- $\rightarrow : S \times A \mapsto S$ is a **partial** transition function
- AP is a finite set of boolean propositions
- $L : S \mapsto 2^{AP}$ is a labelling function

System Modeling

M is a LTS $(S, I, A, \rightarrow, AP, L)$
- S is a *finite* set of states
- I is a finite set of *initial* states
- A is a finite set of actions
- $\rightarrow : S \times A \mapsto S$ is a **partial** transition function
- AP is a finite set of boolean propositions
- $L : S \mapsto 2^{AP}$ is a labelling function

1. TS is assumed to be *action-deterministic*; although this is not a severe restriction!
2. We shall restrict our discussions with *finite traces*

More Definitions

Enabled set, Target state, Independence

- $En(s) = \{a \in Act \mid \exists s'.s \xrightarrow{a} s'\}$

More Definitions
Enabled set, Target state, Independence
- $En(s) = \{a \in Act \mid \exists s'.s \xrightarrow{a} s'\}$

More Definitions

Enabled set, Target state, Independence

- $En(s) = \{a \in Act \mid \exists s'.s \xrightarrow{a} s'\}$
- $\alpha(s) = s'$ s.t. $s \xrightarrow{\alpha} s'$

More Definitions

Enabled set, Target state, Independence

- $En(s) = \{a \in Act \mid \exists s'.s \xrightarrow{a} s'\}$
- $\alpha(s) = s'$ s.t. $s \xrightarrow{\alpha} s'$

More Definitions

Enabled set, Target state, Independence

- $En(s) = \{a \in Act \mid \exists s'.s \xrightarrow{a} s'\}$
- $\alpha(s) = s'$ s.t. $s \xrightarrow{\alpha} s'$
- $I \subseteq A \times A$ is a symmetric and antireflexive s.t. $\forall s \in S, \alpha, \beta \in I$

More Definitions

Enabled set, Target state, Independence

- $En(s) = \{a \in Act \mid \exists s'.s \xrightarrow{a} s'\}$
- $\alpha(s) = s'$ s.t. $s \xrightarrow{\alpha} s'$
- $I \subseteq A \times A$ is a symmetric and antireflexive s.t. $\forall s \in S, \alpha, \beta \in I$

1. Give an example from the book!
2. Intuitively, the pair of actions α and β with $\alpha \neq \beta$ is independent when these actions access disjoint variables.
3. Dependence $D = A \times A \setminus I$
4. Note that enabledness condition does not allow disabling of transitions but allows enabling.
5. Defn. of I which disallows enabling and disabling: if $\alpha in En(s)$ and $s \xrightarrow{\alpha} s'$, then $\beta in En(s)$ iff $\beta in En(s')$ [used in persistent sets]
6. Easy syntactic checks to discover valid dependence relation

More Definitions

Enabled set, Target state, Independence

- $En(s) = \{a \in Act \mid \exists s'. s \xrightarrow{a} s'\}$
- $\alpha(s) = s'$ s.t. $s \xrightarrow{\alpha} s'$
- $I \subseteq A \times A$ is a symmetric and antireflexive s.t. $\forall s \in S, \alpha, \beta \in I$
  - Enabledness: If $\alpha, \beta \in En(s)$, then $\alpha \in En(\beta(s)) \wedge \beta \in En(\alpha(s))$

More Definitions
Enabled set, Target state, Independence
$En(s) = \{a \in Act \mid \exists s'. s \xrightarrow{a} s'\}$
$\alpha(s) = s'$ s.t. $s \xrightarrow{\alpha} s'$
$I \subseteq A \times A$ is a symmetric and antireflexive s.t. $\forall s \in S, \alpha, \beta \in I$
Enabledness: If $\alpha, \beta \in En(s)$, then $\alpha \in En(\beta(s)) \wedge \beta \in En(\alpha(s))$

1. Give an example from the book!
2. Intuitively, the pair of actions α and β with $\alpha \neq \beta$ is independent when these actions access disjoint variables.
3. Dependence $D = A \times A \setminus I$
4. Note that enabledness condition does not allow disabling of transitions but allows enabling.
5. Defn. of I which disallows enabling and disabling: if $\alpha inEn(s)$ and $s \xrightarrow{\alpha} s'$, then $\beta inEn(s)$ iff $\beta inEn(s')$ [used in persistent sets]
6. Easy syntactic checks to discover valid dependence relation

More Definitions

Enabled set, Target state, Independence

- $En(s) = \{a \in Act \mid \exists s'.s \xrightarrow{a} s'\}$
- $\alpha(s) = s'$ s.t. $s \xrightarrow{\alpha} s'$
- $I \subseteq A \times A$ is a symmetric and antireflexive s.t. $\forall s \in S, \alpha, \beta \in I$
  - Enabledness: If $\alpha, \beta \in En(s)$, then $\alpha \in En(\beta(s)) \wedge \beta \in En(\alpha(s))$
  - Commutativity: $\alpha(\beta(s)) = \beta(\alpha(s))$

More Definitions
Enabled set, Target state, Independence
- $En(s) = \{a \in Act \mid \exists s'.s \xrightarrow{a} s'\}$
- $\alpha(s) = s'$ s.t. $s \xrightarrow{\alpha} s'$
- $I \subseteq A \times A$ is a symmetric and antireflexive s.t. $\forall s \in S, \alpha, \beta \in I$
  - Enabledness: If $\alpha, \beta \in En(s)$, then $\alpha \in En(\beta(s)) \wedge \beta \in En(\alpha(s))$
  - Commutativity: $\alpha(\beta(s)) = \beta(\alpha(s))$

1. Give an example from the book!
2. Intuitively, the pair of actions α and β with $\alpha \neq \beta$ is independent when these actions access disjoint variables.
3. Dependence $D = A \times A \setminus I$
4. Note that enabledness condition does not allow disabling of transitions but allows enabling.
5. Defn. of I which disallows enabling and disabling: if $\alpha inEn(s)$ and $s \xrightarrow{\alpha} s'$, then $\beta inEn(s)$ iff $\beta inEn(s')$ [used in persistent sets]
6. Easy syntactic checks to discover valid dependence relation

Permuting Independent Actions

Lemma 1

Let $\rho : s = s_0 \xrightarrow{\beta_1} s_1 \xrightarrow{\beta_2} s_2 \cdots \xrightarrow{\beta_n} s_n$ be an execution fragment, then for any $\alpha \in En(s)$ s.t. $\forall i \in \{1, \cdots, n\}(\alpha, \beta_i) \in I$ we have

- $\alpha \in En(s_i)$

Permuting Independent Actions
Lemma 1
Let $\rho : s = s_0 \xrightarrow{\beta_1} s_1 \xrightarrow{\beta_2} s_2 \cdots \xrightarrow{\beta_n} s_n$ be an execution fragment, then for any $\alpha \in En(s)$ s.t. $\forall i \in \{1, \cdots, n\}(\alpha, \beta_i) \in I$ we have
- $\alpha \in En(s_i)$

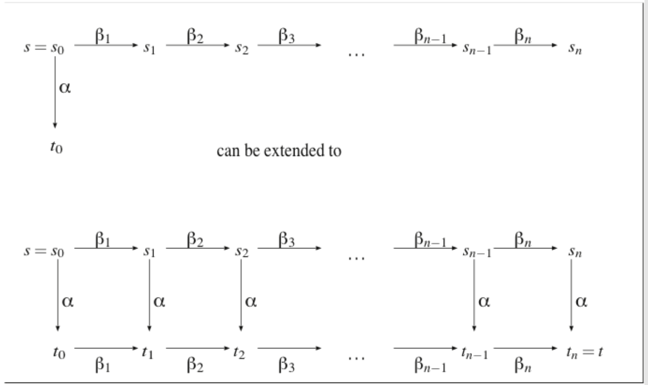


Figure 8.4: Permuting α with the independent actions β_1 through β_n.

Permuting Independent Actions

Lemma 1

Let $\rho : s = s_0 \xrightarrow{\beta_1} s_1 \xrightarrow{\beta_2} s_2 \cdots \xrightarrow{\beta_n} s_n$ be an execution fragment, then for any $\alpha \in En(s)$ s.t. $\forall i \in \{1, \cdots, n\} (\alpha, \beta_i) \in I$ we have

- $\alpha \in En(s_i)$
- $\rho' : s = s_0 \xrightarrow{\alpha} t_0 \xrightarrow{\beta_1} t_1 \cdots \xrightarrow{\beta_n} t_n$ is an execution fragment of the same TS.

Equivalence of executions

When can we say that $\rho \triangleq \rho'$? *Note ρ, ρ' may not be of the same length!*

Permuting Independent Actions
Lemma 1
Let $\rho : s = s_0 \xrightarrow{\beta_1} s_1 \xrightarrow{\beta_2} s_2 \cdots \xrightarrow{\beta_n} s_n$ be an execution fragment, then for any $\alpha \in En(s)$ s.t. $\forall i \in \{1, \cdots, n\} (\alpha, \beta_i) \in I$ we have
- $\alpha \in En(s_i)$
- $\rho' : s = s_0 \xrightarrow{\alpha} t_0 \xrightarrow{\beta_1} t_1 \cdots \xrightarrow{\beta_n} t_n$ is an execution fragment of the same TS.
Equivalence of executions
When can we say that $\rho \triangleq \rho'$? Note ρ, ρ' may not be of the same length!

$$\begin{tikzcd}
s = s_0 \arrow[r, "\beta_1"] \arrow[d, "\alpha"] & s_1 \arrow[r, "\beta_2"] & s_2 \arrow[r, "\beta_3"] & \cdots \arrow[r, "\beta_{n-1}"] & s_{n-1} \arrow[r, "\beta_n"] & s_n \\
t_0 & & & & &
\end{tikzcd}$$

can be extended to

$$\begin{tikzcd}
s = s_0 \arrow[r, "\beta_1"] \arrow[d, "\alpha"] & s_1 \arrow[r, "\beta_2"] \arrow[d, "\alpha"] & s_2 \arrow[r, "\beta_3"] \arrow[d, "\alpha"] & \cdots \arrow[r, "\beta_{n-1}"] & s_{n-1} \arrow[r, "\beta_n"] \arrow[d, "\alpha"] & s_n \arrow[d, "\alpha"] \\
t_0 \arrow[r, "\beta_1"'] & t_1 \arrow[r, "\beta_2"'] & t_2 \arrow[r, "\beta_3"'] & \cdots \arrow[r, "\beta_{n-1}"'] & t_{n-1} \arrow[r, "\beta_n"'] & t_n = t
\end{tikzcd}$$

Figure 8.4: Permuting α with the independent actions β_1 through β_n.

Permuting Independent Actions

Lemma 1

Let $\rho : s = s_0 \xrightarrow{\beta_1} s_1 \xrightarrow{\beta_2} s_2 \cdots \xrightarrow{\beta_n} s_n$ be an execution fragment, then for any $\alpha \in En(s)$ s.t. $\forall i \in \{1, \cdots, n\}(\alpha, \beta_i) \in I$ we have

- $\alpha \in En(s_i)$
- $\rho' : s = s_0 \xrightarrow{\alpha} t_0 \xrightarrow{\beta_1} t_1 \cdots \xrightarrow{\beta_n} t_n$ is an execution fragment of the same TS.

Equivalence of executions

When can we say that $\rho \triangleq \rho'$? *Note ρ, ρ' may not be of the same length!*

Stutter Actions

Action α is a stutter if for all $s \in S$ s.t. $\alpha \in En(s), s \xrightarrow{\alpha} s'$ we have $L(s) = L(s')$.

Permuting Independent Actions

Lemma 1

Let $\rho : s = s_0 \xrightarrow{\beta_1} s_1 \xrightarrow{\beta_2} s_2 \cdots \xrightarrow{\beta_n} s_n$ be an execution fragment, then for any $\alpha \in En(s)$ s.t. $\forall i \in \{1, \cdots, n\}(\alpha, \beta_i) \in I$ we have

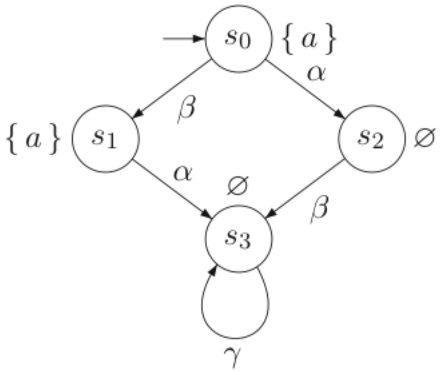
- $\alpha \in En(s_i)$
- $\rho' : s = s_0 \xrightarrow{\alpha} t_0 \xrightarrow{\beta_1} t_1 \cdots \xrightarrow{\beta_n} t_n$ is an execution fragment of the same TS.

Equivalence of executions

When can we say that $\rho \triangleq \rho'$? *Note ρ, ρ' may not be of the same length!*

Stutter Actions

Action α is a stutter if for all $s \in S$ s.t. $\alpha \in En(s), s \xrightarrow{\alpha} s'$ we have $L(s) = L(s')$.



Permuting Independent **Stutter** Actions

Lemma 2

Let ρ, ρ' be finite execution fragments with action sequences $\beta_1 \cdots \beta_n \alpha$ and $\alpha \beta_1 \cdots \beta_n$, respectively s.t. α is stuttering and independent with $\beta_1 \cdots \beta_n$, then $\rho \triangleq \rho'$.

Permuting Independent Stutter Actions

Lemma 2

Let ρ, ρ' be finite execution fragments with action sequences $\beta_1 \cdots \beta_n \alpha$ and $\alpha \beta_1 \cdots \beta_n$, respectively s.t. α is stuttering and independent with $\beta_1 \cdots \beta_n$, then $\rho \triangleq \rho'$.

Permuting Independent Stutter Actions

Lemma 2

Let ρ, ρ' be finite execution fragments with action sequences $\beta_1 \cdots \beta_n \alpha$ and $\alpha \beta_1 \cdots \beta_n$, respectively s.t. α is stuttering and independent with $\beta_1 \cdots \beta_n$, then $\rho \triangleq \rho'$.

Theorem 1

Any LTL_{-X} property is invariant under stuttering

Permuting Independent Stutter Actions
Lemma 2
Let ρ, ρ' be finite execution fragments with action sequences $\beta_1 \cdots \beta_n \alpha$ and $\alpha \beta_1 \cdots \beta_n$, respectively s.t. α is stuttering and independent with $\beta_1 \cdots \beta_n$, then $\rho \triangleq \rho'$.
Theorem 1
Any LTL_{-X} property is invariant under stuttering

1. In fact, since we have not read LTL-X, let us restrict ourselves to safety invariants, i.e., something bad can never happen. Eg. **never** two procs are simultaneously in the critical region: $\Box(\neg cr_1 \vee \neg cr_2)$
2. A property f is invariant under stuttering means: for paths $\pi, \pi', \pi \models f$ iff $\pi' \models f$, where $\pi \sim_{st} \pi'$

Permuting Independent Stutter Actions

Lemma 2

Let ρ, ρ' be finite execution fragments with action sequences $\beta_1 \cdots \beta_n \alpha$ and $\alpha \beta_1 \cdots \beta_n$, respectively s.t. α is stuttering and independent with $\beta_1 \cdots \beta_n$, then $\rho \triangleq \rho'$.

Theorem 1

Any LTL_{-X} property is invariant under stuttering

How do I put all of the above concepts together to use?

Permuting Independent Stutter Actions
Lemma 2
Let ρ, ρ' be finite execution fragments with action sequences $\beta_1 \cdots \beta_n \alpha$ and $\alpha \beta_1 \cdots \beta_n$, respectively s.t. α is stuttering and independent with $\beta_1 \cdots \beta_n$, then $\rho \triangleq \rho'$.
Theorem 1
Any LTL_{-X} property is invariant under stuttering
How do I put all of the above concepts together to use?

1. In fact, since we have not read LTL-X, let us restrict ourselves to safety invariants, i.e., something bad can never happen. Eg. **never** two procs are simultaneously in the critical region: $\Box(\neg cr_1 \vee \neg cr_2)$
2. A property f is invariant under stuttering means: for paths $\pi, \pi', \pi \models f$ iff $\pi' \models f$, where $\pi \sim_{st} \pi'$

Permuting Independent Stutter Actions

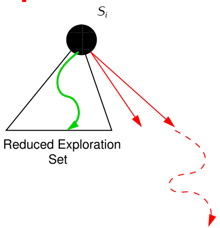
Lemma 2

Let ρ, ρ' be finite execution fragments with action sequences $\beta_1 \cdots \beta_n \alpha$ and $\alpha \beta_1 \cdots \beta_n$, respectively s.t. α is stuttering and independent with $\beta_1 \cdots \beta_n$, then $\rho \triangleq \rho'$.

Theorem 1

Any LTL_{-X} property is invariant under stuttering

How do I put all of the above concepts together to use?



Explore the state graph via **DFS** using **reduced sets**

Permuting Independent Stutter Actions
Lemma 2
Let ρ, ρ' be finite execution fragments with action sequences $\beta_1 \cdots \beta_n \alpha$ and $\alpha \beta_1 \cdots \beta_n$, respectively s.t. α is stuttering and independent with $\beta_1 \cdots \beta_n$, then $\rho \triangleq \rho'$.
Theorem 1
Any LTL_{-X} property is invariant under stuttering
How do I put all of the above concepts together to use?
Explore the state graph via DFS using reduced sets

1. In fact, since we have not read LTL-X, let us restrict ourselves to safety invariants, i.e., something bad can never happen. Eg. **never** two procs are simultaneously in the critical region: $\Box(\neg cr_1 \vee \neg cr_2)$
2. A property f is invariant under stuttering means: for paths $\pi, \pi', \pi \models f$ iff $\pi' \models f$, where $\pi \sim_{st} \pi'$

Reduced Exlporation Sets

Ample sets, Persistent sets, Stubborn sets, Source sets, etc.
Let us understand ample sets

- $ample(s) \subseteq En(s)$

Reduced Exlporation Sets

Ample sets, Persistent sets, Stubborn sets, Source sets, etc.
Let us understand ample sets

- $ample(s) \subseteq En(s)$

Reduced Exlporation Sets

Ample sets, Persistent sets, Stubborn sets, Source sets, etc.
Let us understand ample sets

- $ample(s) \subseteq En(s)$
- $C_0 : ample(s) = \phi$ iff $En(s) = \phi$

Reduced Exlporation Sets

Ample sets, Persistent sets, Stubborn sets, Source sets, etc.
Let us understand ample sets

- $ample(s) \subseteq En(s)$
- $C_0 : ample(s) = \phi$ iff $En(s) = \phi$

1. C_0 guarantees that if s in TS has at least one successor, it is preserved in TS' too

Reduced Exlporation Sets

Ample sets, Persistent sets, Stubborn sets, Source sets, etc.
Let us understand ample sets

- $ample(s) \subseteq En(s)$
- C_0 : $ample(s) = \phi$ iff $En(s) = \phi$
- C_1 : No transition dependent on a transition in $ample(s)$ can occur before *some* a transition in $ample(s)$ occurs.

Reduced Exlporation Sets
Ample sets, Persistent sets, Stubborn sets, Source sets, etc.
Let us understand ample sets
- $ample(s) \subseteq En(s)$
- C_0 : $ample(s) = \phi$ iff $En(s) = \phi$
- C_1 : No transition dependent on a transition in $ample(s)$ can occur before *some* a transition in $ample(s)$ occurs.

1. Computing C1 is difficult w/o constructing the full state graph
2. In fact, it has been shown that the algorithmic complexity to check C1 is as hard as checking a reachability property on the full TS.
3. C1 gurantees that every execution in the quotient TS (TS') is of the form: $s \xrightarrow{\beta_1} s_1 \xrightarrow{\beta_2} \cdots \xrightarrow{\beta_n} s_n \xrightarrow{\alpha} t$ with $\alpha \in ample(s)$ and β_i independent of all transitions in $ample(s)$, $0 < i \leq n$.
4. In fact, this condition also ensures that if s is not fully expanded, then every action in $ample(s)$ is **independent** with $En(s) \setminus ample(s)$: Proof – Let $\gamma \in En(s) \setminus ample(s)$. Suppose $(\gamma, \delta) \in D$ where $\delta \in ample(s)$. Since γ is enabled in s, implies there is a path starting with γ in TS. But this also means that a transition dependent on $ample(s)$ is executed before a transition from $ample(s)$, thus violating C1.

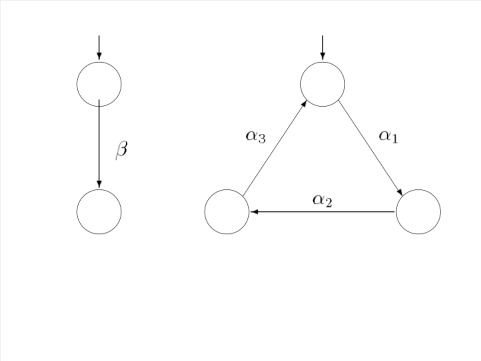
Reduced Exlporation Sets

Ample sets, Persistent sets, Stubborn sets, Source sets, etc.
Let us understand ample sets

- $ample(s) \subseteq En(s)$
- C_0 : $ample(s) = \phi$ iff $En(s) = \phi$
- C_1 : No transition dependent on a transition in $ample(s)$ can occur before *some* a transition in $ample(s)$ occurs.
- C_2 : If $ample(s) \neq En(s)$, then every $\alpha \in ample(s)$ is a stutter action.

Reduced Exlporation Sets
Ample sets, Persistent sets, Stubborn sets, Source sets, etc.
Let us understand ample sets
- $ample(s) \subseteq En(s)$
- C_0 : $ample(s) = \phi$ iff $En(s) = \phi$
- C_1 : No transition dependent on a transition in $ample(s)$ can occur before *some* a transition in $ample(s)$ occurs.
- C_2 : If $ample(s) \neq En(s)$, then every $\alpha \in ample(s)$ is a stutter action.

1. C_2 gurantees that the trace transformations from C_1 are indeed stuttering equivalent execution
2. Note however that the conditions mentioned so far are not sufficient for soundness: $(\beta, \alpha_i) \in I$. Assume β is **not** visible.



Reduced Exlporation Sets

Ample sets, Persistent sets, Stubborn sets, Source sets, etc.
Let us understand ample sets

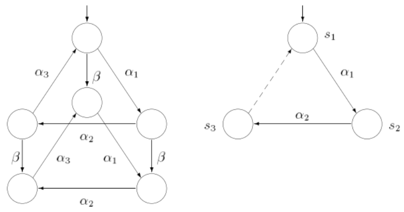
- $ample(s) \subseteq En(s)$
- $C_0 : ample(s) = \phi$ iff $En(s) = \phi$
- C_1 : No transition dependent on a transition in $ample(s)$ can occur before *some* a transition in $ample(s)$ occurs.
- C_2 : If $ample(s) \neq En(s)$, then every $\alpha \in ample(s)$ is a stutter action.
- $C3$: For any cycle $s_0, \cdots, s_n$ in reduced TS where $\alpha \in En(s_i), 0 < i \leq n$, $\exists j \in \{1, \cdots, n\}$ s.t. $\alpha \in ample(s_j)$.

Reduced Exlporation Sets

Ample sets, Persistent sets, Stubborn sets, Source sets, etc.
Let us understand ample sets

- $ample(s) \subseteq En(s)$
- $C_0 : ample(s) = \phi$ iff $En(s) = \phi$
- C_1 : No transition dependent on a transition in $ample(s)$ can occur before *some* a transition in $ample(s)$ occurs.
- C_2 : If $ample(s) \neq En(s)$, then every $\alpha \in ample(s)$ is a stutter action.
- $C3$: For any cycle $s_0, \cdots, s_n$ in reduced TS where $\alpha \in En(s_i), 0 < i \leq n$, $\exists j \in \{1, \cdots, n\}$ s.t. $\alpha \in ample(s_j)$.

1. The idea is that we can delay an enabled action in a cycle indefinitely and not take it.
2. $ample(s_1) = \{\alpha_1\}$, $ample(s_2) = \{\alpha_2\}$, $ample(s_3) = \{\alpha_3\}$. This satisfies the conditions C_1, C_2, C_3 but does not include any sequence where p is changed from false to true.



Reduced Exlporation Sets

Ample sets, Persistent sets, Stubborn sets, Source sets, etc.
Let us understand ample sets

- $ample(s) \subseteq En(s)$
- $C_0 : ample(s) = \phi$ iff $En(s) = \phi$
- C_1 : No transition dependent on a transition in $ample(s)$ can occur before *some* a transition in $ample(s)$ occurs.
- C_2 : If $ample(s) \neq En(s)$, then every $\alpha \in ample(s)$ is a stutter action.
- $C3$: For any cycle $s_0, \cdots, s_n$ in reduced TS where $\alpha \in En(s_i), 0 < i \leq n, \exists j \in \{1, \cdots, n\}$ s.t. $\alpha \in ample(s_j)$.
- **Recent Results (CAV'19, "What's wrong with on-the-fly partial order reduction", Steven Siegel) – C3 is incorrect for certain cases!**

Reduced Exlporation Sets

Ample sets, Persistent sets, Stubborn sets, Source sets, etc.
Let us understand ample sets

- $ample(s) \subseteq En(s)$
- $C_0 : ample(s) = \phi$ iff $En(s) = \phi$
- C_1 : No transition dependent on a transition in $ample(s)$ can occur before *some* a transition in $ample(s)$ occurs.
- C_2 : If $ample(s) \neq En(s)$, then every $\alpha \in ample(s)$ is a stutter action.
- $C3$: For any cycle $s_0, \cdots, s_n$ in reduced TS where $\alpha \in En(s_i), 0 < i \leq n, \exists j \in \{1, \cdots, n\}$ s.t. $\alpha \in ample(s_j)$.
- **Recent Results (CAV'19, "What's wrong with on-the-fly partial order reduction", Steven Siegel) – C3 is incorrect for certain cases!**

Why C_1 is important? The intuitions behind this condition!

Note C_1's statement also means that we see executions only of the form:

- $s \xrightarrow{\beta_1} s_1 \xrightarrow{\beta_2} \cdots \xrightarrow{\beta_n} s_n \xrightarrow{\alpha}$ where $\forall i, \beta_i$ is independent with $ample(s)$ and $\alpha \in ample(s)$.

Why is that so? The high level intuition is:

- among co-enabled actions at a state, put all the actions that are mutually dependent in the ample set of that state.

Why C_1 is important? The intuitions behind this condition!

Note C_1's statement also means that we see executions only of the form:

- $s \xrightarrow{\beta_1} s_1 \xrightarrow{\beta_2} \cdots \xrightarrow{\beta_n} s_n \xrightarrow{\alpha}$ where $\forall i, \beta_i$ is independent with $ample(s)$ and $\alpha \in ample(s)$.

Why is that so? The high level intuition is:

- among co-enabled actions at a state, put all the actions that are mutually dependent in the ample set of that state.

Computing Ample Sets

- Checking and establishing C_3 is straightforward. If C_0 to C_2 hold then $C_3' \Rightarrow C_3$. C_3': Any cycle in TS' contains at least one state with $ample(s) = En(s)$.

Computing Ample Sets

- Checking and establishing C_3 is straightforward. If C_0 to C_2 hold then $C_3' \Rightarrow C_3$. C_3': Any cycle in TS' contains at least one state with $ample(s) = En(s)$.

Computing Ample Sets

- Checking and establishing C_3 is straightforward. If C_0 to C_2 hold then $C_3' \Rightarrow C_3$. C_3': Any cycle in TS' contains at least one state with $ample(s) = En(s)$.
- Local computations. Let $ample(s) = Act_i(s)$. Checking C_0 is straightforward.

Computing Ample Sets

- Checking and establishing C_3 is straightforward. If C_0 to C_2 hold then $C_3' \Rightarrow C_3$. C_3': Any cycle in TS' contains at least one state with $ample(s) = En(s)$.
- Local computations. Let $ample(s) = Act_i(s)$. Checking C_0 is straightforward.

1. See if $Act_i(s)$ is empty.

Computing Ample Sets

- Checking and establishing C_3 is straightforward. If C_0 to C_2 hold then $C_3' \Rightarrow C_3$. C_3': Any cycle in TS' contains at least one state with $ample(s) = En(s)$.
- Local computations. Let $ample(s) = Act_i(s)$. Checking C_0 is straightforward.
- Checking C_2 requires checking for each action in $Act_i(s)$.

Computing Ample Sets

- Checking and establishing C_3 is straightforward. If C_0 to C_2 hold then $C_3' \Rightarrow C_3$. C_3': Any cycle in TS' contains at least one state with $ample(s) = En(s)$.
- Local computations. Let $ample(s) = Act_i(s)$. Checking C_0 is straightforward.
- Checking C_2 requires checking for each action in $Act_i(s)$.

1. Static check whether the actions modify the shared variables (in other words, the A.P. are not refered to in actions of $Act_i(s)$

Computing Ample Sets

- Checking and establishing C_3 is straightforward. If C_0 to C_2 hold then $C_3' \Rightarrow C_3$. C_3': Any cycle in TS' contains at least one state with $ample(s) = En(s)$.
- Local computations. Let $ample(s) = Act_i(s)$. Checking C_0 is straightforward.
- Checking C_2 requires checking for each action in $Act_i(s)$.
- Local criteria to ensure C_1 for the choice $ample(s) = Act_i(s)$
  1. Check if $dep(Act_i(s)$ includes a transition from $P_j, i \neq j$
  2. Any $\beta \in Act_i \setminus Act(s)$ may not become enabled through some process $P_j, j \neq i$.

Computing Ample Sets

- Checking and establishing C_3 is straightforward. If C_0 to C_2 hold then $C_3' \Rightarrow C_3$. C_3': Any cycle in TS' contains at least one state with $ample(s) = En(s)$.
- Local computations. Let $ample(s) = Act_i(s)$. Checking C_0 is straightforward.
- Checking C_2 requires checking for each action in $Act_i(s)$.
- Local criteria to ensure C_1 for the choice $ample(s) = Act_i(s)$
  1. Check if $dep(Act_i(s)$ includes a transition from $P_j, i \neq j$
  2. Any $\beta \in Act_i \setminus Act(s)$ may not become enabled through some process $P_j, j \neq i$.

1. There are 2 cases where this selection might vioate C1
2. In both cases some actions independent with those in $Act_i(s)$ are executed and enable α s.t. α is dependent on $Act_i(s)$.
3. The two conditions in the slide are coming from these two cases. Let us consider the first case: α belong to P_j, implies that all we have to check is whether $dep(Act_i(s))$ contains a transition from P_j.
4. In the second case, α belongs to P_i. Suppose α is executed from s'. The transitions executed from s to s' are independent of $Act_i(s)$, hence from other processes.
5. Since $\alpha \notin Act_i(s)$, it is disabled in s. Thus $pre(\alpha)$ must include actions from processes other than P_i.
6. **both these cases can be effectively checked**.

Understanding DPOR with an Example

T_1	T_2	T_3
$W(x)$	$R(y)$	$R(z)$
	$R(x)$	$R(x)$

2019-07-02
Fundamentals of Partial Order Reduction
Understanding DPOR with an Example

Understanding DPOR with an Example

T_1	T_2	T_3
$W(x)$	$R(y)$	$R(z)$
	$R(x)$	$R(x)$

DPOR - Ample Sets

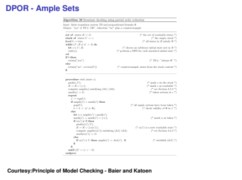
Algorithm 38 Invariant checking using partial order reduction

Input: finite transition system TS and propositional formula Φ
Output: "yes" if $TS \models \Box\Phi$", otherwise "no" plus a counterexample

```
set of states R := ∅;                                    (* the set of reachable states *)
stack of states U := ε;                                   (* the empty stack *)
bool b := true;                                           (* all states in R satisfy Φ *)
while (I \ R ≠ ∅ ∧ b) do
  let s ∈ I \ R;                     (* choose an arbitrary initial state not in R *)
  visit(s);                          (* perform a DFS for each unvisited initial state *)
od
if b then
  return("yes")                                           (* TS ⊨ "always Φ" *)
else
  return("no", reverse(U))           (* counterexample arises from the stack content *)
fi

procedure visit (state s)
  push(s, U);                                             (* push s on the stack *)
  R := R ∪ { s };                                         (* mark s as reachable *)
  compute ample(s) satisfying (A1)–(A3);                  (* see Section 8.2.3 *)
  mark(s) := ∅;                                           (* taken actions in s *)
  repeat
    s' := top(U);
    if ample(s') = mark(s') then
      pop(U);                                   (* all ample actions have been taken *)
      b := b ∧ (s' ⊨ Φ);                        (* check validity of Φ in s' *)
    else
      let α ∈ ample(s') \ mark(s');
      mark(s') := mark(s') ∪ { α };                       (* mark α as taken *)
      if α(s') ∉ R then
        push(α(s'), U);
        R := R ∪ { α(s') };                     (* α(s') is a new reachable state *)
        compute ample(α(s')) satisfying (A1)–(A3);        (* see Section 8.2.3 *)
        mark(α(s')) := ∅;
      else
        if α(s') ∈ U then ample(s') := Act(s'); fi        (* establish (A4') *)
      fi
    fi
  until ((U = ε) ∨ ¬b)
endproc
```

Courtesy:Principle of Model Checking - Baier and Katoen



1. Work out on an example.

Reduced Exploration Sets – Persistent Sets

Persistent Sets

A set T of transitions enabled in a state s is *persistent* in s iff for all nonempty sequence of transitions:

$$s = s_1 \xrightarrow{t_1} s_2 \xrightarrow{t_2} s_3, \cdots, \xrightarrow{t_n} s_{n+1}$$

in A_G and including only transitions $t_i \notin T$, $1 \leq i \leq n$, t_n is independent with all the transition in T.

Reduced Exploration Sets – Persistent Sets

Persistent Sets
A set T of transitions enabled in a state s is *persistent* in s iff for all nonempty sequence of transitions:

$$s = s_1 \xrightarrow{t_1} s_2 \xrightarrow{t_2} s_3, \cdots, \xrightarrow{t_n} s_{n+1}$$

in A_G and including only transitions $t_i \notin T$, $1 \leq i \leq n$, t_n is independent with all the transition in T.

Reduced Exploration Sets – Persistent Sets

Lemma – Non-empty Persistent_Set

Let s be a state in TS', and let d be a terminal reachable from s in TS by a nonempty sequence w of transitions. For all $w_i \in [w]_s$, let t_i denote the first transition of w_i. Let Persistent_Set(s) be a nonempty persistent set in s. Then, at least one of the transitions t_i is in Persistent_Set(s).

Reduced Exploration Sets – Persistent Sets

Lemma – Non-empty Persistent_Set

Let s be a state in TS', and let d be a terminal reachable from s in TS by a nonempty sequence w of transitions. For all $w_i \in [w]_s$, let t_i denote the first transition of w_i. Let Persistent_Set(s) be a nonempty persistent set in s. Then, at least one of the transitions t_i is in Persistent_Set(s).

1. Let w be the sequence $s = s_1 \xrightarrow{t_1} s_2 \xrightarrow{t_2} s_3 \cdots s_n \xrightarrow{t_n} d$. Assume that **none** of w transitions are in Persistent_Set(s).
2. Following the definition, $\forall j : t_j$ is independent in s_j with all the transitions in Persistent_Set(s). Following the definition of Independence relation, entire Persistent_Set(s) is enable in s_j, implying d can not be the terminal state. Thus, some transition of the sequence w from s to d must be in Persistent_Set(s).
3. Let t_k be that first transition in w to be in the Persistent_Set(s). Let $w' = t_k t_1 \cdots t_{k-1} t_{k+1} \cdots t_n$
4. By the definition of Persistent sets $\forall 1 \leq j < k, t_j$ is independent with t_k in s_j.
5. Consequently, by definition of a trace $w' \in [w]_s$ and the lemma is proved.

Reduced Exploration Sets – Persistent Sets

Theorem: Soundness

Let s be a state in TR', and let d be a deadlock reachable from s in TS by a sequence w of transitions. Then, d is also reachable from s in TS'.

Reduced Exploration Sets – Persistent Sets

Theorem: Soundness

Let s be a state in TR', and let d be a deadlock reachable from s in TS by a sequence w of transitions. Then, d is also reachable from s in TS'.

1. Proof is by induction on the length of w. For $|w| = 0$, the result is immediate. Assume the result holds for $n \geq 0$, we will try to prove for paths of length $n+1$.
2. Assume in TS there exists a path of length $n+1$ from state s to d. Let t_i be the first transition of $w_i \in [w]_s$ such that at least one of the t_i is in Persistent_Set(s) (from previous lemma). Which means along with inductive hypothesis we no have a deadlock reachable in TS' of path $n+1$.

Source Sets

Source Sets

Let E be an execution sequence, and let W be a set of sequences, such that $E.w$ is an execution sequence for each $w \in W$. A set of processes is a source set for W after E if for each $w \in W$ we have $WI_{[E]}(w) \cap P \neq \emptyset$

Source Sets

Source Sets

Let E be an execution sequence, and let W be a set of sequences, such that $E.w$ is an execution sequence for each $w \in W$. A set of processes is a source set for W after E if for each $w \in W$ we have $WI_{[E]}(w) \cap P \neq \emptyset$

DPOR - Persistent Sets $\boxed{\textbf{POPL'2005}}$

- I: Enabledness:if $t_1 \in En(s)$ and $s \xrightarrow{t_1} s'$,then $t_2 \in En(s)$ iff $t_2 \in En(s')$

DPOR - Persistent Sets POPL'2005

- I: Enabledness:if $t_1 \in En(s)$ and $s \xrightarrow{t_1} s'$,then $t_2 \in En(s)$ iff $t_2 \in En(s')$

DPOR - Persistent Sets $\boxed{\textbf{POPL'2005}}$

- I: Enabledness:if $t_1 \in En(s)$ and $s \xrightarrow{t_1} s'$,then $t_2 \in En(s)$ iff $t_2 \in En(s')$
- **Happens-before** relation $\longrightarrow_E$ (for an execution sequence E):

.

DPOR - Persistent Sets POPL'2005

- I: Enabledness:if $t_1 \in En(s)$ and $s \xrightarrow{t_1} s'$,then $t_2 \in En(s)$ iff $t_2 \in En(s')$
- **Happens-before** relation $\longrightarrow_E$ (for an execution sequence E):

.

HB is the smallest relation on $\{1, 2, ..., n\}$

DPOR - Persistent Sets $\boxed{\textbf{POPL'2005}}$

- I: Enabledness:if $t_1 \in En(s)$ and $s \xrightarrow{t_1} s'$,then $t_2 \in En(s)$ iff $t_2 \in En(s')$
- **Happens-before** relation $\longrightarrow_E$ (for an execution sequence E):
    1. if $i \leq j$ and $(E_i, E_j) \in D$ then $i \longrightarrow_E j$

.

DPOR - Persistent Sets POPL'2005
I: Enabledness:if $t_1 \in En(s)$ and $s \xrightarrow{t_1} s'$,then $t_2 \in En(s)$ iff $t_2 \in En(s')$
Happens-before relation $\longrightarrow_E$ (for an execution sequence E):
if $i \leq j$ and $(E_i, E_j) \in D$ then $i \longrightarrow_E j$
.

1. E_i: is the transition t_i

DPOR - Persistent Sets $\boxed{\textbf{POPL'2005}}$

- I: Enabledness:if $t_1 \in En(s)$ and $s \xrightarrow{t_1} s'$,then $t_2 \in En(s)$ iff $t_2 \in En(s')$
- **Happens-before** relation $\longrightarrow_E$ (for an execution sequence E):
  1. if $i \leq j$ and $(E_i, E_j) \in D$ then $i \longrightarrow_E j$
  2. $\longrightarrow_E$ is transitively closed.

  .

DPOR - Persistent Sets POPL'2005
I: Enabledness:if $t_1 \in En(s)$ and $s \xrightarrow{t_1} s'$,then $t_2 \in En(s)$ iff $t_2 \in En(s')$
Happens-before relation $\longrightarrow_E$ (for an execution sequence E):
if $i \leq j$ and $(E_i, E_j) \in D$ then $i \longrightarrow_E j$
$\longrightarrow_E$ is transitively closed.

1. E_i: is the transition t_i1. By construction, HB is partial order. Called **Mazurkiewicz trace**

DPOR - Persistent Sets $\boxed{\textbf{POPL'2005}}$

- *I*: Enabledness:if $t_1 \in En(s)$ and $s \xrightarrow{t_1} s'$,then $t_2 \in En(s)$ iff $t_2 \in En(s')$
- **Happens-before** relation $\longrightarrow_E$ (for an execution sequence E):
  1. if $i \leq j$ and $(E_i, E_j) \in D$ then $i \longrightarrow_E j$
  2. $\longrightarrow_E$ is transitively closed.

  .
- HB variant: $i \longrightarrow_E p$ (where p is a process, $i \in Dom(E)$) if either

DPOR - Persistent Sets POPL'2005

- *I*: Enabledness:if $t_1 \in En(s)$ and $s \xrightarrow{t_1} s'$,then $t_2 \in En(s)$ iff $t_2 \in En(s')$
- **Happens-before** relation $\longrightarrow_E$ (for an execution sequence E):
  1. if $i \leq j$ and $(E_i, E_j) \in D$ then $i \longrightarrow_E j$
  2. $\longrightarrow_E$ is transitively closed.

  .
- HB variant: $i \longrightarrow_E p$ (where p is a process, $i \in Dom(E)$) if either

DPOR - Persistent Sets $\boxed{\textbf{POPL'2005}}$

- I: Enabledness:if $t_1 \in En(s)$ and $s \xrightarrow{t_1} s'$,then $t_2 \in En(s)$ iff $t_2 \in En(s')$
- **Happens-before** relation $\longrightarrow_E$ (for an execution sequence E):
  1. if $i \leq j$ and $(E_i, E_j) \in D$ then $i \longrightarrow_E j$
  2. $\longrightarrow_E$ is transitively closed.

  .
- HB variant: $i \longrightarrow_E p$ (where p is a process, $i \in Dom(E)$) if either
  1. $proc(E_i) = p$ or

DPOR - Persistent Sets POPL'2005

- I: Enabledness:if $t_1 \in En(s)$ and $s \xrightarrow{t_1} s'$,then $t_2 \in En(s)$ iff $t_2 \in En(s')$
- **Happens-before** relation $\longrightarrow_E$ (for an execution sequence E):
  1. if $i \leq j$ and $(E_i, E_j) \in D$ then $i \longrightarrow_E j$
  2. $\longrightarrow_E$ is transitively closed.

  .
- HB variant: $i \longrightarrow_E p$ (where p is a process, $i \in Dom(E)$) if either
  1. $proc(E_i) = p$ or

DPOR - Persistent Sets $\boxed{\textbf{POPL'2005}}$

- *I*: Enabledness:if $t_1 \in En(s)$ and $s \xrightarrow{t_1} s'$,then $t_2 \in En(s)$ iff $t_2 \in En(s')$
- **Happens-before** relation $\longrightarrow_E$ (for an execution sequence E):
  1. if $i \leq j$ and $(E_i, E_j) \in D$ then $i \longrightarrow_E j$
  2. $\longrightarrow_E$ is transitively closed.

  .
- HB variant: $i \longrightarrow_E p$ (where p is a process, $i \in Dom(E)$) if either
  1. $proc(E_i) = p$ or
  2. $\exists k \in \{i+1, \cdots, n\}$ s.t. $i \longrightarrow_E k$ and $proc(E_k) = p$

Work out the relations for $p1 : x = 1; x = 2; || p2 : y = 1; x = 3;$

DPOR - Algorithm

```
Initially: Explore(∅);

Explore(S) {
    let s = last(S);
    for all processes p {
        if ∃i = max({i ∈ dom(S) | S_i is dependent and may be co-enabled with next(s,p) and i ↛_s p}) {
            let E = {q ∈ enabled(pre(S,i)) | q = p or ∃j ∈ dom(S) : j > i and q = proc(S_j) and j →_s p};
            if (E ≠ ∅) then add any q ∈ E to backtrack(pre(S,i));
                else add all q ∈ enabled(pre(S,i)) to backtrack(pre(S,i));
        }
    }
    if (∃p ∈ enabled(s)) {
        backtrack(s) := {p};
        let done = ∅;
        while (∃ p ∈ (backtrack(s) \ done)) {
            add p to done;
            Explore(S.next(s,p));
        }
    }
}
```


DPOR - Algorithm

References

1. Principles of Model Checking (Chp 8), Christel Baier and Joost-Pieter Katoen.
2. Using Partial Orders for the Efficient Verification of Deadlock Freedom and Safety Properties. Patrice Godefroid, Pierre Wolper:CAV 1991
3. Partial-Order Methods for the Verification of Concurrent Systems - An Approach to the State-Explosion Problem. Patrice Godefroid:Lecture Notes in Computer Science, 1996.
4. All from One, One for All: Model Checking Using Representatives. Doron Peled: CAV 1993.

References

1. Principles of Model Checking (Chp 8), Christel Baier and Joost-Pieter Katoen.
2. Using Partial Orders for the Efficient Verification of Deadlock Freedom and Safety Properties. Patrice Godefroid, Pierre Wolper:CAV 1991
3. Partial-Order Methods for the Verification of Concurrent Systems - An Approach to the State-Explosion Problem. Patrice Godefroid:Lecture Notes in Computer Science, 1996.
4. All from One, One for All: Model Checking Using Representatives. Doron Peled: CAV 1993.