Program Verification: Lecture 27

José Meseguer
University of Illinois at Urbana-Champaign

Narrowing-Based Symbolic LTL Model Checking

We can verify invariants of a topmost rewrite theory $\mathcal{R} = (\Sigma, E \cup B, R)$ when $E \cup B$ is FVP by narrowing search with $\rightsquigarrow_{R/(E \cup B)}$ from a symbolic initial state $u_1 \vee \ldots \vee u_n$.

Narrowing-Based Symbolic LTL Model Checking

We can verify invariants of a topmost rewrite theory $\mathcal{R} = (\Sigma, E \cup B, R)$ when $E \cup B$ is FVP by narrowing search with $\rightsquigarrow_{R/(E \cup B)}$ from a symbolic initial state $u_1 \vee \ldots \vee u_n$. Can this be generalized to narrowing-based symbolic LTL model checking for such an $\mathcal{R}$?

Narrowing-Based Symbolic LTL Model Checking

We can verify invariants of a topmost rewrite theory $\mathcal{R} = (\Sigma, E \cup B, R)$ when $E \cup B$ is FVP by narrowing search with $\leadsto_{R/(E \cup B)}$ from a symbolic initial state $u_1 \vee \ldots \vee u_n$. Can this be generalized to narrowing-based symbolic LTL model checking for such an $\mathcal{R}$?

The main problem is that, in general, it is meaningless to say which state predicates $p \in \Pi$ are satisfied in a symbolic state u, since some ground instance $u\rho$ may satisfy some predicates in Π, and another ground instance $u\tau$ may satisfy a different set of predicates in Π.

Narrowing-Based Symbolic LTL Model Checking

We can verify invariants of a topmost rewrite theory $\mathcal{R} = (\Sigma, E \cup B, R)$ when $E \cup B$ is FVP by narrowing search with $\rightsquigarrow_{R/(E \cup B)}$ from a symbolic initial state $u_1 \vee \ldots \vee u_n$. Can this be generalized to narrowing-based symbolic LTL model checking for such an $\mathcal{R}$?

The main problem is that, in general, it is meaningless to say which state predicates $p \in \Pi$ are satisfied in a symbolic state u, since some ground instance $u\rho$ may satisfy some predicates in Π, and another ground instance $u\tau$ may satisfy a different set of predicates in Π.

However, if the states $\mathcal{R}$-reachable from $u_1 \vee \ldots \vee u_n$ are deadlock-free, and the equations D defining the satisfaction relation $u \models p$ between terms of top sort *State* and state predicates Π for the true and false cases are such that $E \cup D \cup B$ are FVP and protect `BOOL`, LTL symbolic model checking of $\mathcal{R}$ from a symbolic initial state $u_1 \vee \ldots \vee u_n$ becomes possible in a symbolic Kripke structure $\mathcal{N}_{\mathcal{R}}^{\Pi}(u_1 \vee \ldots \vee u_n)$, whose symbolic transitions are performed by a Π-aware narrowing relation $\rightsquigarrow_{\Pi}$.

The Narrowing Relation $\rightsquigarrow_{\Pi}$

Given a topmost rewrite theory $\mathcal{R} = (\Sigma, E \cup B, R)$ with rules $(l \to r) \in R$ s.t. $l, r \in T_{\Sigma}(X) \setminus X$, topmost of sort $State$, and a set $\Pi = \{p_1, \ldots, p_k\}$ of state predicates whose satisfaction in $\mathcal{R}$ is defined by equations D such that $E \cup D \cup B$ is FVP modulo axioms B, the Π-aware narrowing relation between terms $u, w \in T_{\Sigma, State}(X)$ is defined as follows:

The Narrowing Relation $\rightsquigarrow_\Pi$

Given a topmost rewrite theory $\mathcal{R} = (\Sigma, E \cup B, R)$ with rules $(l \to r) \in R$ s.t. $l, r \in T_\Sigma(X) \setminus X$, topmost of sort *State*, and a set $\Pi = \{p_1, \ldots, p_k\}$ of state predicates whose satisfaction in $\mathcal{R}$ is defined by equations D such that $E \cup D \cup B$ is FVP modulo axioms B, the Π-aware narrowing relation between terms $u, w \in T_{\Sigma,State}(X)$ is defined as follows:

$$u \overset{\alpha\gamma}{\rightsquigarrow}_\Pi w$$

holds iff (by definition)

The Narrowing Relation $\leadsto_{\Pi}$

Given a topmost rewrite theory $\mathcal{R} = (\Sigma, E \cup B, R)$ with rules $(l \to r) \in R$ s.t. $l, r \in T_{\Sigma}(X) \setminus X$, topmost of sort $State$, and a set $\Pi = \{p_1, \ldots, p_k\}$ of state predicates whose satisfaction in $\mathcal{R}$ is defined by equations D such that $E \cup D \cup B$ is FVP modulo axioms B, the Π-aware narrowing relation between terms $u, w \in T_{\Sigma, State}(X)$ is defined as follows:

$$u \overset{\alpha\gamma}{\leadsto}_{\Pi} w$$

holds iff (by definition)

- $\exists v \;\; s.t. \;\; u \overset{\alpha}{\leadsto}_{R/(E \cup B)} v$

The Narrowing Relation $\rightsquigarrow_{\Pi}$

Given a topmost rewrite theory $\mathcal{R} = (\Sigma, E \cup B, R)$ with rules $(l \to r) \in R$ s.t. $l, r \in T_{\Sigma}(X) \setminus X$, topmost of sort *State*, and a set $\Pi = \{p_1, \ldots, p_k\}$ of state predicates whose satisfaction in $\mathcal{R}$ is defined by equations D such that $E \cup D \cup B$ is FVP modulo axioms B, the Π-aware narrowing relation between terms $u, w \in T_{\Sigma,State}(X)$ is defined as follows:

$$u \overset{\alpha\gamma}{\rightsquigarrow}_{\Pi} w$$

holds iff (by definition)

- $\exists v \;\; s.t. \;\; u \rightsquigarrow^{\alpha}_{R/(E \cup B)} v$
- $\exists (b_1, \ldots, b_k) \in \{true, false\}^k$

The Narrowing Relation $\rightsquigarrow_{\Pi}$

Given a topmost rewrite theory $\mathcal{R} = (\Sigma, E \cup B, R)$ with rules $(l \to r) \in R$ s.t. $l, r \in T_{\Sigma}(X) \setminus X$, topmost of sort *State*, and a set $\Pi = \{p_1, \ldots, p_k\}$ of state predicates whose satisfaction in $\mathcal{R}$ is defined by equations D such that $E \cup D \cup B$ is FVP modulo axioms B, the Π-aware narrowing relation between terms $u, w \in T_{\Sigma,State}(X)$ is defined as follows:

$$u \overset{\alpha\gamma}{\rightsquigarrow}_{\Pi} w$$

holds iff (by definition)

- $\exists v \;\; s.t. \;\; u \rightsquigarrow^{\alpha}_{R/(E \cup B)} v$
- $\exists (b_1, \ldots, b_k) \in \{true, false\}^k$
- $\exists \gamma \in Unif_{E \cup D \cup B}(v \models p_1 = b_1 \wedge \ldots \wedge v \models p_k = b_k)$

The Narrowing Relation $\rightsquigarrow_{\Pi}$

Given a topmost rewrite theory $\mathcal{R} = (\Sigma, E \cup B, R)$ with rules $(l \rightarrow r) \in R$ s.t. $l, r \in T_{\Sigma}(X) \setminus X$, topmost of sort *State*, and a set $\Pi = \{p_1, \ldots, p_k\}$ of state predicates whose satisfaction in $\mathcal{R}$ is defined by equations D such that $E \cup D \cup B$ is FVP modulo axioms B, the Π-aware narrowing relation between terms $u, w \in T_{\Sigma, State}(X)$ is defined as follows:

$$u \overset{\alpha\gamma}{\rightsquigarrow}_{\Pi} w$$

holds iff (by definition)

- $\exists v \;\; s.t. \;\; u \rightsquigarrow^{\alpha}_{R/(E \cup B)} v$
- $\exists (b_1, \ldots, b_k) \in \{true, false\}^k$
- $\exists \gamma \in Unif_{E \cup D \cup B}(v \models p_1 = b_1 \wedge \ldots \wedge v \models p_k = b_k)$

such that $w = v\gamma$.

The Kripke Structure $\mathcal{N}_{\mathcal{R}}^{\Pi}(\bigvee_{i \in I} u_i)$

For $\bigvee_{i \in I} u_i$, $I = \{1, \ldots, n\}$, define its Π-instances $\{u'_1, \ldots, u'_m\} =$
$\{u_i \gamma \mid i \in I,\ \exists (b_1, \ldots, b_k) \in \{true, false\}^k,\ \exists \gamma \in Unif_{E \cup D \cup B}(u \models p_1 = b_1 \wedge \ldots \wedge u \models p_k = b_k)\}$.
The Kripke structure $\mathcal{N}_{\mathcal{R}}^{\Pi}(\bigvee_{i \in I} u_i)$ has states

The Kripke Structure $\mathcal{N}^{\Pi}_{\mathcal{R}}(\bigvee_{i\in I} u_i)$

For $\bigvee_{i\in I} u_i$, $I = \{1,\ldots,n\}$, define its Π-instances $\{u'_1,\ldots,u'_m\} =$
$\{u_i\gamma \mid i \in I,\ \exists(b_1,\ldots,b_k) \in \{true,false\}^k,\ \exists\gamma \in Unif_{E\cup D\cup B}(u \models p_1 = b_1 \wedge \ldots \wedge u \models p_k = b_k)\}$.

The Kripke structure $\mathcal{N}^{\Pi}_{\mathcal{R}}(\bigvee_{i\in I} u_i)$ has states $N^{\Pi}_{\mathcal{R}}(\bigvee_{i\in I} u_i) =$

The Kripke Structure $\mathcal{N}^{\Pi}_{\mathcal{R}}(\bigvee_{i\in I} u_i)$

For $\bigvee_{i\in I} u_i$, $I = \{1,\ldots,n\}$, define its Π-instances $\{u'_1,\ldots,u'_m\} =$
$\{u_i\gamma \mid i \in I,\ \exists(b_1,\ldots,b_k) \in \{true,false\}^k,\ \exists\gamma \in Unif_{E\cup D\cup B}(u \models p_1 = b_1 \wedge \ldots \wedge u \models p_k = b_k)\}$.
The Kripke structure $\mathcal{N}^{\Pi}_{\mathcal{R}}(\bigvee_{i\in I} u_i)$ has states $N^{\Pi}_{\mathcal{R}}(\bigvee_{i\in I} u_i) =$
$\{w \in T_{\Sigma,State}(X) \mid \exists j,\ 1 \leq j \leq m,\ u'_j \rightsquigarrow^{*}_{\Pi} w\}/{\approx_{E\cup B}}$,

The Kripke Structure $\mathcal{N}^{\Pi}_{\mathcal{R}}(\bigvee_{i\in I} u_i)$

For $\bigvee_{i\in I} u_i$, $I = \{1,\ldots,n\}$, define its Π-instances $\{u'_1,\ldots,u'_m\} =$ $\{u_i\gamma \mid i \in I,\ \exists(b_1,\ldots,b_k) \in \{true,false\}^k,\ \exists\gamma \in Unif_{E\cup D\cup B}(u \models p_1 = b_1 \wedge \ldots \wedge u \models p_k = b_k)\}$.

The Kripke structure $\mathcal{N}^{\Pi}_{\mathcal{R}}(\bigvee_{i\in I} u_i)$ has states $N^{\Pi}_{\mathcal{R}}(\bigvee_{i\in I} u_i) =$ $\{w \in T_{\Sigma,State}(X) \mid \exists j,\ 1 \leq j \leq m,\ u'_j \rightsquigarrow^*_{\Pi} w\}/{\approx_{E\cup B}}$, where $v \approx_{E\cup B} w$ iff exists a variable renaming α s.t. $v!_{\vec{E}/B}\alpha =_B w!_{\vec{E}/B}$,

The Kripke Structure $\mathcal{N}^{\Pi}_{\mathcal{R}}(\bigvee_{i \in I} u_i)$

For $\bigvee_{i \in I} u_i$, $I = \{1, \ldots, n\}$, define its Π-instances $\{u'_1, \ldots, u'_m\} =$
$\{u_i\gamma \mid i \in I,\ \exists(b_1, \ldots, b_k) \in \{true, false\}^k,\ \exists\gamma \in Unif_{E \cup D \cup B}(u \models p_1 = b_1 \wedge \ldots \wedge u \models p_k = b_k)\}$.
The Kripke structure $\mathcal{N}^{\Pi}_{\mathcal{R}}(\bigvee_{i \in I} u_i)$ has states $N^{\Pi}_{\mathcal{R}}(\bigvee_{i \in I} u_i) =$
$\{w \in T_{\Sigma,State}(X) \mid \exists j,\ 1 \leq j \leq m,\ u'_j \rightsquigarrow^*_{\Pi} w\}/{\approx_{E \cup B}}$, where $v \approx_{E \cup B} w$ iff exists a variable renaming α s.t. $v!_{\vec{E}/B}\alpha =_B w!_{\vec{E}/B}$, transition relation
$\rightsquigarrow_{\Pi}$,

The Kripke Structure $\mathcal{N}_{\mathcal{R}}^{\Pi}(\bigvee_{i \in I} u_i)$

For $\bigvee_{i \in I} u_i$, $I = \{1, \ldots, n\}$, define its Π-instances $\{u'_1, \ldots, u'_m\} =$
$\{u_i\gamma \mid i \in I,\ \exists(b_1, \ldots, b_k) \in \{true, false\}^k,\ \exists\gamma \in Unif_{E \cup D \cup B}(u \models p_1 = b_1 \wedge \ldots \wedge u \models p_k = b_k)\}$.

The Kripke structure $\mathcal{N}_{\mathcal{R}}^{\Pi}(\bigvee_{i \in I} u_i)$ has states $N_{\mathcal{R}}^{\Pi}(\bigvee_{i \in I} u_i) =$
$\{w \in T_{\Sigma,State}(X) \mid \exists j,\ 1 \leq j \leq m,\ u'_j \rightsquigarrow_{\Pi}^{*} w\}/{\approx_{E \cup B}}$, where $v \approx_{E \cup B} w$ iff exists a variable renaming α s.t. $v!_{\vec{E}/B}\alpha =_B w!_{\vec{E}/B}$, transition relation $\rightsquigarrow_{\Pi}$, and satisfaction relation $[w] \models_{\mathcal{N}_{\mathcal{R}}^{\Pi}(\bigvee_{i \in I} u_i)} p_i$ defined for each $[w] \in N_{\mathcal{R}}^{\Pi}(\bigvee_{i \in I} u_i)$ and $p_i \in \Pi$ by the unique $b'_i \in \{true, false\}^k$ such that $(w \models p_i)!_{\vec{E \cup D},B} = b'_i$, $1 \leq i \leq k$.

The Kripke Structure $\mathcal{N}_{\mathcal{R}}^{\Pi}(\bigvee_{i\in I} u_i)$

For $\bigvee_{i\in I} u_i$, $I = \{1,\ldots,n\}$, define its Π-instances $\{u_1',\ldots,u_m'\} =$
$\{u_i\gamma \mid i \in I,\ \exists(b_1,\ldots,b_k) \in \{\mathit{true},\mathit{false}\}^k,\ \exists\gamma \in \mathit{Unif}_{E\cup D\cup B}(u \models p_1 = b_1 \wedge \ldots \wedge u \models p_k = b_k)\}$.
The Kripke structure $\mathcal{N}_{\mathcal{R}}^{\Pi}(\bigvee_{i\in I} u_i)$ has states $N_{\mathcal{R}}^{\Pi}(\bigvee_{i\in I} u_i) =$
$\{w \in T_{\Sigma,\mathit{State}}(X) \mid \exists j,\ 1 \leq j \leq m,\ u_j' \rightsquigarrow_{\Pi}^{*} w\}/{\approx_{E\cup B}}$, where $v \approx_{E\cup B} w$ iff exists a variable renaming α s.t. $v!_{\vec{E}/B}\alpha =_B w!_{\vec{E}/B}$, transition relation $\rightsquigarrow_{\Pi}$, and satisfaction relation $[w] \models_{\mathcal{N}_{\mathcal{R}}^{\Pi}(\bigvee_{i\in I} u_i)} p_i$ defined for each $[w] \in N_{\mathcal{R}}^{\Pi}(\bigvee_{i\in I} u_i)$ and $p_i \in \Pi$ by the unique $b_i' \in \{\mathit{true},\mathit{false}\}^k$ such that $(w \models p_i)!_{\vec{E\cup D},B} = b_i'$, $1 \leq i \leq k$.

If $\mathcal{N}_{\mathcal{R}}^{\Pi}(\bigvee_{i\in I} u_i)$ is deadlock-free, any LTL formula φ holds for a symbolic initial state $\bigvee_{i\in I} u_i$ in $\mathbb{T}_{\mathcal{R}}^{\Pi}$ if (resp. iff) it does in $\mathcal{N}_{\mathcal{R}}^{\Pi}(\bigvee_{i\in I} u_i)$ from $\{u_1',\ldots,u_m'\}$ (resp. assuming $N_{\mathcal{R}}^{\Pi}(\bigvee_{i\in I} u_i)$ is finite) (see Appendix 1):

The Kripke Structure $\mathcal{N}^{\Pi}_{\mathcal{R}}(\bigvee_{i\in I} u_i)$

For $\bigvee_{i\in I} u_i$, $I = \{1,\ldots,n\}$, define its Π-instances $\{u'_1,\ldots,u'_m\} =$
$\{u_i\gamma \mid i \in I,\ \exists(b_1,\ldots,b_k) \in \{true, false\}^k,\ \exists\gamma \in Unif_{E\cup D\cup B}(u \models p_1 = b_1 \wedge \ldots \wedge u \models p_k = b_k)\}$.
The Kripke structure $\mathcal{N}^{\Pi}_{\mathcal{R}}(\bigvee_{i\in I} u_i)$ has states $N^{\Pi}_{\mathcal{R}}(\bigvee_{i\in I} u_i) =$
$\{w \in T_{\Sigma,State}(X) \mid \exists j,\ 1 \leq j \leq m,\ u'_j \rightsquigarrow^{*}_{\Pi} w\}/{\approx_{E\cup B}}$, where $v \approx_{E\cup B} w$ iff exists a variable renaming α s.t. $v!_{\vec{E}/B}\alpha =_B w!_{\vec{E}/B}$, transition relation $\rightsquigarrow_{\Pi}$, and satisfaction relation $[w] \models_{\mathcal{N}^{\Pi}_{\mathcal{R}}(\bigvee_{i\in I} u_i)} p_i$ defined for each $[w] \in N^{\Pi}_{\mathcal{R}}(\bigvee_{i\in I} u_i)$ and $p_i \in \Pi$ by the unique $b'_i \in \{true, false\}^k$ such that $(w \models p_i)!_{\vec{E\cup D},B} = b'_i$, $1 \leq i \leq k$.

If $\mathcal{N}^{\Pi}_{\mathcal{R}}(\bigvee_{i\in I} u_i)$ is deadlock-free, any LTL formula φ holds for a symbolic initial state $\bigvee_{i\in I} u_i$ in $\mathbb{T}^{\Pi}_{\mathcal{R}}$ if (resp. iff) it does in $\mathcal{N}^{\Pi}_{\mathcal{R}}(\bigvee_{i\in I} u_i)$ from $\{u'_1,\ldots,u'_m\}$ (resp. assuming $N^{\Pi}_{\mathcal{R}}(\bigvee_{i\in I} u_i)$ is finite) (see Appendix 1):

Theorem

For $\varphi \in LTL(\Pi)$ (resp. assuming $N^{\Pi}_{\mathcal{R}}(\bigvee_{i\in I} u_i)$ is a finite set)

$$\mathcal{N}^{\Pi}_{\mathcal{R}}(\bigvee_{i\in I} u_i), \{u'_1,\ldots,u'_m\} \models_{LTL} \varphi. \ \Rightarrow\ (resp.\ \Leftrightarrow)\ \mathbb{T}^{\Pi}_{\mathcal{R}}, [\![\bigvee_{i\in I} u_i]\!]_{E\cup B} \models_{LTL} \varphi.$$

State Space Reduction in $\mathcal{N}^{\Pi}_{\mathcal{R}}(\bigvee_{i \in I} u_i)$

By the above Theorem, if the state space $N^{\Pi}_{\mathcal{R}}(\bigvee_{i \in I} u_i)$ is finite, the Kripke structure $\mathcal{N}^{\Pi}_{\mathcal{R}}(\bigvee_{i \in I} u_i)$ supports explicit-state LTL model checking using the decision procedure described in Lecture 23 to verify $\mathbb{T}^{\Pi}_{\mathcal{R}}, [\![\bigvee_{i \in I} u_i]\!]_{E \cup B} \models_{LTL} \varphi$.

State Space Reduction in $\mathcal{N}_{\mathcal{R}}^{\Pi}(\bigvee_{i\in I} u_i)$

By the above Theorem, if the state space $N_{\mathcal{R}}^{\Pi}(\bigvee_{i\in I} u_i)$ is finite, the Kripke structure $\mathcal{N}_{\mathcal{R}}^{\Pi}(\bigvee_{i\in I} u_i)$ supports explicit-state LTL model checking using the decision procedure described in Lecture 23 to verify $\mathbb{T}_{\mathcal{R}}^{\Pi}, [\![\bigvee_{i\in I} u_i]\!]_{E\cup B} \models_{LTL} \varphi$.

When $N_{\mathcal{R}}^{\Pi}(\bigvee_{i\in I} u_i)$ is infinite, we can try one of the following three possibilities to reduce the state space of $\mathcal{N}_{\mathcal{R}}^{\Pi}(\bigvee_{i\in I} u_i)$ to a finite state space:

State Space Reduction in $\mathcal{N}^{\Pi}_{\mathcal{R}}(\bigvee_{i\in I} u_i)$

By the above Theorem, if the state space $N^{\Pi}_{\mathcal{R}}(\bigvee_{i\in I} u_i)$ is finite, the Kripke structure $\mathcal{N}^{\Pi}_{\mathcal{R}}(\bigvee_{i\in I} u_i)$ supports explicit-state LTL model checking using the decision procedure described in Lecture 23 to verify $\mathbb{T}^{\Pi}_{\mathcal{R}}, [\![\bigvee_{i\in I} u_i]\!]_{E\cup B} \models_{LTL} \varphi$.

When $N^{\Pi}_{\mathcal{R}}(\bigvee_{i\in I} u_i)$ is infinite, we can try one of the following three possibilities to reduce the state space of $\mathcal{N}^{\Pi}_{\mathcal{R}}(\bigvee_{i\in I} u_i)$ to a finite state space:

1. Perform LTL model checking by folding variant narrowing, provided the folding $\rightsquigarrow_{\Pi}$-narrowing forest from $\{u'_1, \ldots, u'_m\}$ is finite.

State Space Reduction in $\mathcal{N}^{\Pi}_{\mathcal{R}}(\bigvee_{i\in I} u_i)$

By the above Theorem, if the state space $N^{\Pi}_{\mathcal{R}}(\bigvee_{i\in I} u_i)$ is finite, the Kripke structure $\mathcal{N}^{\Pi}_{\mathcal{R}}(\bigvee_{i\in I} u_i)$ supports explicit-state LTL model checking using the decision procedure described in Lecture 23 to verify $\mathbb{T}^{\Pi}_{\mathcal{R}}, [\![\bigvee_{i\in I} u_i]\!]_{E\cup B} \models_{LTL} \varphi$.

When $N^{\Pi}_{\mathcal{R}}(\bigvee_{i\in I} u_i)$ is infinite, we can try one of the following three possibilities to reduce the state space of $\mathcal{N}^{\Pi}_{\mathcal{R}}(\bigvee_{i\in I} u_i)$ to a finite state space:

1. Perform LTL model checking by folding variant narrowing, provided the folding $\rightsquigarrow_{\Pi}$-narrowing forest from $\{u'_1, \ldots, u'_m\}$ is finite.
2. Define an equational abstraction $\mathcal{R}/G$ s.t.: (i) $E \cup D \cup D' \cup G \cup B$ is FVP and protects `BOOL`, and (ii) the folding $\rightsquigarrow_{\Pi}$-narrowing forest is finite for $\mathcal{N}^{\Pi}_{\mathcal{R}/G}(\bigvee_{i\in I} u_i)$.

State Space Reduction in $\mathcal{N}^{\Pi}_{\mathcal{R}}(\bigvee_{i\in I} u_i)$

By the above Theorem, if the state space $N^{\Pi}_{\mathcal{R}}(\bigvee_{i\in I} u_i)$ is finite, the Kripke structure $\mathcal{N}^{\Pi}_{\mathcal{R}}(\bigvee_{i\in I} u_i)$ supports explicit-state LTL model checking using the decision procedure described in Lecture 23 to verify $\mathbb{T}^{\Pi}_{\mathcal{R}}, [\![\bigvee_{i\in I} u_i]\!]_{E\cup B} \models_{LTL} \varphi$.

When $N^{\Pi}_{\mathcal{R}}(\bigvee_{i\in I} u_i)$ is infinite, we can try one of the following three possibilities to reduce the state space of $\mathcal{N}^{\Pi}_{\mathcal{R}}(\bigvee_{i\in I} u_i)$ to a finite state space:

1. Perform LTL model checking by folding variant narrowing, provided the folding $\rightsquigarrow_{\Pi}$-narrowing forest from $\{u'_1, \ldots, u'_m\}$ is finite.
2. Define an equational abstraction $\mathcal{R}/G$ s.t.: (i) $E \cup D \cup D' \cup G \cup B$ is FVP and protects `BOOL`, and (ii) the folding $\rightsquigarrow_{\Pi}$-narrowing forest is finite for $\mathcal{N}^{\Pi}_{\mathcal{R}/G}(\bigvee_{i\in I} u_i)$.
3. Perform bounded LTL symbolic model checking.

The Folding $\rightsquigarrow_{\Pi}$-narrowing forest from $\{u'_1, \ldots, u'_m\}$

Replacing $\rightsquigarrow_{R/(E\cup B)}$ by $\rightsquigarrow_{\Pi}$, just as we have a folding narrowing forest $FNF_{\mathcal{R}}(\bigvee_{i\in I} u_i)$ for the $\rightsquigarrow_{R/(E\cup B)}$-narrowing tree, we also have a folding narrowing forest (a Kripke structure!) $FNF_{\mathcal{R}}^{\Pi}(\bigvee_{j\in J} u'_j)$ for $\mathcal{N}_{\mathcal{R}}^{\Pi}(\bigvee_{i\in I} u_i)$ from $\{u'_j\}_{j\in J}$, $J = \{1, \ldots m\}$, the Π-instances of $\bigvee_{i\in I} u_i$.

The Folding $\leadsto_{\Pi}$-narrowing forest from $\{u'_1, \ldots, u'_m\}$

Replacing $\leadsto_{R/(E\cup B)}$ by $\leadsto_{\Pi}$, just as we have a folding narrowing forest $FNF_{\mathcal{R}}(\bigvee_{i\in I} u_i)$ for the $\leadsto_{R/(E\cup B)}$-narrowing tree, we also have a folding narrowing forest (a Kripke structure!) $FNF^{\Pi}_{\mathcal{R}}(\bigvee_{j\in J} u'_j)$ for $\mathcal{N}^{\Pi}_{\mathcal{R}}(\bigvee_{i\in I} u_i)$ from $\{u'_j\}_{j\in J}$, $J = \{1, \ldots m\}$, the Π-instances of $\bigvee_{i\in I} u_i$.

The construction of $FNF^{\Pi}_{\mathcal{R}}(\bigvee_{j\in J} u'_j)$ is similar to that of $FNF_{\mathcal{R}}(\bigvee_{i\in I} u_i)$ in Lecture 25, replacing the folding relation $v \sqsubseteq_{E\cup B} w$ by the folding relation $v \sqsubseteq^{\Pi}_{E\cup D\cup B} w$ defined by the equivalence:

The Folding $\rightsquigarrow_{\Pi}$-narrowing forest from $\{u'_1, \ldots, u'_m\}$

Replacing $\rightsquigarrow_{R/(E\cup B)}$ by $\rightsquigarrow_{\Pi}$, just as we have a folding narrowing forest $FNF_{\mathcal{R}}(\bigvee_{i\in I} u_i)$ for the $\rightsquigarrow_{R/(E\cup B)}$-narrowing tree, we also have a folding narrowing forest (a Kripke structure!) $FNF^{\Pi}_{\mathcal{R}}(\bigvee_{j\in J} u'_j)$ for $\mathcal{N}^{\Pi}_{\mathcal{R}}(\bigvee_{i\in I} u_i)$ from $\{u'_j\}_{j\in J}$, $J = \{1, \ldots m\}$, the Π-instances of $\bigvee_{i\in I} u_i$.

The construction of $FNF^{\Pi}_{\mathcal{R}}(\bigvee_{j\in J} u'_j)$ is similar to that of $FNF_{\mathcal{R}}(\bigvee_{i\in I} u_i)$ in Lecture 25, replacing the folding relation $v \sqsubseteq_{E\cup B} w$ by the folding relation $v \sqsubseteq^{\Pi}_{E\cup D\cup B} w$ defined by the equivalence:

$$v \sqsubseteq^{\Pi}_{E\cup D\cup B} w \quad \Leftrightarrow_{def} \quad v \sqsubseteq_{E\cup B} w \wedge \forall p \in \Pi,\ (v \models p)!_{\vec{E\cup D},B} = (w \models p)!_{\vec{E\cup D},B}.$$

The Folding $\rightsquigarrow_{\Pi}$-narrowing forest from $\{u'_1, \ldots, u'_m\}$

Replacing $\rightsquigarrow_{R/(E \cup B)}$ by $\rightsquigarrow_{\Pi}$, just as we have a folding narrowing forest $FNF_{\mathcal{R}}(\bigvee_{i \in I} u_i)$ for the $\rightsquigarrow_{R/(E \cup B)}$-narrowing tree, we also have a folding narrowing forest (a Kripke structure!) $FNF_{\mathcal{R}}^{\Pi}(\bigvee_{j \in J} u'_j)$ for $\mathcal{N}_{\mathcal{R}}^{\Pi}(\bigvee_{i \in I} u_i)$ from $\{u'_j\}_{j \in J}$, $J = \{1, \ldots m\}$, the Π-instances of $\bigvee_{i \in I} u_i$.

The construction of $FNF_{\mathcal{R}}^{\Pi}(\bigvee_{j \in J} u'_j)$ is similar to that of $FNF_{\mathcal{R}}(\bigvee_{i \in I} u_i)$ in Lecture 25, replacing the folding relation $v \sqsubseteq_{E \cup B} w$ by the folding relation $v \sqsubseteq_{E \cup D \cup B}^{\Pi} w$ defined by the equivalence:

$$v \sqsubseteq_{E \cup D \cup B}^{\Pi} w \quad \Leftrightarrow_{def} \quad v \sqsubseteq_{E \cup B} w \wedge \forall p \in \Pi,\ (v \models p)!_{\vec{E \cup D},B} = (w \models p)!_{\vec{E \cup D},B}.$$

and adding extra transitions for each folding.

The Folding $\rightsquigarrow_{\Pi}$-narrowing forest from $\{u'_1,\ldots,u'_m\}$

Replacing $\rightsquigarrow_{R/(E\cup B)}$ by $\rightsquigarrow_{\Pi}$, just as we have a folding narrowing forest $FNF_{\mathcal{R}}(\bigvee_{i\in I} u_i)$ for the $\rightsquigarrow_{R/(E\cup B)}$-narrowing tree, we also have a folding narrowing forest (a Kripke structure!) $FNF_{\mathcal{R}}^{\Pi}(\bigvee_{j\in J} u'_j)$ for $\mathcal{N}_{\mathcal{R}}^{\Pi}(\bigvee_{i\in I} u_i)$ from $\{u'_j\}_{j\in J}$, $J=\{1,\ldots m\}$, the Π-instances of $\bigvee_{i\in I} u_i$.

The construction of $FNF_{\mathcal{R}}^{\Pi}(\bigvee_{j\in J} u'_j)$ is similar to that of $FNF_{\mathcal{R}}(\bigvee_{i\in I} u_i)$ in Lecture 25, replacing the folding relation $v \sqsubseteq_{E\cup B} w$ by the folding relation $v \sqsubseteq_{E\cup D\cup B}^{\Pi} w$ defined by the equivalence:

$$v \sqsubseteq_{E\cup D\cup B}^{\Pi} w \quad \Leftrightarrow_{def} \quad v \sqsubseteq_{E\cup B} w \wedge \forall p\in\Pi,\ (v \models p)!_{\vec{E\cup D},B} = (w \models p)!_{\vec{E\cup D},B}.$$

and adding extra transitions for each folding. The Completeness Theorem for $FNF_{\mathcal{R}}(\bigvee_{i\in I} u_i)$ in Lecture 25 generalizes to (**Ths** 8,12 in Appendix 2):

The Folding $\leadsto_{\Pi}$-narrowing forest from $\{u'_1, \ldots, u'_m\}$

Replacing $\leadsto_{R/(E \cup B)}$ by $\leadsto_{\Pi}$, just as we have a folding narrowing forest $FNF_{\mathcal{R}}(\bigvee_{i \in I} u_i)$ for the $\leadsto_{R/(E \cup B)}$-narrowing tree, we also have a folding narrowing forest (a Kripke structure!) $FNF^{\Pi}_{\mathcal{R}}(\bigvee_{j \in J} u'_j)$ for $\mathcal{N}^{\Pi}_{\mathcal{R}}(\bigvee_{i \in I} u_i)$ from $\{u'_j\}_{j \in J}$, $J = \{1, \ldots m\}$, the Π-instances of $\bigvee_{i \in I} u_i$.

The construction of $FNF^{\Pi}_{\mathcal{R}}(\bigvee_{j \in J} u'_j)$ is similar to that of $FNF_{\mathcal{R}}(\bigvee_{i \in I} u_i)$ in Lecture 25, replacing the folding relation $v \sqsubseteq_{E \cup B} w$ by the folding relation $v \sqsubseteq^{\Pi}_{E \cup D \cup B} w$ defined by the equivalence:

$$v \sqsubseteq^{\Pi}_{E \cup D \cup B} w \quad \Leftrightarrow_{def} \quad v \sqsubseteq_{E \cup B} w \wedge \forall p \in \Pi, \ (v \models p)!_{\vec{E \cup D},B} = (w \models p)!_{\vec{E \cup D},B}.$$

and adding extra transitions for each folding. The Completeness Theorem for $FNF_{\mathcal{R}}(\bigvee_{i \in I} u_i)$ in Lecture 25 generalizes to (**Ths** 8,12 in Appendix 2):

Theorem

For $\varphi \in LTL(\Pi)$ (resp. φ a safety formula) we have:

$$FNF^{\Pi}_{\mathcal{R}}(\bigvee_{j \in J} u'_j), \{u'_j\}_{j \in J} \models \varphi \ \Rightarrow \ (resp. \ \Leftrightarrow) \ \mathcal{N}^{\Pi}_{\mathcal{R}}(\bigvee_{i \in I} u_i), \{u'_j\}_{j \in J} \models \varphi.$$

State Space Reduction through Equational Abstractions

Under the assumptions about $\mathcal{R}$ in pg. 2, and those about $\mathcal{R}/G$ in (2) of pg. 5, we are back in the game: $\mathcal{R}/G$ itself satisfies the assumptions in pg. 2. Therefore, for $\varphi \in LTL(\Pi)$ we have (by **Theorem** in pg. 6):

State Space Reduction through Equational Abstractions

Under the assumptions about $\mathcal{R}$ in pg. 2, and those about $\mathcal{R}/G$ in (2) of pg. 5, we are back in the game: $\mathcal{R}/G$ itself satisfies the assumptions in pg. 2. Therefore, for $\varphi \in LTL(\Pi)$ we have (by **Theorem** in pg. 6):

$$(\dagger)\;\; FNF^{\Pi}_{\mathcal{R}/G}(\bigvee_{l\in L} u''_l), \{u''_l\}_{l\in L} \models \varphi \;\;\Rightarrow\;\; \mathcal{N}^{\Pi}_{\mathcal{R}/G}(\bigvee_{i\in I} u_i), \{u'_j\}_{j\in J} \models \varphi.$$

State Space Reduction through Equational Abstractions

Under the assumptions about $\mathcal{R}$ in pg. 2, and those about $\mathcal{R}/G$ in (2) of pg. 5, we are back in the game: $\mathcal{R}/G$ itself satisfies the assumptions in pg. 2. Therefore, for $\varphi \in LTL(\Pi)$ we have (by **Theorem** in pg. 6):

$$(\dagger)\ \ FNF^{\Pi}_{\mathcal{R}/G}(\bigvee_{l\in L} u''_l), \{u''_l\}_{l\in L} \models \varphi \ \ \Rightarrow\ \ \mathcal{N}^{\Pi}_{\mathcal{R}/G}(\bigvee_{i\in I} u_i), \{u'_j\}_{j\in J} \models \varphi.$$

where the $\{u''_l\}_{l\in L}$ are the Π-instances of $\bigvee_{i\in I} u_i$ in $\mathcal{R}/G$.

State Space Reduction through Equational Abstractions

Under the assumptions about $\mathcal{R}$ in pg. 2, and those about $\mathcal{R}/G$ in (2) of pg. 5, we are back in the game: $\mathcal{R}/G$ itself satisfies the assumptions in pg. 2. Therefore, for $\varphi \in LTL(\Pi)$ we have (by **Theorem** in pg. 6):

$$(\dagger)\ \ FNF^{\Pi}_{\mathcal{R}/G}(\bigvee_{l\in L} u''_l), \{u''_l\}_{l\in L} \models \varphi \ \ \Rightarrow\ \ \mathcal{N}^{\Pi}_{\mathcal{R}/G}(\bigvee_{i\in I} u_i), \{u'_j\}_{j\in J} \models \varphi.$$

where the $\{u''_l\}_{l\in L}$ are the Π-instances of $\bigvee_{i\in I} u_i$ in $\mathcal{R}/G$. Furthermore, it follows from **Theorem** in pg. 4 and **Theorem** 3 in Appendix to Lecture 26 (proof in Appendix 1), that we also have the implications:

State Space Reduction through Equational Abstractions

Under the assumptions about $\mathcal{R}$ in pg. 2, and those about $\mathcal{R}/G$ in (2) of pg. 5, we are back in the game: $\mathcal{R}/G$ itself satisfies the assumptions in pg. 2. Therefore, for $\varphi \in LTL(\Pi)$ we have (by **Theorem** in pg. 6):

$$(\dagger)\;\; FNF^{\Pi}_{\mathcal{R}/G}(\bigvee_{l\in L} u''_l), \{u''_l\}_{l\in L} \models \varphi \;\;\Rightarrow\;\; \mathcal{N}^{\Pi}_{\mathcal{R}/G}(\bigvee_{i\in I} u_i), \{u'_j\}_{j\in J} \models \varphi.$$

where the $\{u''_l\}_{l\in L}$ are the Π-instances of $\bigvee_{i\in I} u_i$ in $\mathcal{R}/G$. Furthermore, it follows from **Theorem** in pg. 4 and **Theorem** 3 in Appendix to Lecture 26 (proof in Appendix 1), that we also have the implications:

$$(\ddagger)\;\; \mathcal{N}^{\Pi}_{\mathcal{R}/G}(\bigvee_{i\in I} u_i), \{u'_j\}_{j\in J} \models \varphi \;\Rightarrow\; \mathbb{T}^{\Pi}_{\mathcal{R}/G}, [\![\bigvee_{i\in I} u_i]\!]_{E\cup G\cup B} \models_{LTL} \varphi \;\Rightarrow\; \mathbb{T}^{\Pi}_{\mathcal{R}}, [\![\bigvee_{i\in I} u_i]\!]_{E\cup B} \models_{LTL} \varphi$$

State Space Reduction through Equational Abstractions

Under the assumptions about $\mathcal{R}$ in pg. 2, and those about $\mathcal{R}/G$ in (2) of pg. 5, we are back in the game: $\mathcal{R}/G$ itself satisfies the assumptions in pg. 2. Therefore, for $\varphi \in LTL(\Pi)$ we have (by **Theorem** in pg. 6):

$$(\dagger)\;\; FNF^{\Pi}_{\mathcal{R}/G}(\bigvee_{l\in L} u''_l), \{u''_l\}_{l\in L} \models \varphi \;\;\Rightarrow\;\; \mathcal{N}^{\Pi}_{\mathcal{R}/G}(\bigvee_{i\in I} u_i), \{u'_j\}_{j\in J} \models \varphi.$$

where the $\{u''_l\}_{l\in L}$ are the Π-instances of $\bigvee_{i\in I} u_i$ in $\mathcal{R}/G$. Furthermore, it follows from **Theorem** in pg. 4 and **Theorem** 3 in Appendix to Lecture 26 (proof in Appendix 1), that we also have the implications:

$$(\ddagger)\;\; \mathcal{N}^{\Pi}_{\mathcal{R}/G}(\bigvee_{i\in I} u_i), \{u'_j\}_{j\in J} \models \varphi \;\Rightarrow\; \mathbb{T}^{\Pi}_{\mathcal{R}/G}, [\![\bigvee_{i\in I} u_i]\!]_{E\cup G\cup B} \models_{LTL} \varphi \;\Rightarrow\; \mathbb{T}^{\Pi}_{\mathcal{R}}, [\![\bigvee_{i\in I} u_i]\!]_{E\cup B} \models_{LTL} \varphi$$

Therefore, from $(\dagger)$ and $(\ddagger)$ if $\mathcal{N}^{\Pi}_{\mathcal{R}}(\bigvee_{i\in I} u_i)$ is deadlock-free we get:

State Space Reduction through Equational Abstractions

Under the assumptions about $\mathcal{R}$ in pg. 2, and those about $\mathcal{R}/G$ in (2) of pg. 5, we are back in the game: $\mathcal{R}/G$ itself satisfies the assumptions in pg. 2. Therefore, for $\varphi \in LTL(\Pi)$ we have (by **Theorem** in pg. 6):

$$(\dagger) \;\; FNF^{\Pi}_{\mathcal{R}/G}(\bigvee_{l \in L} u''_l), \{u''_l\}_{l \in L} \models \varphi \;\; \Rightarrow \;\; \mathcal{N}^{\Pi}_{\mathcal{R}/G}(\bigvee_{i \in I} u_i), \{u'_j\}_{j \in J} \models \varphi.$$

where the $\{u''_l\}_{l \in L}$ are the Π-instances of $\bigvee_{i \in I} u_i$ in $\mathcal{R}/G$. Furthermore, it follows from **Theorem** in pg. 4 and **Theorem** 3 in Appendix to Lecture 26 (proof in Appendix 1), that we also have the implications:

$$(\ddagger) \;\; \mathcal{N}^{\Pi}_{\mathcal{R}/G}(\bigvee_{i \in I} u_i), \{u'_j\}_{j \in J} \models \varphi \;\Rightarrow\; \mathbb{T}^{\Pi}_{\mathcal{R}/G}, [\![\bigvee_{i \in I} u_i]\!]_{E \cup G \cup B} \models_{LTL} \varphi \;\Rightarrow\; \mathbb{T}^{\Pi}_{\mathcal{R}}, [\![\bigvee_{i \in I} u_i]\!]_{E \cup B} \models_{LTL} \varphi$$

Therefore, from $(\dagger)$ and $(\ddagger)$ if $\mathcal{N}^{\Pi}_{\mathcal{R}}(\bigvee_{i \in I} u_i)$ is deadlock-free we get:

Theorem

Under the above assumptions about $\mathcal{R}$ and $\mathcal{R}/G$ the following implication holds:

$$FNF^{\Pi}_{\mathcal{R}/G}(\bigvee_{l \in L} u''_l), \{u''_l\}_{l \in L} \models \varphi \;\; \Rightarrow \;\; \mathbb{T}^{\Pi}_{\mathcal{R}}, [\![\bigvee_{i \in I} u_i]\!]_{E \cup B} \models_{LTL} \varphi.$$

Bounded Narrowing-Based LTL Model Checking

- Construct a depth $\leq k$ under-approximation of the folding narrowing forest (and Kripke structure) $FNF_{\mathcal{R}}^{\Pi}(\bigvee_{j\in J} u'_j)$

Bounded Narrowing-Based LTL Model Checking

- Construct a depth $\leq k$ under-approximation of the folding narrowing forest (and Kripke structure) $FNF^{\Pi}_{\mathcal{R}}(\bigvee_{j \in J} u'_j)$ (a more expensive, but more accurate, version under-approximates $\mathcal{N}^{\Pi}_{\mathcal{R}}(\bigvee_{i \in I} u_i)$).

Bounded Narrowing-Based LTL Model Checking

- Construct a depth $\leq k$ under-approximation of the folding narrowing forest (and Kripke structure) $FNF^{\Pi}_{\mathcal{R}}(\bigvee_{j \in J} u'_j)$ (a more expensive, but more accurate, version under-approximates $\mathcal{N}^{\Pi}_{\mathcal{R}}(\bigvee_{i \in I} u_i)$).

Algorithm: Given a bound n, incrementally build a depth $\leq k$ under-approximation of $FNF^{\Pi}_{\mathcal{R}}(\bigvee_{j \in J} u'_j)$, increasing $k \leq n$ iteratively.

Bounded Narrowing-Based LTL Model Checking

- Construct a depth $\leq k$ under-approximation of the folding narrowing forest (and Kripke structure) $FNF^{\Pi}_{\mathcal{R}}(\bigvee_{j\in J} u'_j)$ (a more expensive, but more accurate, version under-approximates $\mathcal{N}^{\Pi}_{\mathcal{R}}(\bigvee_{i\in I} u_i)$).

Algorithm: Given a bound n, incrementally build a depth $\leq k$ under-approximation of $FNF^{\Pi}_{\mathcal{R}}(\bigvee_{j\in J} u'_j)$, increasing $k \leq n$ iteratively.

1. Apply a standard explicit-state LTL model checking algorithm to verify φ in the depth $\leq k$ under-approximation of $FNF^{\Pi}_{\mathcal{R}}(\bigvee_{j\in J} u'_j)$. If a counterexample is found, stop and return the counterexample.

Bounded Narrowing-Based LTL Model Checking

- Construct a depth $\leq k$ under-approximation of the folding narrowing forest (and Kripke structure) $FNF^{\Pi}_{\mathcal{R}}(\bigvee_{j\in J} u'_j)$ (a more expensive, but more accurate, version under-approximates $\mathcal{N}^{\Pi}_{\mathcal{R}}(\bigvee_{i\in I} u_i)$).

Algorithm: Given a bound n, incrementally build a depth $\leq k$ under-approximation of $FNF^{\Pi}_{\mathcal{R}}(\bigvee_{j\in J} u'_j)$, increasing $k \leq n$ iteratively.

1. Apply a standard explicit-state LTL model checking algorithm to verify φ in the depth $\leq k$ under-approximation of $FNF^{\Pi}_{\mathcal{R}}(\bigvee_{j\in J} u'_j)$. If a counterexample is found, stop and return the counterexample.
2. Suppose that there is no counterexample at depth $\leq k$.

Bounded Narrowing-Based LTL Model Checking

- Construct a depth $\leq k$ under-approximation of the folding narrowing forest (and Kripke structure) $FNF^{\Pi}_{\mathcal{R}}(\bigvee_{j\in J} u'_j)$ (a more expensive, but more accurate, version under-approximates $\mathcal{N}^{\Pi}_{\mathcal{R}}(\bigvee_{i\in I} u_i)$).

Algorithm: Given a bound n, incrementally build a depth $\leq k$ under-approximation of $FNF^{\Pi}_{\mathcal{R}}(\bigvee_{j\in J} u'_j)$, increasing $k \leq n$ iteratively.

1. Apply a standard explicit-state LTL model checking algorithm to verify φ in the depth $\leq k$ under-approximation of $FNF^{\Pi}_{\mathcal{R}}(\bigvee_{j\in J} u'_j)$. If a counterexample is found, stop and return the counterexample.
2. Suppose that there is no counterexample at depth $\leq k$.
   1. If $k = n$, stop and report that the model does not violate φ up to the current bound n.

Bounded Narrowing-Based LTL Model Checking

- Construct a depth $\leq k$ under-approximation of the folding narrowing forest (and Kripke structure) $FNF_{\mathcal{R}}^{\Pi}(\bigvee_{j\in J} u_j')$ (a more expensive, but more accurate, version under-approximates $\mathcal{N}_{\mathcal{R}}^{\Pi}(\bigvee_{i\in I} u_i)$).

Algorithm: Given a bound n, incrementally build a depth $\leq k$ under-approximation of $FNF_{\mathcal{R}}^{\Pi}(\bigvee_{j\in J} u_j')$, increasing $k \leq n$ iteratively.

1. Apply a standard explicit-state LTL model checking algorithm to verify φ in the depth $\leq k$ under-approximation of $FNF_{\mathcal{R}}^{\Pi}(\bigvee_{j\in J} u_j')$. If a counterexample is found, stop and return the counterexample.
2. Suppose that there is no counterexample at depth $\leq k$.
   1. If $k = n$, stop and report that the model does not violate φ up to the current bound n.
   2. Otherwise, generate the depth $\leq k+1$ under-approximation of $FNF_{\mathcal{R}}^{\Pi}(\bigvee_{j\in J} u_j')$

Bounded Narrowing-Based LTL Model Checking

- Construct a depth $\leq k$ under-approximation of the folding narrowing forest (and Kripke structure) $FNF^{\Pi}_{\mathcal{R}}(\bigvee_{j\in J} u'_j)$ (a more expensive, but more accurate, version under-approximates $\mathcal{N}^{\Pi}_{\mathcal{R}}(\bigvee_{i\in I} u_i)$).

Algorithm: Given a bound n, incrementally build a depth $\leq k$ under-approximation of $FNF^{\Pi}_{\mathcal{R}}(\bigvee_{j\in J} u'_j)$, increasing $k \leq n$ iteratively.

1. Apply a standard explicit-state LTL model checking algorithm to verify φ in the depth $\leq k$ under-approximation of $FNF^{\Pi}_{\mathcal{R}}(\bigvee_{j\in J} u'_j)$. If a counterexample is found, stop and return the counterexample.
2. Suppose that there is no counterexample at depth $\leq k$.
    1. If $k = n$, stop and report that the model does not violate φ up to the current bound n.
    2. Otherwise, generate the depth $\leq k+1$ under-approximation of $FNF^{\Pi}_{\mathcal{R}}(\bigvee_{j\in J} u'_j)$
        1. If no new nodes are added to the $\leq k$ under-approximation, $FNF^{\Pi}_{\mathcal{R}}(\bigvee_{j\in J} u'_j)$ has been actually generated! Then return *true*;

Bounded Narrowing-Based LTL Model Checking

- Construct a depth $\leq k$ under-approximation of the folding narrowing forest (and Kripke structure) $FNF_{\mathcal{R}}^{\Pi}(\bigvee_{j \in J} u'_j)$ (a more expensive, but more accurate, version under-approximates $\mathcal{N}_{\mathcal{R}}^{\Pi}(\bigvee_{i \in I} u_i)$).

Algorithm: Given a bound n, incrementally build a depth $\leq k$ under-approximation of $FNF_{\mathcal{R}}^{\Pi}(\bigvee_{j \in J} u'_j)$, increasing $k \leq n$ iteratively.

1. Apply a standard explicit-state LTL model checking algorithm to verify φ in the depth $\leq k$ under-approximation of $FNF_{\mathcal{R}}^{\Pi}(\bigvee_{j \in J} u'_j)$. If a counterexample is found, stop and return the counterexample.
2. Suppose that there is no counterexample at depth $\leq k$.
   1. If $k = n$, stop and report that the model does not violate φ up to the current bound n.
   2. Otherwise, generate the depth $\leq k+1$ under-approximation of $FNF_{\mathcal{R}}^{\Pi}(\bigvee_{j \in J} u'_j)$
      1. If no new nodes are added to the $\leq k$ under-approximation, $FNF_{\mathcal{R}}^{\Pi}(\bigvee_{j \in J} u'_j)$ has been actually generated! Then return *true*;
      2. Otherwise, go to Step 1 with the depth $\leq k+1$ under-approximation of $FNF_{\mathcal{R}}^{\Pi}(\bigvee_{j \in J} u'_j)$.

Maude's Logical LTL Model Checker Tool

Maude's Logical LTL Model Checker supports symbolic LTL model checking just explained.

Maude's Logical LTL Model Checker Tool

Maude's Logical LTL Model Checker supports symbolic LTL model checking just explained. This is a new implementation, not that in the CS 476 web page.

Maude's Logical LTL Model Checker Tool

Maude's Logical LTL Model Checker supports symbolic LTL model checking just explained. This is a new implementation, not that in the CS 476 web page. A `README` overview can be found here:

Maude's Logical LTL Model Checker Tool

Maude's Logical LTL Model Checker supports symbolic LTL model checking just explained. This is a new implementation, not that in the CS 476 web page. A `README` overview can be found here:

`https://github.com/kquine/maude-model-checker/blob/master/README-lmc.md`

Maude's Logical LTL Model Checker Tool

Maude's Logical LTL Model Checker supports symbolic LTL model checking just explained. This is a new implementation, not that in the CS 476 web page. A `README` overview can be found here:

`https://github.com/kquine/maude-model-checker/blob/master/README-lmc.md`

It uses a a special version of Maude that extends Maude 3.3.1. Executables for both Linux and MacOS and a folder `symbolic-examples` can be found here:

Maude's Logical LTL Model Checker Tool

Maude's Logical LTL Model Checker supports symbolic LTL model checking just explained. This is a new implementation, not that in the CS 476 web page. A `README` overview can be found here:

`https://github.com/kquine/maude-model-checker/blob/master/README-lmc.md`

It uses a a special version of Maude that extends Maude 3.3.1. Executables for both Linux and MacOS and a folder `symbolic-examples` can be found here:

`https://github.com/kquine/maude-model-checker/releases/tag/v3.3.1-ltlr-lmc`

Maude's Logical LTL Model Checker Tool

Maude's Logical LTL Model Checker supports symbolic LTL model checking just explained. This is a new implementation, not that in the CS 476 web page. A `README` overview can be found here:

`https://github.com/kquine/maude-model-checker/blob/master/README-lmc.md`

It uses a a special version of Maude that extends Maude 3.3.1. Executables for both Linux and MacOS and a folder `symbolic-examples` can be found here:

`https://github.com/kquine/maude-model-checker/releases/tag/v3.3.1-ltlr-lmc`

As explained in the `README` overview, the user:

Maude's Logical LTL Model Checker Tool

Maude's Logical LTL Model Checker supports symbolic LTL model checking just explained. This is a new implementation, not that in the CS 476 web page. A `README` overview can be found here:

`https://github.com/kquine/maude-model-checker/blob/master/README-lmc.md`

It uses a a special version of Maude that extends Maude 3.3.1. Executables for both Linux and MacOS and a folder `symbolic-examples` can be found here:

`https://github.com/kquine/maude-model-checker/releases/tag/v3.3.1-ltlr-lmc`

As explained in the `README` overview, the user:

1. Enters into this special version of Maude a user module `M`.

Maude's Logical LTL Model Checker Tool

Maude's Logical LTL Model Checker supports symbolic LTL model checking just explained. This is a new implementation, not that in the CS 476 web page. A `README` overview can be found here:

`https://github.com/kquine/maude-model-checker/blob/master/README-lmc.md`

It uses a a special version of Maude that extends Maude 3.3.1. Executables for both Linux and MacOS and a folder `symbolic-examples` can be found here:

`https://github.com/kquine/maude-model-checker/releases/tag/v3.3.1-ltlr-lmc`

As explained in the `README` overview, the user:

1. Enters into this special version of Maude a user module `M`.
2. Then gives the command `load symbolic-checker`.

Maude's Logical LTL Model Checker Tool

Maude's Logical LTL Model Checker supports symbolic LTL model checking just explained. This is a new implementation, not that in the CS 476 web page. A `README` overview can be found here:

`https://github.com/kquine/maude-model-checker/blob/master/README-lmc.md`

It uses a a special version of Maude that extends Maude 3.3.1. Executables for both Linux and MacOS and a folder `symbolic-examples` can be found here:

`https://github.com/kquine/maude-model-checker/releases/tag/v3.3.1-ltlr-lmc`

As explained in the `README` overview, the user:

1. Enters into this special version of Maude a user module `M`.
2. Then gives the command `load symbolic-checker`. The user then enters enclosed in parentheses the user module `M-CHECK` defining:

Maude's Logical LTL Model Checker Tool

Maude's Logical LTL Model Checker supports symbolic LTL model checking just explained. This is a new implementation, not that in the CS 476 web page. A `README` overview can be found here:

`https://github.com/kquine/maude-model-checker/blob/master/README-lmc.md`

It uses a a special version of Maude that extends Maude 3.3.1. Executables for both Linux and MacOS and a folder `symbolic-examples` can be found here:

`https://github.com/kquine/maude-model-checker/releases/tag/v3.3.1-ltlr-lmc`

As explained in the `README` overview, the user:

1. Enters into this special version of Maude a user module `M`.
2. Then gives the command `load symbolic-checker`. The user then enters enclosed in parentheses the user module `M-CHECK` defining:
   - the equational definition of state predicates just as for Maude's LTL model checker, but giving to all equations the `[variant]` attribute.

Maude's Logical LTL Model Checker Tool

Maude's Logical LTL Model Checker supports symbolic LTL model checking just explained. This is a new implementation, not that in the CS 476 web page. A `README` overview can be found here:

`https://github.com/kquine/maude-model-checker/blob/master/README-lmc.md`

It uses a a special version of Maude that extends Maude 3.3.1. Executables for both Linux and MacOS and a folder `symbolic-examples` can be found here:

`https://github.com/kquine/maude-model-checker/releases/tag/v3.3.1-ltlr-lmc`

As explained in the `README` overview, the user:

1. Enters into this special version of Maude a user module `M`.
2. Then gives the command `load symbolic-checker`. The user then enters enclosed in parentheses the user module `M-CHECK` defining:
   - the equational definition of state predicates just as for Maude's LTL model checker, but giving to all equations the `[variant]` attribute.
   - a subsort inclusion `User-State < State`

Maude's Logical LTL Model Checker Tool

Maude's Logical LTL Model Checker supports symbolic LTL model checking just explained. This is a new implementation, not that in the CS 476 web page. A `README` overview can be found here:

`https://github.com/kquine/maude-model-checker/blob/master/README-lmc.md`

It uses a a special version of Maude that extends Maude 3.3.1. Executables for both Linux and MacOS and a folder `symbolic-examples` can be found here:

`https://github.com/kquine/maude-model-checker/releases/tag/v3.3.1-ltlr-lmc`

As explained in the `README` overview, the user:

1. Enters into this special version of Maude a user module `M`.
2. Then gives the command `load symbolic-checker`. The user then enters enclosed in parentheses the user module `M-CHECK` defining:
   - the equational definition of state predicates just as for Maude's LTL model checker, but giving to all equations the `[variant]` attribute.
   - a subsort inclusion `User-State < State`
   - imports `M` and `SYMBOLIC-CHECKER` as submodules.

Maude's Logical LTL Model Checker Tool

Maude's Logical LTL Model Checker supports symbolic LTL model checking just explained. This is a new implementation, not that in the CS 476 web page. A `README` overview can be found here:

`https://github.com/kquine/maude-model-checker/blob/master/README-lmc.md`

It uses a a special version of Maude that extends Maude 3.3.1. Executables for both Linux and MacOS and a folder `symbolic-examples` can be found here:

`https://github.com/kquine/maude-model-checker/releases/tag/v3.3.1-ltlr-lmc`

As explained in the `README` overview, the user:

1. Enters into this special version of Maude a user module `M`.
2. Then gives the command `load symbolic-checker`. The user then enters enclosed in parentheses the user module `M-CHECK` defining:
   - the equational definition of state predicates just as for Maude's LTL model checker, but giving to all equations the `[variant]` attribute.
   - a subsort inclusion `User-State < State`
   - imports `M` and `SYMBOLIC-CHECKER` as submodules.
3. Then one can give symbolic model checking commands to the tool.

Maude's Logical LTL Model Checker Tool

Maude's Logical LTL Model Checker supports symbolic LTL model checking just explained. This is a new implementation, not that in the CS 476 web page. A `README` overview can be found here:

`https://github.com/kquine/maude-model-checker/blob/master/README-lmc.md`

It uses a a special version of Maude that extends Maude 3.3.1. Executables for both Linux and MacOS and a folder `symbolic-examples` can be found here:

`https://github.com/kquine/maude-model-checker/releases/tag/v3.3.1-ltlr-lmc`

As explained in the `README` overview, the user:

1. Enters into this special version of Maude a user module `M`.
2. Then gives the command `load symbolic-checker`. The user then enters enclosed in parentheses the user module `M-CHECK` defining:
   - the equational definition of state predicates just as for Maude's LTL model checker, but giving to all equations the `[variant]` attribute.
   - a subsort inclusion `User-State < State`
   - imports `M` and `SYMBOLIC-CHECKER` as submodules.
3. Then one can give symbolic model checking commands to the tool.

Let us illustrate everything with two examples.

Symbolic LTL Model Checking: a `R&W` Example

This special version of Maude supports the LTL symbolic model checker:

Symbolic LTL Model Checking: a R&W Example

This special version of Maude supports the LTL symbolic model checker:

```
meseguer@CS-MESEGUER-MBA LTL-LMC-11-23 % ./maude-ltlr-lmc.darwin64
     \||||||||||||||||||/
   --- Welcome to Maude ---
     /||||||||||||||||||\
     Maude 3.3.1 built: Nov 22 2023 21:46:36
     Copyright 1997-2023 SRI International
   Sat Nov 25 20:42:15 2023
Maude>
```

Symbolic LTL Model Checking: a R&W Example

This special version of Maude supports the LTL symbolic model checker:

```
meseguer@CS-MESEGUER-MBA LTL-LMC-11-23 % ./maude-ltlr-lmc.darwin64
     \||||||||||||||||||/
   --- Welcome to Maude ---
     /||||||||||||||||||\
     Maude 3.3.1 built: Nov 22 2023 21:46:36
     Copyright 1997-2023 SRI International
   Sat Nov 25 20:42:15 2023
Maude>
```

We then load the module of interest, here R&W:

Symbolic LTL Model Checking: a R&W Example

This special version of Maude supports the LTL symbolic model checker:

```
meseguer@CS-MESEGUER-MBA LTL-LMC-11-23 % ./maude-ltlr-lmc.darwin64
     \||||||||||||||||||/
   --- Welcome to Maude ---
     /||||||||||||||||||\
     Maude 3.3.1 built: Nov 22 2023 21:46:36
     Copyright 1997-2023 SRI International
   Sat Nov 25 20:42:15 2023
Maude>
```

We then load the module of interest, here R&W:

```
mod R&W is
  sort Natural .
  op 0 : -> Natural [ctor] .
  op s : Natural -> Natural [ctor] .
  sort Config .
  op <_,_> : Natural Natural -> Config [ctor] .

  vars R W : Natural .

  rl [enter-w] : < 0, 0 > => < 0, s(0) > [narrowing] .
  rl [leave-w] : < R, s(W) > => < R, W > [narrowing] .
  rl [enter-r] : < R, 0 > => < s(R), 0 > [narrowing] .
  rl [leave-r] : < s(R), W > => < R, W > [narrowing] .
endm
```

Symbolic LTL Model Checking: a `R&W` Example (II)

We then load the symbolic LTL model checker and enter the `R&W-CHECK` module enclosed in parentheses:

Symbolic LTL Model Checking: a R&W Example (II)

We then load the symbolic LTL model checker and enter the R&W-CHECK module enclosed in parentheses:

```
load symbolic-checker

(mod R&W-CHECK is
  protecting R&W .
  including SYMBOLIC-CHECKER .

  subsort Config < State .

  vars N M : Natural .

  op reads : -> Prop .
  eq < s(N), M > |= reads = true [variant] .
  eq < 0,    M > |= reads = false [variant] .

  op writes : -> Prop .
  eq < M, s(N) > |= writes = true [variant] .
  eq < M,    0 > |= writes = false [variant] .

  op writers>1 : -> Prop .
  eq < M, s(s(N)) > |= writers>1 = true [variant] .
   eq < M, s(0)    > |= writers>1 = false [variant] .
  eq < M,     0    > |= writers>1 = false [variant] .
endm)
```

Symbolic LTL Model Checking: a `R&W` Example (III)

We can now give symbolic model checking commands enclosed in parentheses. The `lmc` commands from the symbolic initial state `< N,0 >` to verify mutex and one-writer invariants do not terminate, but we can model check check them up to, e.g., bound 100:

Symbolic LTL Model Checking: a R&W Example (III)

We can now give symbolic model checking commands enclosed in parentheses. The `lmc` commands from the symbolic initial state `< N,0 >` to verify mutex and one-writer invariants do not terminate, but we can model check check them up to, e.g., bound 100:

```
Maude> (lmc [100] < N, 0 > |= [] ~ (reads /\ writes) .)

result: no counterexample found within bound 100

Maude> (lmc [100] < N, 0 > |= [] ~ (writers>1) .)

result: no counterexample found within bound 100
```

Symbolic LTL Model Checking: a R&W Example (III)

We can now give symbolic model checking commands enclosed in parentheses. The lmc commands from the symbolic initial state < N,0 > to verify mutex and one-writer invariants do not terminate, but we can model check check them up to, e.g., bound 100:

```
Maude> (lmc [100] < N, 0 > |= [] ~ (reads /\ writes) .)

result: no counterexample found within bound 100

Maude> (lmc [100] < N, 0 > |= [] ~ (writers>1) .)

result: no counterexample found within bound 100
```

However, the folding lfmc commands terminate proving the invariants:

Symbolic LTL Model Checking: a R&W Example (III)

We can now give symbolic model checking commands enclosed in parentheses. The lmc commands from the symbolic initial state < N,0 > to verify mutex and one-writer invariants do not terminate, but we can model check check them up to, e.g., bound 100:

```
Maude> (lmc [100] < N, 0 > |= [] ~ (reads /\ writes) .)

result: no counterexample found within bound 100

Maude> (lmc [100] < N, 0 > |= [] ~ (writers>1) .)

result: no counterexample found within bound 100
```

However, the folding lfmc commands terminate proving the invariants:

```
Maude> (lfmc < N, 0 > |= [] ~ (reads /\ writes) .)

result: true (complete with depth 3)

Maude> (lfmc < N, 0 > |= [] ~ (writers>1) .)

result: true (complete with depth 3)
```

Symbolic LTL Model Checking: a `R&W` Example (IV)

Likewise, we can prove (or disprove) some non-starvation properties:

Symbolic LTL Model Checking: a R&W Example (IV)

Likewise, we can prove (or disprove) some non-starvation properties:

```
Maude> (lmc < N, 0 > |= []<> reads .)

result: counterexample found at depth 4

prefix
  {< 0,0 >,none,'enter-w}
loop
  {< 0,s(0)>,none,'leave-w}
  {< 0,0 >,none,'enter-w}

Maude> (lmc < N, 0 > |= []<> writes .)

result: counterexample found at depth 3

prefix
  {< N:Natural,0 >,'N <- s(%1:Natural),'leave-r}
loop
  {< N:Natural,0 >,'N <- s(%1:Natural),'leave-r}

Maude> (lfmc < N, 0 > |= []<> (reads \/ writes) .)

result: true
```

Symbolic LTL Model Checking: a `BAKERY` Example

The following `BAKERY` version is harder to verify than that in Lecture 21:

Symbolic LTL Model Checking: a BAKERY Example

The following BAKERY version is harder to verify than that in Lecture 21:

```
fmod BAKERY-SYNTAX is
  sort Name .
  op 0 : -> Name [ctor] .
  op s : -> Name [ctor] .
  op __ : Name Name -> Name [ctor comm assoc id: 0] .

  sorts ModeIdle ModeWait ModeCrit Mode Conf .
  subsorts ModeIdle ModeWait ModeCrit < Mode .
  sorts ProcIdle ProcWait Proc ProcIdleSet ProcWaitSet ProcSet .
  subsorts ProcIdle < ProcIdleSet .
  subsorts ProcWait < ProcWaitSet .
  subsorts ProcIdle ProcWait < Proc < ProcSet .
  subsorts ProcIdleSet < ProcWaitSet < ProcSet .

  op idle : -> ModeIdle .
  op wait : Name -> ModeWait .
  op crit : Name -> ModeCrit .
  op [_] : ModeIdle -> ProcIdle .
  op [_] : ModeWait -> ProcWait .
  op [_] : Mode -> Proc .
  op none : -> ProcIdleSet .
  op __ : ProcIdleSet ProcIdleSet -> ProcIdleSet [assoc comm ] .
  op __ : ProcWaitSet ProcWaitSet -> ProcWaitSet [assoc comm ] .
  op __ : ProcSet ProcSet -> ProcSet [assoc comm ] .

  op _;_;_ : Name Name ProcSet -> Conf .
endfm
```

Symbolic LTL Model Checking: a BAKERY Example (II)

```
mod BAKERY is
  protecting BAKERY-SYNTAX .

  var PS : ProcSet .  vars N M : Name .

  rl [wake] : N ; M ; [idle] PS    => s N ; M ; [wait(N)] PS [narrowing] .
  rl [crit] : N ; M ; [wait(M)] PS => N ; M ; [crit(M)] PS [narrowing] .
  rl [exit] : N ; M ; [crit(M)] PS => N ; s M ; [idle] PS [narrowing] .
endm

load symbolic-checker

(mod BAKERY-CHECK1 is
  pr BAKERY .
  including SYMBOLIC-CHECKER .

  subsort Conf < State .

  ops was-wait? was-crit? : -> Prop . *** was or is in wait (resp. crit)

  vars N M : Name . vars PS : ProcSet .

  eq s N ; M ; PS |= was-wait? = true [variant] .
  eq 0 ; M ; PS    |= was-wait? = false [variant] .
  eq N ; s M ; PS |= was-crit? = true [variant] .
  eq N ; 0 ; PS    |= was-crit? = false [variant] .
endm)
```

Symbolic LTL Model Checking: a BAKERY Example (III)

Does having been waiting always lead to some process being in the critical section?

Symbolic LTL Model Checking: a BAKERY Example (III)

Does having been waiting always lead to some process being in the critical section?

```
(lfmc N ; N ; [idle] [idle] |= [] (was-wait? -> <> was-crit?) .)

result: true (complete with depth 5)

(lfmc N ; M ; IS:ProcIdleSet |= [] (was-wait? -> <> was-crit?) .)

result: counterexample found at depth 5  *** deadlock counterexample

prefix
  {(s #1:Name); 0 ; IS:ProcIdleSet,'IS <- %1:ProcIdleSet[idle],'wake}
  {(s s %2:Name); 0 ; %1:ProcIdleSet[wait(s %2:Name)],'%1 <-[idle],'wake}
loop
  {(s s s %2:Name); 0 ;[wait(s %2:Name)][wait(s s %2:Name)],none,deadlock}

(lfmc N ; M ; WS:ProcWaitSet |= [] (was-wait? -> <> was-crit?) .)

result: counterexample found at depth 3  *** non-deadlock counterexample

prefix
  {(s #1:Name); 0 ; WS:ProcWaitSet,'WS <- %1:ProcWaitSet[idle],'wake}
loop
  {(s #1:Name); 0 ; WS:ProcWaitSet,'WS <- %1:ProcWaitSet[idle],'wake}
```

Symbolic LTL Model Checking: a BAKERY Example (IV)

Does mutual exclusion hold?

Symbolic LTL Model Checking: a BAKERY Example (IV)

Does mutual exclusion hold?

```
(mod BAKERY-CHECK2 is  pr BAKERY .  including SYMBOLIC-CHECKER .
  subsort Conf < State .

  ops mutex : -> Prop .

  var WS : ProcWaitSet . var IS : ProcIdleSet . var PS : ProcSet .
  vars N M M1 M2 : Name .

  eq N ; M ; WS |= mutex = true [variant] .
  eq N ; M ; [crit(M1)] WS |= mutex = true [variant] .
  eq N ; M ; [crit(M1)] [crit(M2)] PS |= mutex = false [variant] .
endm)

(lmc [100] N:Name ; N:Name ; [idle] [idle] |= [] mutex .)

result: no counterexample found within bound 100

(lfmc N:Name ; N:Name ; [idle] [idle] |= [] mutex .)

result: true (complete with depth 5)
```

Symbolic LTL Model Checking: a BAKERY Example (V)

```
(lfmc N ; M ; WS |= [] mutex  .)

result: counterexample found at depth 5

prefix
{N:Name ; M:Name ; WS:ProcWaitSet,'WS <- %1:ProcWaitSet[wait(M:Name)],'crit}
{N:Name ; M:Name ; %1:ProcWaitSet[crit(M:Name)],'%1 <- %3:ProcWaitSet[wait(M:Name)],'crit}
{N:Name ; M:Name ; %3:ProcWaitSet[crit(M:Name)][crit(M:Name)],'%3 <-[wait(M:Name)],'crit}
loop
  nil

(lfmc N ; N ; WS |= [] mutex  .)

result: counterexample found at depth 5

prefix
{N:Name ; N:Name ; WS:ProcWaitSet,'WS <- %1:ProcWaitSet[wait(N:Name)],'crit}
{N:Name ; N:Name ; %1:ProcWaitSet[crit(N:Name)],'%1 <- %2:ProcWaitSet[wait(N:Name)],'crit}
{N:Name ; N:Name ; %2:ProcWaitSet[crit(N:Name)][crit(N:Name)],'%2 <-[wait(N:Name)],'crit}
loop
  nil

(lfmc [100] N ; N ; IS |= [] mutex  .)

result: no counterexample found within bound 100
```