Directed Model Checking
– Partial Order Reduction –

Stefan Edelkamp
1 Overview

• Motivation
• Independence and Invisibility
• Stuttering Equivalence
• Ample Set
• General State Expanding Search
• Hierarchy of Cycle Conditions
• (Recovering) Solution Quality
2 Motivation

Idea: exploit the commutativity of asynchronous systems to reduce the size of the state space

Resulting state space: equivalent to the original one wrt. specification

Two main families:

- net unfoldings
- diamond properties (here)

Several approaches:

- “stubborn” sets, “persistent” sets, and
- “ample” sets (here)
Independence and Invisibility

\( \alpha, \beta \in T \) independent: for each state \( s \in S \) in which \( \alpha, \beta \) are defined we have

1. \( \alpha \in \text{enabled}(\beta(s)) \) and \( \beta \in \text{enabled}(\alpha(s)) \) (enableness preserving)
2. \( \alpha(\beta(s)) = \beta(\alpha(s)) \) (commutative)

\( \alpha \) invisible wrt. a set of propositions \( P \): \( \forall s, s', s' = \alpha(s): L(s) \cap P = L(s') \cap P \).

Example: Transitions \( \alpha, \beta \) and \( \gamma \) are pairwise independent; \( \alpha \) and \( \beta \) are invisible

![Diagram]

with respect to the set of propositions \( P = \{p\} \), while \( \gamma \) is not
Stuttering Equivalence

Two executions are *stuttering equivalent* wrt. $P$: atomic propositions of the $i$-th block of both executions have the same intersection with $P$, for each $i > 0$

Example: stuttering equivalent paths wrt. LTL property in which only $p$ and $q$ occur.

Two transition systems *stuttering equivalent*: same set of initial states and for each execution in one systems $\exists$ stuttering equivalent execution in the other one

LTL$_X$ cannot distinguish between stuttering equivalent transition systems

$\mathcal{M}$ and $\mathcal{N}$ are two stuttering equivalent transition systems $\Rightarrow \mathcal{M}$ satisfies a given LTL$_X$ specification $\iff \mathcal{N}$ does

Motivation
Ample Set

\textbf{C0}: \( \text{ample}(s) \) is empty exactly when \( \text{enabled}(s) \) is empty.

\textbf{C1}: Along every path in full state space starting at \( s \), a transition dependent on one in \( \text{ample}(s) \) does not occur without a transition in \( \text{ample}(s) \) occurring first.

\textbf{C2}: If a state \( s \) is not fully expanded, then each transition \( \alpha \) in the ample set of \( s \) must be invisible with regard to \( P \).

\textbf{C3}: If for each state of a cycle in the reduced state space, a transition \( \alpha \) is enabled, then \( \alpha \) must be in the ample set of some of the states of the cycle.

\textbf{C0} and \textbf{C2}: independent to search algorithm applied

Checking Condition \textbf{C1} at least as hard as checking reachability for the full state space \( \Rightarrow \) over-approximation
Dynamically Checking the Cycle Condition

**Condition C3\textsubscript{cycle}:** Every cycle in the reduced state space contains at least one state that is fully expanded.

⇒ checking C3 can be reduced to detecting cycles during the search.

Avoiding ample sets containing backward edges except when state is fully expanded ensures C3 when using DFS/IDA*

⇒ C3\textsubscript{stack}:

C3\textsubscript{stack}: If a state s is not fully expanded, then no transition in ample(s) leads to a state on the search stack.
Safety Cycle Condition

**C3⁻**: If for each state of a cycle in the reduced state space, \( \alpha \) is enabled, then \( \alpha \) must be in the ample set of some successor of a state of the cycle.

**Condition C3⁻ \( \text{stack} \)**: If a state \( s \) is not fully expanded, then at least one transition in \( \text{ample}(s) \) does not lead to a state on the search stack.

**Example**: Contrary to \( \text{C3}_{\text{stack}} \), \( \text{C3⁻}_{\text{stack}} \) accepts \( \{\alpha_1, \alpha_2\} \) as valid ample set.

\[ \Rightarrow \text{C3⁻}_{\text{stack}} \text{ not sufficient to guarantee C3 (necessary for checking liveness)} \]
3 Checking Cycle Condition with GSEA

*Common:* assumes cycle to exist whenever an already visited state is found.

⇒ weaker reductions, as it is known that state spaces of concurrent systems usually have a high density of duplicate states.

**C3\textsubscript{duplicate}:** If a state \(s\) is not fully expanded, then no transition in \(ample(s)\) leads to an already visited state.

\[\{\alpha_1\} \text{ and } \{\alpha_1, \alpha_2\} \text{ are examples of non valid ample sets. On the other hand, the set } \{\alpha_2\} \text{ is not refuted.}\]
Safety Cycle Condition for a GSEA

... alternative condition in order to enforce the cycle condition $C3^-$

... sufficient to guarantee a correct reduction when checking safety properties.

... based on the same idea as $C3_{\text{duplicate}}$

**Condition $C3^-_{\text{duplicate}}$:** If a state $s$ is not fully expanded, then at least one transition in $ample(s)$ does not lead to an already visited state.

Example: $\{\alpha_1, \alpha_2\}$ is rejected as ample set by condition $C3_{\text{duplicate}}$, but not by $C3^-_{\text{duplicate}}$. 
4 Hierarchy of Cycle Conditions

*C3 Conditions*: All presented cycle conditions for checking safety properties.

Arrows indicate which condition enforces which other.
5 Solution Quality

*Observation:* Shortest path to error in reduced space often longer than shortest path to error in full space

*Intuition:* Concept of stuttering equivalence does not make assumptions about length of equivalent blocks.

*Example:* $\alpha$ and $\beta$ independent, $\alpha$ invisible wrt. $p$; invariant $Gp$ to be checked

$\Rightarrow$ reduced state space stuttering equivalent to full one, shortest path length of 2 vs. 1 for original space
Recovering Solution Quality

Idea: process error trail after the verification; ignore those transitions that are independent from the one that directly lead to the error state

Approach: extracting irrelevant transitions from the counterexamples until there is no such transition

Observation: after extracting an irrelevant transition, more new transitions may become irrelevant, e.g. if they were dependent on the recently extracted transition.

⇒ an efficient algorithm must extract transitions beginning from the last one.
Procedure $Filter(r)$

$r' \leftarrow r; j \leftarrow n - 1$;

while $1 \leq j < n$ do
    irrelevant $\leftarrow$ true;
    if $visible(r'[j])$ then irrelevant $\leftarrow$ false;
    else
        for $i$ in $j + 1 .. n$ do
            if $dependent(r'[i], r'[j])$ or $can_enable(r'[j], r'[i])$ then
                irrelevant $\leftarrow$ false; break;
            if irrelevant then $r' \leftarrow extract(r', j); n \leftarrow n - 1;$
            else $j \leftarrow j - 1$;
    return $r'$;
Admissability

Observation: resulting counterexample may not be optimal.

Error: $p$ doesn’t hold; Independent: $(\alpha_3, \alpha_4), (\alpha_6, \alpha_7), (\alpha_6, \alpha_8)$; Visible: $\alpha_6, \alpha_4$

Path formed by transitions $\alpha_1, \alpha_2, \alpha_3$ and $\alpha_4$ can be established as shortest path in the reduced state space denoted by the dashed region.

Established error path $\alpha_1\alpha_2\alpha_4$; Optimal error path in full space: $\alpha_5\alpha_6$. 

Pseudo-Code
References


