Symbolic Model Checking: Introduction

- Symbolic model checking tries to alleviate the state space explosion problem by using efficient encodings of the state space.
- The state space is encoded using an implicit representation.
- For model checking we then need a symbolic representation of the transition relation and the temporal operators.
- One symbolic encoding is using boolean functions (propositional logic).
Symbolic Model Checking: Binary Decision Diagrams

Any formula can be converted to using only \texttt{Ite} with the Shannon expansion:

\[ \varphi \leftrightarrow \texttt{Ite}(v, \varphi\{v := \top\}, \varphi\{v := \bot\}) \]

**Example.** Let \[ \varphi = (v_1 \land v_2) \rightarrow v_3 \]. We expand the variables in descending index order. Then the corresponding \textit{INF} is:

\[
\begin{align*}
\varphi &= \text{Ite}(v_3, \varphi_1, \varphi_0) \\
\varphi_1 &= \text{Ite}(v_2, \varphi_{11}, \top) \\
\varphi_0 &= \text{Ite}(v_2, \varphi_{01}, \top) \\
\varphi_{11} &= \text{Ite}(v_1, \top, \top) \\
\varphi_{01} &= \text{Ite}(v_1, \bot, \top)
\end{align*}
\]

The expression can be visualized as an expression tree called a \textit{decision tree}.

Symbolic Model Checking: Binary Decision Diagrams

Finding the shortest formula representing a given set is co-NP-hard. Therefore we need efficient methods for manipulating the formulae, which can become very large. \textit{Binary Decision Diagrams} provide such methods.

We define the three-place connective \texttt{Ite}(\varphi, \psi_T, \psi_F) (‘if-then-else’) in the following way:

\[ \text{Ite}(\varphi, \psi_T, \psi_F) \overset{\text{def}}{=} (\varphi \land \psi_T) \lor (\neg \varphi \land \psi_F) \]

Any propositional formula can be expressed using \texttt{Ite} and the constants \top, \bot as \[ \varphi \rightarrow \psi \leftrightarrow \text{Ite}(\varphi, \psi, \top) \].

Symbolic Model Checking: Binary Decision Diagrams

A binary decision diagram (BDD) is a rooted, directed acyclic graph which has the following characteristics.

- There are one or two terminal nodes with zero outdegree labeled 0 and 1.
- Each variable node \( u \) has two outgoing edges \( \text{low}(u) \) and \( \text{high}(u) \).
- Each variable node \( u \) is associated with a variable \( \text{var}(u) \).
- All paths in the graph respect the given linear ordering \( x_1 < x_2 < \cdots < x_n \).
Symbolic Model Checking: Binary Decision Diagrams

Theorem (Canonicity). For any function \( f : \mathbb{B}^n \rightarrow \mathbb{B} \) there is exactly one BDD \( u \) with variable ordering \( v_1 < v_2 < \cdots < v_n \) such that \( f^u = f(v_1, v_2, \ldots, v_n) \).

Proof: (sketch). The proof proceeds by induction on the number of arguments of \( f \). For \( n = 0 \) the two possible boolean functions are true and false. Each of these have a unique BDD representation \( \top \) and \( \bot \). Since redundant tests are always removed, a BDD with a variable node must be non-constant. Let \( f(v_1, \ldots, v_n, v_{n+1}) \) be a function of \( n + 1 \) arguments. Define \( f_i(x_2, \ldots, x_{n+1}) = f(i, v_2, \ldots, v_{n+1}) \), \( i \in \mathbb{B} \). By the induction hypothesis both \( f_0 \) and \( f_1 \) have unique BDD representations \( u_0 \) and \( u_1 \) such that \( f^{u_0} = f_0 \) and \( f^{u_1} = f_1 \). By Shannon’s expansion we have that:

\[
 f(v_1, v_2, \ldots, v_{n+1}) = \text{Ite}(v_1, f_1, f_2).
\]

A simple case analysis (\( u_0 = u_1 \) and \( u_0 \neq u_1 \)) shows that this resultant BDD is unique.

Symbolic Model Checking: Binary Decision Diagrams

- We identify a BDD by its root node \( u \).
- The true branch of a node is denoted \( \text{high}(u) \) and the false branch is denoted \( \text{low}(u) \).

A BDD \( \varphi^n \) defines a boolean function in the following way.

\[
 \varphi^0 = 0 \\
 \varphi^1 = 1 \\
 \varphi^u = \text{Ite}(\text{var}(u), \varphi^{\text{high}(u)}, \varphi^{\text{low}(u)}), u \text{ is a variable node.}
\]

Binary Decision Diagrams: Algorithms and Implementation

- The set of BDD nodes is implemented as a hash table.
- Let \( \delta = \text{Ite}(v, \delta_1, \delta_2) \), then the hash table maps triples \((v, \delta_1, \delta_2)\) to \( \delta \).
- Each BDD is identified by its variable and two children. A reduced BDD can now be created by recursively performing the Shannon expansion on the formula.
**Binary Decision Diagrams: Algorithms and Implementation**

**Function BDDImp**:Given Bdd \( \varphi \), Bdd \( \psi \):

- If \( \varphi = 0 \) or \( \psi = 1 \) return 1;
- Else if \( \varphi = 1 \) return \( \psi \);
- Else if \( \psi = 0 \) and \( (\varphi, i, \varphi_1, \varphi_2) \in \text{table}_\varphi \)
  - then return new node \((i, \text{BDDImp}(\varphi_1, 0), \text{BDDImp}(\varphi_2, 0))\);
- Else /* \((\varphi, i, \varphi_1, \varphi_2) \) and \((\psi, j, \psi_1, \psi_2)\)*/
  - if \( i = j \) then
    - return new node \((i, \text{BDDImp}(\varphi_1, \psi_1), \text{BDDImp}(\varphi_2, \psi_2))\);
  - Else if \( i < j \) then
    - return new node \((i, \text{BDDImp}(\varphi_1, \psi_1), \text{BDDImp}(\varphi_2, \psi_2))\);
  - Else if \( i > j \) then
    - return new node \((i, \text{BDDImp}(\varphi_1, \psi_1), \text{BDDImp}(\varphi_2, \psi_2))\);

**Function new_node**:Given BddVar \( i \), Bdd \( \delta_1 \), Bdd \( \delta_2 \):

- If \( \delta_1 = \delta_2 \) then return \( \delta_1 \);
- Else if \( \exists \delta: (\delta, i, \delta_1, \delta_2) \in \text{table} \) then return \( \delta \);
- Else \( \max := \max + 1; \text{table} := \text{table} \cup \{\{\max, i, \delta_1, \delta_2\}\} \)
  - return \( \max \);

**Symbolic Model Checking for CTL**

- We describe algorithms for computing a BDD representation \( \varphi^F \) of the set states where the formula \( \varphi \) holds.
- The system is described by variables \( \vec{v} = \{v_1, \ldots, v_n\} \). The transition relation \( R \) is over the variables \( \{v_1, \ldots, v_n, v'_1, \ldots, v'_n\} \).
- For each \( p \in P \) a BDD is given which represents the set \( \mathcal{I}(p) \).
- Computing the BDDs for the propositional case is easy. We simply use the algorithms described the in the previous section.
- How do we compute \( \text{A}(\varphi \text{U} \psi) \) and \( \text{E}(\varphi \text{U} \psi) \)?

**Binary Decision Diagrams: Algorithms and Implementation**

- The size of the constructed BDD can greatly depend on the ordering of the variables. Example \((v_1 \leftarrow v_3) \land (v_2 \leftarrow v_4)\).
- A good ordering can result in a BDD linear w.r.t. the number of variables while a bad ordering may result in an exponential BDD.
- Finding the optimal ordering is an NP-hard problem.
- There provably exists boolean expression which always result in an exponential BDD, irrespectively of the variable ordering.

**Function BDDExists**:Given Bdd \( \varphi \), Vars \( \vec{v} \):

- If \( \varphi \in \{0, 1\} \) then return \( \varphi \);
- Else /* \((\varphi, i, \varphi_1, \varphi_2) \in \text{table}*/
  - \( \delta_1 = \text{BDDExists}(\varphi_1, \vec{v}) \);
  - \( \delta_2 = \text{BDDExists}(\varphi_2, \vec{v}) \);
  - If \( i \notin \vec{v} \) then return \( \text{BDDApply}(\vee, \delta_1, \delta_2) \);
  - Else return new node \((i, \delta_1, \delta_2)\);
Symbolic Model Checking for CTL

Essentially, we only convert the previously presented algorithm to symbolic terms.

\[ E(\psi_2 U^+ \psi_1) : \text{We must compute the least fixpoint of the set} \]
\[ \{ w \mid \exists w'(w < w' \land (w' \in (\psi_1^E \cup \psi_2^E \land E)) \} \] where \( E \) is an intermediate result of the iteration.

\[ A(\psi_2 U^+ \psi_1) : \text{The greatest least of the set} \]
\[ \{ w \mid \forall w'(w < w' \rightarrow (w' \in (\psi_1^E \cup \psi_2^E \land E)) \} \] must be computed.

For \( E(\psi_2 U^+ \psi_1) \):
\[ E_0(w') = \emptyset \]
\[ E_{i+1}(w) = E_i(w) \lor \exists w'(R(w, w') \land (\psi_1^F(w') \lor \psi_2^F(w') \land E_i(w'))) \] where \( \psi_1^F, \psi_2^F \) and \( E \) are BDDs.

Fixpoint calculations with BDDs are easy, as equality checking is a constant time operation.

Relational \( \mu \)-calculus: Preliminaries

- A collection of disjoint sets with a collection of relations over the sets is called a (typed) structure.
- A pair \( \Sigma = (D, R) \) is called a signature, where \( D \) is a finite set of domain names and \( R \) is a set of relation symbols.
- Each relation symbol has an associated type \( \tau \).
- An interpretation \( I \) assign a structure \( S \) to signature \( \Sigma \). Formally, \( I : \Sigma \rightarrow S \).
- For relation a \( R \) with type \( \tau(R) = (D_1, \ldots, D_n) \) the interpretation is \( I(R) \subseteq I(D_1) \times \cdots \times I(D_n) \).

Relational \( \mu \)-calculus: Syntax

Assume that the symbols \( (, \), \( \perp, \rightarrow, =, \exists, \mu, \lambda \) are not in the signature. A well-formed formula has the following syntax:

- \( \perp, (\varphi \rightarrow \psi) \), where \( \varphi \) and \( \psi \) are well-formed formulas,
- \( x_1 = x_2 \), where \( x_1 \) and \( x_2 \) are individual variables of the same type,
- \( \exists x \varphi \), where \( \varphi \) is a well-formed formula, and \( x \) is an individual variable, or
- \( \rho \, x_1 \ldots x_n \), where \( \rho \) is a relation term of type \( (D_1, \ldots, D_n) \), and \( x_i \) is an individual variable of type \( D_i \).

Relational \( \mu \)-calculus: Introduction

- The relational \( \mu \)-calculus is rich logical language. It can be seen as a first order predicate logic with a recursion operator.
- The symbolic techniques presented previously can also be extended for model checking this expressive logic.
**Relational μ-calculus: Models and Semantics**

A relation model $M = (S, I, v)$ for a signature $\Sigma$ consists of a structure $S$, an interpretation $I$ and variable valuation $v$. The semantics are as follows:

- $x^M = v(x)$, if $x$ is an individual variable,
- $\bot^M = \text{false}$,
- $(\varphi \rightarrow \psi)^M = \text{true}$ iff $\varphi^M = \text{false}$ or $\psi^M = \text{true}$,
- $(x_1 = x_2)^M = \text{true}$ iff $x_1^M = x_2^M$,
- $(\exists x \varphi)^M = \text{true}$ iff $\varphi(S, I, v') = \text{true}$ and $v'$ differs from $v$ at most in $x$.

**Relational μ-calculus: Expressivity**

- The expressive power of the relational μ-calculus is between first-order logic and second order logic.
- With the μ-recursion operator all recursive functions of arithmetic can be defined. This means that on infinite domains the relational μ-calculus has the expressive power of Turing machines.
- The addition-relation on natural numbers can be defined in the following way.

Let $Z$ be the constant zero and $S$ the successor relation. The addition-relation is defined by

$$\mu X(\lambda xyz(Zx \land y = z \lor \exists uv(Sux \land Svz \land Xuyv)))$$

**Relational μ-calculus: Syntax**

The relation terms have their own syntax. Very complex relation can be formed using λ-abstraction or μ-recursion. A relation term of type $(D_1, \ldots, D_n)$ is

- a relation symbol $R$ or a relation variable $X$ of type $(D_1, \ldots, D_n)$,
- $\lambda x_1 \ldots x_n \varphi$, where $\varphi$ is a well-formed formula and each $x_i$ is an individual variable of type $D_i$, or
- $\mu X \rho$, where $X$ is a relation variable of type $(D_1, \ldots, D_n)$, and $\rho$ a relation term which is positive in $X$.

A relational term $\rho$ is positive in $X$ if every occurrence of $X$ is under an even number of negation signs.

**Relational μ-calculus: Models and Semantics**

- $(\rho x_1 \ldots x_n)^M = \text{true}$ iff $(x_1^M, \ldots, x_n^M) \in \rho^M$.
- $R^M = I(R)$, if $R$ is a relation symbol, i.e. the name is connected to the preselected interpretation,
- $X^M = v(X)$, if $X$ is a relation variable,
- $(\lambda x_1 \ldots x_n \varphi)^M = \{(d_1, \ldots, d_n) \mid \varphi(S, I, v') = \text{true} \}$ where $v'$ differs from $v$ only in the assignment of $d_i$ to $x_i$, i.e. $(\lambda x_1 \ldots x_n \varphi)^M$ is the relation consisting of all tuples of objects for which $\varphi$ is true, and
- $(\mu X \rho)^M = \cap \{Q \mid \rho^F(Q) \subseteq Q\}$, where $\rho^F(Q) = \rho(S, I, v')$, and $v'$ differs from $v$ only in $v'(X) = Q$. $\mu X \rho$ is the least fixpoint of the functional $\rho^F$. 

- $(\exists x \varphi)^M = \text{true}$ iff $\varphi(S, I, v') = \text{true}$ and $v'$ differs from $v$ at most in $x$.
Relational $\mu$-calculus: Model checking

- A term or a formula with free individual variables $x_1, \ldots, x_m$ is represented as a BDD and BDD variables $x_1, \ldots, x_m$.

- As the variables can appear as successors in the BDDs, substitution is simple matter replacing the variable with a relation.

- The algorithm recursively evaluates the given formula with a case analysis.

Relational $\mu$-calculus: Model checking

Problem: given a relational frame $\mathcal{F} = (S, I)$ and a relational term $\rho$ or a formula $\varphi$, what is the denotation of $\rho^\mathcal{F}$ or $\varphi^\mathcal{F}$.

- Model checking for finite domains is polynomial in the size of the structure.

- Assume binary domains

- BDDs are tuples $(\delta, i, \delta_1, \delta_2)$, where $\delta$ is the name of the node, $i$ is a variable from the set $\{v_1, \ldots, v_n, x_1, \ldots, x_m\}$ and each $\delta_j$ is one of the constants 0 or 1, the name of a relation variable, or the name of another BDD node.

- The interpretation $I$ of a relation is a BDD over the variables $v_1, \ldots, v_n$.

Relational $\mu$-calculus: Model checking

function BDDTerm(RelationalTerm $\rho$, Interpretation $I$) : = Bdd
  case $\rho$ of
    $R \in \mathcal{R}$: return $I(R)$; /*pointer to BDD for $R$*/
    $X \in \mathcal{V}$: return $X$; /*name of $X$*/
    $\lambda x_1 \ldots x_n \varphi$: return BDDForm($\varphi$, $I$)\{ $v_1 := x_1$ \} \cdots \{ $v_n := x_n$ \};
    $\mu X \rho$: return BDDTerm($\rho$, $I$); return BDDlfp($r$, 0);

function BDDlfp(BDD $r$, BDD $X$): BDD =
  $X^{i+1} := r(X := X^i)$;
  if ($X^{i+1} = X^i$) then return $X^i$;
  else return BDDlfp($r$, $X^{i+1}$);

function BDDForm(Formula $\varphi$, Interpretation $I$) : = Bdd
  /*Calculates the BDD of formula $\varphi$ in the interpretation $I$*/
  case $\varphi$ of
    $x \in \mathcal{V}$: return Ite($x$, 1, 0);
    ($x_1 = x_2$): return Ite($x_1$, Ite($x_2$, 1, 0), Ite($x_2$, 0, 1));
    $\bot$ return 0;
    ($\psi_1 \rightarrow \psi_2$): return BDDImp($\psi_1$, $I$), BDDForm($\psi_2$, $I$));
    $\exists x \varphi$: return BDDExists($x$, BDDForm($\varphi$, $I$));
    $\rho x_1 \ldots x_n$: return BDDTerm($\rho$, $I$)\{ $v_1 := x_1$ \} \cdots \{ $v_n := x_n$ \);
Bounded Model Checking: Example

We consider a three-bit shift register. We wish to verify $\text{AF}(x = 0)$.

The contents of the register function as state variables. The transition relation:

$$R(x, x') = (x'[0] = x[1]) \land (x'[1] = x[2]) \land (x'[2] = 1)$$

In the initial state, all registers contain 1, as represented by the predicate $I(x_i) = x_i[0] = 1 \land x_i[1] = 1 \land x_i[2] = 1$.

We identify $x_i$ with vector containing a copy of the state variables. By unrolling the transition relation we get formula

$$f_m \equiv I(x_0) \land R(x_0, x_1) \land R(x_1, x_2)$$

which represents the legal paths $x_0x_1x_2$ of length two of the system.

Bounded Model Checking: Translation to Propositional Logic

Let $M$ be a Kripke structure, $I(w)$ the initial predicate and $T(w)$ the terminal predicate.

Each state $w$ is a vector of $n$ propositional variables $w_i$.

The following formula $[[M]]$ describes the legal maximal paths $w^0 \ldots w^k$ of length $k$.

$$[[M]] = I(w^0) \land \bigwedge_{i=1}^{k} R(w^{i-1}, w^i) \land \left(T(w^k) \lor \bigvee_{i=0}^{k} R(w^k, w^i)\right)$$

The path represented by $w^0 \ldots w^k$ can represent infinite behaviour if it contains a loop.

Bounded Model Checking

- For some cases the BDD-based approach to model checking does not perform very well.

- There are systems for which an exponential BDD is required to represent the system w.r.t. the number of state variables.

- An alternative approach to symbolic model checking is to encode the problem as an instance of propositional satisfiability and use state of the art satisfiability solvers to attack the problem.

- The encoding is possible for finite domains, as translating first-order logic to linear temporal logic is possible.

Bounded Model Checking: Example

The universal model checking problem is converted to an existential by negating the formula: $\text{EG}(x \neq 0)$. Any witness to $\text{G}(x \neq 0)$ must contain a loop. Thus we require that there is a transition from $x_2$ to itself, or to $x_1$ or to $x_0$. This transition is defined as

$$L_i \equiv R(x_2, x_i)$$

The constraint imposed by the formula is that $x \neq 0$ at each state. This can be captured by the formula

$$S_i \equiv (x_i[0] = 1) \lor (x_i[1] = 1) \lor (x_i[2] = 1)$$

Putting this together we get

$$f_m \land \bigvee_{i=0}^{2} L_i \land \bigwedge_{i=0}^{2} S_i$$

This formula is satisfiable iff there is a counterexample of length 2 for the original formula $F(x = 0)$. 
Bounded Model Checking: Translation to Propositional Logic

\[
[((\varphi U^+ \psi))]_k^i = \bigvee_{j=i+1}^k \left( [[\psi]]_k^j \land \bigwedge_{m=i+1}^{j-1} [[\varphi]]_k^m \right) \lor \\
\bigvee_{l=0}^k \left( \bigwedge_{m=i+1}^l [[\varphi]]_k^m \land R(w^k, w^l) \land \bigvee_{j=l}^i \left( [[\psi]]_k^j \land \bigwedge_{m=l}^{j-1} [[\varphi]]_k^m \right) \right)
\]

The translation as it has been presented here is not very efficient. By introducing translations \(G^+, F^+, \) etc. it is possible to make a more efficient translation.

Bounded Model Checking: Translation to Propositional Logic

We define \([[\psi]]_k^i\) recursively on the structure of \(\psi\). In this recursion \(k\) is fixed while \(i\) depends on the evaluation point. Let \(k, i \in \mathbb{N}\) and \(\forall_{j=i}^k \psi = \bot\) for \(l > i\).

- \([[p]]_k^i = p(w^i)\)
- \([[\bot]]_k^i = \bot\)
- \([[\varphi \rightarrow \psi]]_k^i = ([[\varphi]]_k^i \rightarrow [[\psi]]_k^i)\)

Partial Order Methods: Introduction

- The interleaving semantics for parallel processes causes all independent events to interleave.
- The global state space includes these interleavings.
- Partial order methods aim at only generating the necessary part of the state space needed for the evaluation of a formula.
- Only representatives of these interleavings are generated.

Bounded Model Checking

**Theorem.** There exists a maximal path of length \(k\) generated by \(\mathcal{M}\) which initially validates \(\psi\) iff \(([[\mathcal{M}]][[\psi]]_k^0)\) is propositionally satisfiable.

- Without knowing an upperbound for \(k\), bounded model checking can only be used for falsification and not proving.
- For LTL, the upperbound for \(k\) is \(|\mathcal{M}| \times 2^{|\psi|}\).
- It is likely that for many cases a better upper bound exists, it is however difficult to compute.
Partial Order Methods: Stuttering Invariance

Stuttering equivalence is the concept which allows us to identify which interleavings are identical and group them into equivalence classes.

- Let $P = \{p_1, \ldots, p_k\} \subseteq \mathcal{P}$. Two natural models $\mathcal{M}$ and $\mathcal{M}'$ are strongly equivalent w.r.t. $P$, if they are of the same cardinality and for all $i \geq$ and all $p \in P$, $w_i \in \mathcal{I}(p)$ iff $w'_i \in \mathcal{I}'(p)$.

- A point $w_{i+1}$ in $\mathcal{M}$ is stuttering w.r.t. $P$, if $w_i \in \mathcal{I}(p)$ iff $w_{i+1} \in \mathcal{I}(p)$.

- The stutter-free kernel $\mathcal{M}^0$ of a model $\mathcal{M}$ is obtained by retaining all non-stuttering states of $\mathcal{M}$.

Partial Order Methods: Introduction

- The semantics of the concurrency is not changed, but the partial order nature of events is utilized.

- The partial order methods will be presented in the context of elementary Petri nets and Linear Temporal Logic.

Partial Order Methods: Analysis of Elementary Nets

We are given an elementary Petri net $N$ and an LTL-\(X\) formula $\varphi$, with atomic propositions $P_\varphi \subseteq S$.

- Independendence of two transition $t_1$ and $t_2$
  
  - Independent transitions must neither enable or disable each other
  
  - Independent transitions enabled at $m$ must be able to commute

- This definition is too hard to check. We need a syntactic condition.

A subset $T_m \subseteq T$ is persistent in a marking $m$ iff for all $t \in T_m$ and all firing sequences $t_0, t_1, \ldots, t_n, t$ such that $t_i \notin T_m$, there exists a stuttering equivalent firing sequence starting with $t$.

Partial Order Methods: Stuttering Invariance

- In $\mathcal{M}^0$: $w < w'$ iff $w < w'$ in $\mathcal{M}$ or there are stuttering points $w_1, \ldots, w_k$ such that $w < w_1 < \cdots < w_k < w'$ in $\mathcal{M}$.

- A formula $\varphi$ is stuttering invariant if for all stuttering equivalent models $\mathcal{M}, \mathcal{M}'$, $\mathcal{M} \models \varphi$ iff $\mathcal{M}' \models \varphi$.

- For our reduction to work we can use only stuttering invariant formulae.

- Let LTL-\(X\) the logic built from propositions $p \in \mathcal{P}$, boolean connectives $\bot, \rightarrow$ and the reflexive until operator $U^*$.

Lemma. Any LTL-\(X\) formula is stuttering invariant.

Theorem Any LTL formula which is stuttering invariant is expressible in LTL-\(X\).
Partial Order Methods: Analysis of Elementary Nets

Let $t_f$ be an enabled transition in $m$ and $t$ a disabled transition.

- $NEC(t, m) = \{t' \mid p \in t'\bullet\}$, for some $p \in (\bullet t \setminus m)$.
- $NEC^*(t, m) =$ transitive closure of $NEC(t, m)$.
- If $t$ is disabled in $m$, $t$ cannot fire before some transition in $NEC^*(t, m)$ fire.
- A transition is visible for $\varphi$ if $(\bullet t \cup t\bullet) \cap P_\varphi \neq \emptyset$.
- The conflict of $t$ is defined as $C(t) = \{t' \mid t' \cap \bullet t \neq \emptyset\} \cup \{t\}$.

Partial Order Methods: Analysis of Elementary Nets

If $T_m$ is persistent at $m$, we do not need to consider transitions outside $T_m$, as there will be a stuttering equivalent sequence starting with $t \in T_m$.

- The previous definition is still not efficient. No way of efficiently computing a minimal persistent set (NP-hard?).
- We approximate using heuristics.

IDEA: We start with $T_m = t$. Then we add all transitions which can “interfere” with some transition in $T_m$.

Interfere means either the transition cannot commute with some transition in $T_m$ or it enables or disables a transition in $T_m$.

Partial Order Methods: Analysis of Elementary Nets

Theorem. For any firing sequence $\rho$ of the net there exists a firing sequence $\rho'$ generated only by firing the enabled ready transitions such that $\rho$ and $\rho'$ are equivalent w.r.t. all LTL-X safety properties.

- The procedure could be extended to liveness properties by making sure a different set is generated, if a marking is reached again.
- Can at best result in an exponential reduction.
- Worst case complexity cubic in the size of the net. Average example’s complexity is linear.

Partial Order Methods: Analysis of Elementary Nets

- The extended conflict of $t$ is $C(t)$ if $t$ is invisible; otherwise it is $C(t)$ and all other visible transitions.
- A dependent set $DEP(t_f, m)$ of $t_f$ is any set of transitions such that for any $t$ in the extended conflict of $t_f$, there exists a set $NEC(t, m) \subseteq DEP(t_f, m)$.
- Transitions which are fired should be transitively closed under dependency.
- $READY(m)$ is any nonempty set of transitions s.t. $DEP(t_f, m) \subseteq READY(m)$, if $t_f \in READY(m)$. 