LTL MODEL CHECKING
WE NEED TO LEARN A BIT ABOUT AUTOMATA ON INFINITE WORDS ...
**LTL Syntax/Semantics**

\[
\varphi ::= \text{true} \mid a \mid \varphi_1 \land \varphi_2 \mid \neg \varphi \mid \Box \varphi \mid \varphi_1 \mathbf{U} \varphi_2
\]

- \( \sigma \models \text{true} \) iff \( \sigma \models a \) (i.e., \( A_0 \models a \))
- \( \sigma \models a \) iff \( a \in A_0 \) and \( \sigma \models a \)
- \( \sigma \models \varphi_1 \land \varphi_2 \) iff \( \sigma \models \varphi_1 \) and \( \sigma \models \varphi_2 \)
- \( \sigma \models \neg \varphi \) iff \( \sigma \not\models \varphi \)
- \( \sigma \models \Box \varphi \) iff \( \sigma[1...] = A_1A_2A_3... \models \varphi \)
- \( \sigma \models \varphi_1 \mathbf{U} \varphi_2 \) iff \( \exists j \geq 0. \sigma[j...] \models \varphi_2 \) and \( \sigma[i...] \models \varphi_1 \), for all \( 0 \leq i < j \)
We will be using **positive normal form (PNF):** negations pushed down to the level of atomic propositions.
We will be using positive normal form (PNF): negations pushed down to the level of atomic propositions.

The only problem is with the until operator:

\[
\neg(\varphi \mathbin{U} \psi) \equiv ((\varphi \land \neg \psi) \mathbin{U} (\neg \varphi \land \neg \psi)) \lor \Box(\varphi \land \neg \psi)
\]
We will be using positive normal form (PNF): negations pushed down to the level of atomic propositions.

The only problem is with the until operator:

$$\neg(\varphi \mathsf{U} \psi) \equiv ((\varphi \land \neg \psi) \mathsf{U} (\neg \varphi \land \neg \psi)) \lor \square(\varphi \land \neg \psi)$$

We introduce the release operator to solve this:

$$\varphi \mathsf{R} \psi \overset{\text{def}}{=} \neg(\neg \varphi \mathsf{U} \neg \psi)$$
Release

We will be using positive normal form (PNF): negations pushed down to the level of atomic propositions.

The only problem is with the until operator:

\[ \neg (\varphi U \psi) \equiv ((\varphi \land \neg \psi) U (\neg \varphi \land \neg \psi)) \lor \Box (\varphi \land \neg \psi) \]

We introduce the release operator to solve this:

\[ \varphi R \psi \overset{\text{def}}{=} \neg (\neg \varphi U \neg \psi) \]

Release has its own expansion law:

\[ \varphi R \psi \equiv \psi \land (\varphi \lor \bigcirc (\varphi R \psi)) \]
RELEASE

We will be using positive normal form (PNF): negations pushed down to the level of atomic propositions.

The only problem is with the until operator:

$$\neg(\varphi U \psi) \equiv ((\varphi \land \neg \psi) U (\neg \varphi \land \neg \psi)) \lor \Box(\varphi \land \neg \psi)$$

We introduce the release operator to solve this:

$$\varphi R \psi \overset{\text{def}}{=} \neg(\neg \varphi U \neg \psi)$$

Release has its own expansion law:

$$\varphi R \psi \equiv \psi \land (\varphi \lor \bigcirc (\varphi R \psi))$$

Alternative notation that we use later: $$\varphi R \psi = \varphi \tilde{U} \psi$$
STEP 3: MODEL CHECKING AGAINST AN LTL PROPERTY
System Verification

deriving algorithms and data structures, together with the availability of faster computers and larger computer memories, model-based techniques that a decade ago only worked for very simple examples are nowadays applicable to realistic designs. As the starting point of these techniques is a model of the system under consideration, we have as a given fact that any verification using model-based techniques is only as good as the model of the system.

Model checking is a verification technique that explores all possible system states in a brute-force manner. Similar to a computer chess program that checks possible moves, a model checker, the software tool that performs the model checking, examines all possible system scenarios in a systematic manner. In this way, it can be shown that a given system model truly satisfies a certain property. It is a real challenge to examine the largest possible state spaces that can be treated with current means, i.e., processors and memories. State-of-the-art model checkers can handle state spaces of about $10^{8}$ to $10^{9}$ states with explicit state-space enumeration. Using clever algorithms and tailored data structures, larger state spaces ($10^{20}$ up to even $10^{476}$ states) can be handled for specific problems. Even the subtle errors that remain undiscovered using emulation, testing and simulation can potentially be revealed using model checking.

Figure 1.4: Schematic view of the model-checking approach.

Typical properties that can be checked using model checking are of a qualitative nature: Is the generated result OK?, Can the system reach a deadlock situation, e.g., when two
\[ A = (Q, \Sigma, \delta, Q_0, \mathcal{F}) \]

\[ \mathcal{F} = \{ F_1, \ldots, F_n \} \]

A run \( \sigma \) is accepting iff \( \forall i : inf(\sigma) \cap F_i \neq \emptyset \)
**LTL Model Checking**

**Definition.** LTL model checking is a decision problem that given a finite LTS $T$ and an LTL formula $\phi$ returns YES if $T \models \phi$, and NO together with a counterexample trace, otherwise.

So that we can do:

$$TS \models \phi \iff \text{paths}(TS) \subseteq L_\phi$$

$$\iff \text{paths}(TS) \cap \overline{L_\phi} = \emptyset$$

$$\iff \text{paths}(TS) \cap L_{\neg \phi} = \emptyset$$
Definition. LTL model checking is a decision problem that given a finite LTS $T$ and an LTL formula $\phi$ returns YES if $T \models \phi$, and NO together with a counterexample trace, otherwise.

To use automata, let:

$$\Sigma = 2^{AP}$$

$$L_{\phi} = \{ \pi \in \Sigma^\omega | \pi \models \phi \}$$

So that we can do:

$$TS \models \phi \iff \text{paths}(TS) \subseteq L_{\phi}$$

$$\iff \text{paths}(TS) \cap \overline{L_{\phi}} = \emptyset$$

$$\iff \text{paths}(TS) \cap L_{\neg \phi} = \emptyset$$
EXAMPLES

How can an LTL formula be represented as an NBA?

How can this NBA be algorithmically computed?
How can an LTL formula be represented as an NBA?

How can this NBA be algorithmically computed?

□◊ green
EXAMPLES

How can an LTL formula be represented as an NBA?

How can this NBA be algorithmically computed?

□◊ green
EXAMPLES

How can an LTL formula be represented as an NBA?

How can this NBA be algorithmically computed?

□◊green

□(a → ◊b)
EXAM P L E S

How can an LTL formula be represented as an NBA?

How can this NBA be algorithmically computed?

□◊green

□(a → ◊b)
LTL TO GNBA
Let's consider a path in a TS:

$$\pi = \pi_0\pi_1 \ldots$$

$$\forall i : L(\pi_i) \subseteq AP$$
Let’s consider a path in a TS:

\[ \pi = \pi_0 \pi_1 \ldots \]

\[ \forall i : L(\pi_i) \subseteq AP \]

Consider an LTL property \( \varphi \):

\[ cl(\varphi) : \text{all the relevant sub-formulas of } \varphi \]
Let's consider a path in a TS:
\[ \pi = \pi_0 \pi_1 \ldots \]
\[ \forall i : L(\pi_i) \subseteq AP \]

Consider an LTL property \( \varphi \):

\[ cl(\varphi) : \text{all the relevant sub-formulas of } \varphi \]

Example:
\[ \varphi = a U (\neg a \land b) \]
CONSTRUCTION IDEA

Let’s consider a path in a TS:

\[ \pi = \pi_0 \pi_1 \ldots \]

\[ \forall i : L(\pi_i) \subseteq AP \]

Consider an LTL property \( \varphi \):

\[ cl(\varphi) : \text{all the relevant sub-formulas of } \varphi \]

Example:

\[ \varphi = a U (\neg a \land b) \]

\[ a, b, \neg a, \neg b, \neg a \land b, \neg(\neg a \land b), \varphi, \neg \varphi \]
CONSTRUCTION IDEA

Let's consider a path in a TS:

$$\pi = \pi_0 \pi_1 \ldots$$

$$\forall i : \ L(\pi_i) \subseteq AP$$

Consider an LTL property $\varphi$:

$$cl(\varphi) : \text{all the relevant sub-formulas of } \varphi$$

Example: We lift the path from AP to the set of all sub-formulas that hold.

$$\sigma = \{ a \} \{ a, b \} \{ b \} \ldots.$$
CONSTRUCTION IDEA

Let’s consider a path in a TS:

\[ \pi = \pi_0\pi_1\ldots \]

\[ \forall i : L(\pi_i) \subseteq AP \]

Consider an LTL property \( \varphi \):

\( cl(\varphi) : \) all the relevant sub-formulas of \( \varphi \)

Example: We lift the path from AP to the set of all sub-formulas that hold.

\[ \sigma = \{a\} \{a, b\} \{b\} \ldots . \]

\[ \{a, \neg b, \neg(\neg a \land b), \phi\} \]
CONSTRUCTION IDEA

Let’s consider a path in a TS:  

\[ \pi = \pi_0 \pi_1 \ldots \]

\[ \forall i : L(\pi_i) \subseteq AP \]

Consider an LTL property \( \varphi \):

\[ cl(\varphi) : \text{all the relevant sub-formulas of } \varphi \]

Example:  We lift the path from AP to the set of all sub-formulas that hold.

\[ \sigma = \{ a \} \{ a, b \} \{ b \} \ldots . \]

\[ \{ a, \neg b, \neg (\neg a \land b), \phi \} \quad \{ a, b, \neg (\neg a \land b), \phi \} \]
CONSTRUCTION IDEA

Let’s consider a path in a TS:

\[ \pi = \pi_0 \pi_1 \ldots \]

\[ \forall i : L(\pi_i) \subseteq AP \]

Consider an LTL property \( \varphi \):

\[ cl(\varphi) : \text{all the relevant sub-formulas of } \varphi \]

Example: We lift the path from AP to the set of all sub-formulas that hold.

\[ \sigma = \{ a \} \{ a, b \} \{ b \} \ldots \]

\[ \{ a, \neg b, \neg (\neg a \land b), \phi \} \quad \{ a, b, \neg (\neg a \land b), \phi \} \quad \{ \neg a, b, (\neg a \land b), \phi \} \]
CONSTRUCTION

First, we put $\varphi$ into positive normal form, so that negation is only applied at the level of individual propositions.
CONSTRUCTION

First, we put $\varphi$ into positive normal form, so that negation is only applied at the level of individual propositions.

Let’s formally define $\text{cl}(\varphi)$. The smallest set satisfying:

- $\varphi \in \text{cl}(\varphi)$,
- $\varphi_1 \land \varphi_2 \in \text{cl}(\varphi) \Rightarrow \varphi_1, \varphi_2 \in \text{cl}(\varphi)$,
- $\varphi_1 \lor \varphi_2 \in \text{cl}(\varphi) \Rightarrow \varphi_1, \varphi_2 \in \text{cl}(\varphi)$,
- $\bigcirc \varphi_1 \in \text{cl}(\varphi) \Rightarrow \varphi_1 \in \text{cl}(\varphi)$,
- $\varphi_1 \lor \varphi_2 \in \text{cl}(\varphi) \Rightarrow \varphi_1, \varphi_2 \in \text{cl}(\varphi)$,
- $\varphi_1 \lor \varphi_2 \in \text{cl}(\varphi) \Rightarrow \varphi_1, \varphi_2 \in \text{cl}(\varphi)$.
CONSTRUCTION

Let’s define the labelling rule.
Let's define the labelling rule.

For every position $i \in \mathbb{N}$, the labeling $\tau : \mathbb{N} \to 2^{\text{cl}(\varphi)}$ satisfies:
Let's define the labelling rule.

For every position $i \in \mathbb{N}$, the labeling $\tau : \mathbb{N} \to 2^{cl(\varphi)}$ satisfies:

$$\forall p \in AP : (p \in \tau(i) \implies p \in L(\pi_i)) \land (\neg p \in \tau(i) \implies p \notin L(\pi_i))$$
Let's define the labelling rule.

For every position $i \in \mathbb{N}$, the labeling $\tau : \mathbb{N} \rightarrow 2^{cl(\varphi)}$ satisfies:

- $\forall p \in AP : (p \in \tau(i) \implies p \in L(\pi_i)) \land (\neg p \in \tau(i) \implies p \notin L(\pi_i))$
- $\phi_1 \land \phi_2 \in \tau(i) \implies (\phi_1 \in \tau(i) \land \phi_2 \in \tau(i))$
Let's define the labelling rule.

For every position $i \in \mathbb{N}$, the labeling $\tau : \mathbb{N} \rightarrow 2^{cl(\varphi)}$ satisfies:

- $\forall p \in AP: \ (p \in \tau(i) \implies p \in L(\pi_i)) \land (\neg p \in \tau(i) \implies p \notin L(\pi_i))$
- $\phi_1 \land \phi_2 \in \tau(i) \implies (\phi_1 \in \tau(i) \land \phi_2 \in \tau(i))$
- $\phi_1 \lor \phi_2 \in \tau(i) \implies (\phi_1 \in \tau(i) \lor \phi_2 \in \tau(i))$
Let’s define the labelling rule.

For every position $i \in \mathbb{N}$, the labeling $\tau : \mathbb{N} \rightarrow 2^{\text{cl}(\varphi)}$ satisfies:

- $\forall p \in AP : (p \in \tau(i) \implies p \in L(\pi_i)) \land (\neg p \in \tau(i) \implies p \notin L(\pi_i))$
- $\phi_1 \land \phi_2 \in \tau(i) \implies (\phi_1 \in \tau(i) \land \phi_2 \in \tau(i))$
- $\phi_1 \lor \phi_2 \in \tau(i) \implies (\phi_1 \in \tau(i) \lor \phi_2 \in \tau(i))$
- $\Box \phi \in \tau(i) \implies \phi \in \tau(i + 1)$
Let’s define the labelling rule.

For every position $i \in \mathbb{N}$, the labeling $\tau : \mathbb{N} \rightarrow 2^{cl(\varphi)}$ satisfies:

- $\forall p \in AP : (p \in \tau(i) \implies p \in L(\pi_i)) \land (\neg p \in \tau(i) \implies p \notin L(\pi_i))$
- $\phi_1 \land \phi_2 \in \tau(i) \implies (\phi_1 \in \tau(i) \land \phi_2 \in \tau(i))$
- $\phi_1 \lor \phi_2 \in \tau(i) \implies (\phi_1 \in \tau(i) \lor \phi_2 \in \tau(i))$
- $\Box \phi \in \tau(i) \implies \phi \in \tau(i+1)$
- $\phi_1 U \phi_2 \in \tau(i) \implies (\phi_2 \in \tau(i) \lor (\phi_1 \in \tau(i) \land \phi_1 U \phi_2 \in \tau(i)))$
Let's define the labelling rule.

For every position $i \in \mathbb{N}$, the labeling $\tau : \mathbb{N} \rightarrow 2^{cl(\varphi)}$ satisfies:

- $\forall p \in AP : (p \in \tau(i) \implies p \in L(\pi_i)) \land (\neg p \in \tau(i) \implies p \notin L(\pi_i))$
- $\phi_1 \land \phi_2 \in \tau(i) \implies (\phi_1 \in \tau(i) \land \phi_2 \in \tau(i))$
- $\phi_1 \lor \phi_2 \in \tau(i) \implies (\phi_1 \in \tau(i) \lor \phi_2 \in \tau(i))$
- $\bigcirc \phi \in \tau(i) \implies \phi \in \tau(i + 1)$
- $\phi_1 U \phi_2 \in \tau(i) \implies (\phi_2 \in \tau(i) \lor (\phi_1 \in \tau(i) \land \phi_1 U \phi_2 \in \tau(i))$
- $\phi_1 \tilde{U} \phi_2 \in \tau(i) \implies (\phi_2 \in \tau(i) \land (\phi_1 \in \tau(i) \lor \phi_1 \tilde{U} \phi_2 \in \tau(i))$
Let's define the labelling rule.

For every position $i \in \mathbb{N}$, the labeling $\tau : \mathbb{N} \to 2^{cl(\varphi)}$ satisfies:

- $\forall p \in AP : \ (p \in \tau(i) \implies p \in L(\pi_i)) \land (\neg p \in \tau(i) \implies p \notin L(\pi_i))$
- $\phi_1 \land \phi_2 \in \tau(i) \implies (\phi_1 \in \tau(i) \land \phi_2 \in \tau(i))$
- $\phi_1 \lor \phi_2 \in \tau(i) \implies (\phi_1 \in \tau(i) \lor \phi_2 \in \tau(i))$
- $\bigcirc \phi \in \tau(i) \implies \phi \in \tau(i+1)$
- $\phi_1 U \phi_2 \in \tau(i) \implies (\phi_2 \in \tau(i) \lor (\phi_1 \in \tau(i) \land \phi_1 U \phi_2 \in \tau(i)))$
- $\phi_1 \tilde{U} \phi_2 \in \tau(i) \implies (\phi_2 \in \tau(i) \land (\phi_1 \in \tau(i) \lor \phi_1 \tilde{U} \phi_2 \in \tau(i)))$

We can't expand forever!
A sequence $\pi$ satisfies a formula $\varphi$ if there is a labelling $\tau$ that satisfies:

- The conditions from last slide,
CONSTRUCTION

A sequence \( \pi \) satisfies a formula \( \varphi \) if there is a labelling \( \tau \) that satisfies:

- The conditions from last slide,
- we have \( \varphi \in \tau(0) \), and
A sequence $\pi$ satisfies a formula $\varphi$ if there is a labelling $\tau$ that satisfies:

- The conditions from last slide,
- we have $\varphi \in \tau(0)$, and
- if $\phi_1 \cup \phi_2 \in \tau(i)$ (similarly, $\phi_1 \tilde{\cup} \phi_2$) then $\exists j \geq i : \phi_2 \in \tau(j)$. 


A sequence \( \pi \) satisfies a formula \( \varphi \) if there is a labelling \( \tau \) that satisfies:

- The conditions from last slide,
- we have \( \varphi \in \tau(0) \), and
- if \( \phi_1 U \phi_2 \in \tau(i) \) (similarly, \( \phi_1 \tilde{U} \phi_2 \)) then \( \exists j \geq i : \phi_2 \in \tau(j) \).

Think of the GNBA as such a labelling rule!
Generalized Büchi Automaton $A_\varphi = (Q, \Sigma, \delta, Q_0, F)$ for LTL formula $\varphi$
Generalized Büchi Automaton $A_\varphi = (Q, \Sigma, \delta, Q_0, F)$ for LTL formula $\varphi$

- $\Sigma = 2^{AP}$
Generalized Büchi Automaton $A_\varphi = (Q, \Sigma, \delta, Q_0, F)$ for LTL formula $\varphi$

- $\Sigma = 2^{AP}$

- $Q = 2^{cl(\varphi)}$ where
Generalized Büchi Automaton $A_{\varphi} = (Q, \Sigma, \delta, Q_0, F)$ for LTL formula $\varphi$

- $\Sigma = 2^{AP}$
- $Q = 2^{cl(\varphi)}$ where
  - $\phi_1 \lor \phi_2 \in q \implies (\phi_1 \in q \lor \phi_2 \in q)$
  - $\phi_1 \land \phi_2 \in q \implies (\phi_1 \in q \land \phi_2 \in q)$
Generalized Büchi Automaton \( A_\varphi = (Q, \Sigma, \delta, Q_0, F) \) for LTL formula \( \varphi \)

- \( \Sigma = 2^{AP} \)
- \( Q = 2^{cl(\varphi)} \) where
  - \( \phi_1 \lor \phi_2 \in q \implies (\phi_1 \in q \lor \phi_2 \in q) \)
  - \( \phi_1 \land \phi_2 \in q \implies (\phi_1 \in q \land \phi_2 \in q) \)
- We have \( q' \in \delta(q, a) \) iff
Generalized Büchi Automaton $A_{\varphi} = (Q, \Sigma, \delta, Q_0, F)$ for LTL formula $\varphi$

- $\Sigma = 2^{AP}$
- $Q = 2^{cl(\varphi)}$ where
  - $\phi_1 \lor \phi_2 \in q \implies (\phi_1 \in q \lor \phi_2 \in q)$
  - $\phi_1 \land \phi_2 \in q \implies (\phi_1 \in q \land \phi_2 \in q)$
- We have $q' \in \delta(q, a)$ iff
  - $\forall p \in AP : (p \in q \implies p \in a)$
  - $\forall p \in AP : (\neg p \in q \implies p \notin a)$
Generalized Büchi Automaton $A_\varphi = (Q, \Sigma, \delta, Q_0, F)$ for LTL formula $\varphi$

- $\Sigma = 2^{AP}$
- $Q = 2^{cl(\varphi)}$ where
  - $\varphi_1 \lor \varphi_2 \in q \implies (\varphi_1 \in q \lor \varphi_2 \in q)$
  - $\varphi_1 \land \varphi_2 \in q \implies (\varphi_1 \in q \land \varphi_2 \in q)$
- We have $q' \in \delta(q, a)$ iff
  - $\forall p \in AP : (p \in q \implies p \in a)$
  - $\forall p \in AP : (\neg p \in q \implies p \notin a)$
  - $\bigcirc \varphi \in q \implies \varphi \in q'$
Generalized Büchi Automaton $A_\varphi = (Q, \Sigma, \delta, Q_0, F)$ for LTL formula $\varphi$

- $\Sigma = 2^{AP}$

- $Q = 2^{cl(\varphi)}$ where
  - $\mathbf{\varphi}_1 \lor \mathbf{\varphi}_2 \in q \implies (\mathbf{\varphi}_1 \in q \lor \mathbf{\varphi}_2 \in q)$
  - $\mathbf{\varphi}_1 \land \mathbf{\varphi}_2 \in q \implies (\mathbf{\varphi}_1 \in q \land \mathbf{\varphi}_2 \in q)$

- We have $q' \in \delta(q, a)$ iff
  - $\forall p \in AP : (p \in q \implies p \in a)$
  - $\forall p \in AP : (\neg p \in q \implies p \notin a)$
  - $\Diamond \phi \in q \implies \phi \in q'$
  - $\mathbf{\varphi}_1 U \mathbf{\varphi}_2 \in q \implies (\mathbf{\varphi}_2 \in q \lor (\mathbf{\varphi}_1 \in q \land \mathbf{\varphi}_1 U \mathbf{\varphi}_2 \in q'))$
  - $\mathbf{\varphi}_1 \tilde{U} \mathbf{\varphi}_2 \in q \implies (\mathbf{\varphi}_2 \in q \land (\mathbf{\varphi}_1 \in q \lor \mathbf{\varphi}_1 \tilde{U} \mathbf{\varphi}_2 \in q'))$
Generalized Büchi Automaton $A_\varphi = (Q, \Sigma, \delta, Q_0, F)$ for LTL formula $\varphi$

- $\Sigma = 2^{AP}$
- $Q = 2^{cl(\varphi)}$ where
  - $\phi_1 \lor \phi_2 \in q \implies (\phi_1 \in q \lor \phi_2 \in q)$
  - $\phi_1 \land \phi_2 \in q \implies (\phi_1 \in q \land \phi_2 \in q)$
- We have $q' \in \delta(q, a)$ iff
  - $\forall p \in AP: (p \in q \implies p \in a)$
  - $\forall p \in AP: (\neg p \in q \implies p \notin a)$
  - $\bigcirc \phi \in q \implies \phi \in q'$
  - $\phi_1 U \phi_2 \in q \implies (\phi_2 \in q \lor (\phi_1 \in q \land \phi_1 U \phi_2 \in q'))$
  - $\phi_1 \tilde{U} \phi_2 \in q \implies (\phi_2 \in q \land (\phi_1 \in q \lor \phi_1 \tilde{U} \phi_2 \in q'))$
- $Q_0 = \{ q \mid \varphi \in q \}$
Generalized Büchi Automaton $A_\varphi = (Q, \Sigma, \delta, Q_0, \mathcal{F})$ for LTL formula $\varphi$

- $\Sigma = 2^{AP}$

- $Q = 2^{cl(\varphi)}$ where
  - $\phi_1 \lor \phi_2 \in q \implies (\phi_1 \in q \lor \phi_2 \in q)$
  - $\phi_1 \land \phi_2 \in q \implies (\phi_1 \in q \land \phi_2 \in q)$

- We have $q' \in \delta(q, a)$ iff
  - $\forall p \in AP : (p \in q \implies p \in a)$
  - $\forall p \in AP : (\neg p \in q \implies p \notin a)$
  - $\bigcirc \phi \in q \implies \phi \in q'$
  - $\phi_1 U \phi_2 \in q \implies (\phi_2 \in q \lor (\phi_1 \in q \land \phi_1 U \phi_2 \in q'))$
  - $\phi_1 \tilde{U} \phi_2 \in q \implies (\phi_2 \in q \land (\phi_1 \in q \lor \phi_1 \tilde{U} \phi_2 \in q'))$

- $Q_0 = \{ q \mid \varphi \in q \}$

- $\mathcal{F} = \{ F_1, \ldots, F_m \}$ where $F_i = \{ q \mid e_i, \phi_i \in q \lor e_i \notin q \}$
  - $e_i$'s are eventuality formulas of the form $-U \phi_i$ or $-\tilde{U} \phi_i$. 
COMPLEXITY
$|cl(\varphi)|$ is linear on $|\varphi|$, therefore $|Q|$ is at most $O(2^{|\varphi|})$. 
\[ |cl(\varphi)| \text{ is linear on } |\varphi|, \text{ therefore } |Q| \text{ is at most } O(2^{|\varphi|}). \]

More generally \(|A_\varphi| \text{ is of } O(2^{|\varphi|}). \)
$|\text{cl}(\varphi)|$ is linear on $|\varphi|$, therefore $|Q|$ is at most $O(2^{|\varphi|})$.

More generally $|A_\varphi|$ is of $O(2^{|\varphi|})$.

Therefore, size of $TS \otimes A_\varphi$ is of $O(|TS| \cdot 2^{|\varphi|})$. 
**COMPLEXITY**

$|cl(\varphi)|$ is linear on $|\varphi|$, therefore $|Q|$ is at most $O(2^{|\varphi|})$.

More generally $|A_\varphi|$ is of $O(2^{|\varphi|})$.

Therefore, size of $TS \otimes A_\varphi$ is of $O(|TS| \cdot 2^{|\varphi|})$.

Model checking can be done in time $O(|TS| \cdot 2^{|\varphi|})$ by looking for a path in the product construction that satisfies an accepting condition of $A$. 
OPTIMIZING MODEL CHECKING
PARTIAL ORDER REDUCTION
CONCURRENCY

Chapter 8
Partial Order Reduction

Consider the parallel composition of a number of processes $P_1$ through $P_n$. The size of the state space of $P_1 \parallel P_2 \parallel ... \parallel P_n$, where $\parallel$ denotes some parallel composition operator, is exponential in the number $n$ of processes. To check the validity of a linear-time property of this system requires an inspection of all states in the underlying transition system. In the simple setting where there are no synchronizations between the individual processes—neither through shared variables nor via communication channels or the like—there are $n!$ different orderings of the interleaved execution of $n$ local actions. The effect of concurrent actions, however, is often independent of their ordering. Consider, e.g., the assignments $x := x + 1$ and $y := y - 3$ in the concurrent system $P_1 \parallel P_2$, where $x$ is a local variable of $P_1$, say, and $y$ of $P_2$, and $\parallel$ denotes the interleaving operator. It is evident that regardless of the ordering of these assignments, the result will be the same. This is illustrated in Figure 8.1:

\[ \alpha \parallel \beta \]
8.1, where actions $\alpha$ and $\beta$ denote the assignments of $P_1$ and $P_2$, respectively. Instead of analyzing the $2!$ orderings of $x := x + 1$ and $y := y - 3$, it suffices to check just a single ordering. This is correct as long as the intermediate states reached after the execution of either $\alpha$ or $\beta$ (see states $t$ and $u$ in Figure 8.1), are irrelevant for the properties to be proved. Extending the simple example with a third process $P_3$ that, e.g., resets its variable $z$ to 0, yields, following an analogous reasoning, that it suffices to consider just one of the $3!$ possible orderings. This approach can be generalized for action sequences $\alpha_1 \alpha_2 \ldots \alpha_n$ and $\beta_1 \beta_2 \ldots \beta_m$ that are executed independently by processes $P_1$ and $P_2$. The transition system of $P_1 ||| P_2 ||| \ldots ||| P_n$ represents all interleavings of these action sequences, whereas as in a path fragment respecting the order in the sequences, provided the intermediate states are irrelevant.

Put in a nutshell, the aim of partial order reduction, the technique that is treated in this chapter, is to reduce the number of possible orderings that need to be analyzed for checking formulae stated in a temporal logic such as LTL or CTL. This is in concept to reduce the state space of the transition system that needs to be analyzed. Thus, the idea is to replace the full transition system for $P_1 ||| P_2 ||| \ldots ||| P_n$ by a small fragment.

Figure 8.2 illustrates this for two processes that execute the action sequences $\alpha_1 \alpha_2$ and $\beta_1 \beta_2$, respectively. The transition system on the left contains all possible interleavings, while the reduced transition system on the right just consists of a single path that might serve as a representative for all possible interleavings. On increasing the number of concurrent processes, this effect becomes even more drastic—the size of the full transition system grows exponentially in the number of processes, whereas the reduced system consists of a single path that grows linear in $n$. 

$(\alpha_1; \alpha_2) || (\beta_1; \beta_2)$
CONCURRENCY

Partial Order Reduction

In place of analyzing the $2!$ orderings of $x := x + 1$ and $y := y - 3$, it suffices to check just a single ordering. This is correct as long as the intermediate states reached after the execution of either $\alpha$ or $\beta$ (see states $t$ and $u$ in Figure 8.1), are irrelevant for the properties to be proved. Extending the simple example with a third process $P_3$ that, e.g., resets its variable $z$ to 0, yields following an analogous reasoning that it suffices to consider just one of the $3!$ possible orderings. This approach can be generalized for action sequences $\alpha_1 \alpha_2 \ldots \alpha_n$ and $\beta_1 \beta_2 \ldots \beta_m$ that are executed independently by processes $P_1$ and $P_2$. The transition system of $P_1 ||| P_2 ||| \ldots ||| P_n$ represents all interleavings of these action sequences, whereas as in the path fragment respecting the order in the sequences, provided the intermediate states are irrelevant.

Put in a nutshell, the aim of partial order reduction, the technique that is treated in this chapter, is to reduce the number of possible orderings that need to be analyzed for checking formulae stated in a temporal logic such as LTL or CTL. This main concept is to reduce the state space of the transition system that needs to be analyzed. Thus, the idea is to replace the full transition system for $P_1 ||| P_2 ||| \ldots ||| P_n$ by a small fragment.

Figure 8.2 illustrates this for two processes that execute the action sequences $\alpha_1 \alpha_2$ and $\beta_1 \beta_2$, respectively. The transitions on the left contain all possible interleavings, while the reduced transition system on the right just consists a single path that might serve as a representative for all possible interleavings. On increasing the number of concurrent processes, this effect becomes even more drastic—the size of the full transition system grows exponentially in the number of processes, whereas the reduced system consists of a single path that grows linear in $n$. 

To avoid peak memory requirements, such a reduced transition system is obtained without

The State Explosion Problem

Allowing all possible orderings is a potential cause of the state explosion problem. To see this, consider transitions that can be executed concurrently. In this case, there are different orderings and different states (one for each subset of the transitions). If the specification does not distinguish between these sequences, it is beneficial to consider only one with $3$ states.
We don’t have to check \textit{all interleavings} for most properties. It suffices to check some \textit{representative} interleavings. The \textit{property} being checked determines what these representative interleaving are.
INDEPENDENT ACTIONS

\[ enabled(s) = \{ \alpha | \exists s' : s \xrightarrow{\alpha} s' \} \]
INDEPENDENT ACTIONS

\[
\text{enabled}(s) = \{ \alpha | \exists s' : s \xrightarrow{\alpha} s' \}
\]

Let \( \alpha(s) \) be state \( s' \) where \( s \xrightarrow{\alpha} s' \).
INDEPENDENT ACTIONS

\[
\text{enabled}(s) = \{ \alpha | \exists s' : s \xrightarrow{\alpha} s' \}
\]

Let \( \alpha(s) \) be state \( s' \) where \( s \xrightarrow{\alpha} s' \).

Actions \( \alpha \) and \( \beta \) are independent iff for all states \( s \) where \( \alpha, \beta \in \text{enabled}(s) \):

\[
\alpha \in \text{enabled}(\beta(s)) \land \beta \in \text{enabled}(\alpha(s)) \land \alpha(\beta(s)) = \beta(\alpha(s))
\]
INDEPENDENT ACTIONS

\[ \text{enabled}(s) = \{ \alpha \mid \exists s' : s \xrightarrow{\alpha} s' \} \]

Let \( \alpha(s) \) be state \( s' \) where \( s \xrightarrow{\alpha} s' \).

Actions \( \alpha \) and \( \beta \) are independent iff for all states \( s \) where \( \alpha, \beta \in \text{enabled}(s) \):

\[ \alpha \in \text{enabled}(\beta(s)) \land \beta \in \text{enabled}(\alpha(s)) \land \alpha(\beta(s)) = \beta(\alpha(s)) \]
The independence of action $\alpha$ TS be an action-deterministic transition system, we obtain an infinite execution fragment which first executes $\alpha_0$). Then Lemma 8.6 applied to the finite prefixes of $\alpha$ yields the existence of finite $\rho_1$, and $\rho_2$ is enabled in state $\rho_1$. More precisely, we have:

$$s = s_0 \xrightarrow{\beta_1} s_1 \xrightarrow{\beta_2} s_2 \xrightarrow{\beta_3} \ldots \xrightarrow{\beta_{n-1}} s_{n-1} \xrightarrow{\beta_n} s_n$$

Consider the infinite execution fragment

Figure 8.4: Permuting Independent Actions

PERMUTING INDEPENDENT ACTIONS
If $\alpha$ is independent of $\{\beta_1, \ldots, \beta_n\}$
Lemma 8.7. Adding an Independent Action

The independence of action \( \alpha \) is illustrated in Figure 8.4.

\[ s = s_0 \xrightarrow{\alpha} s_1 \xrightarrow{\beta_1} s_2 \xrightarrow{\beta_2} \ldots \xrightarrow{\beta_{n-1}} s_{n-1} \xrightarrow{\beta_n} s_n \]

If \( \alpha \) is independent of \( \{\beta_1, \ldots, \beta_n\} \),

\[ s = s_0 \xrightarrow{\alpha} s_1 \xrightarrow{\beta_1} s_2 \xrightarrow{\beta_2} \ldots \xrightarrow{\beta_{n-1}} s_{n-1} \xrightarrow{\beta_n} s_n \]

\[ t_0 \xrightarrow{\beta_1} t_1 \xrightarrow{\beta_2} t_2 \xrightarrow{\beta_3} \ldots \xrightarrow{\beta_{n-1}} t_{n-1} \xrightarrow{\beta_n} t_n = t \]
**STUTTER/INVISIBLE ACTION**

\[ s = s_0 \xrightarrow{\beta_1} s_1 \xrightarrow{\beta_2} s_2 \xrightarrow{\beta_3} \ldots \xrightarrow{\beta_{n-1}} s_{n-1} \xrightarrow{\beta_n} s_n \]

\[ t_0 \xrightarrow{\beta_1} t_1 \xrightarrow{\beta_2} t_2 \xrightarrow{\beta_3} \ldots \xrightarrow{\beta_{n-1}} t_{n-1} \xrightarrow{\beta_n} t_n = t \]
α is a **stutter action** if \( L(s) = L(\alpha(s)) \) for all \( s \) where \( \alpha \in \text{enabled}(s) \).
**STUTTER/INVISIBLE ACTION**

\[ s = s_0 \xrightarrow{\beta_1} s_1 \xrightarrow{\beta_2} s_2 \xrightarrow{\beta_3} \ldots \xrightarrow{\beta_{n-1}} s_{n-1} \xrightarrow{\beta_n} s_n \]

\[ t_0 \xrightarrow{\beta_1} t_1 \xrightarrow{\beta_2} t_2 \xrightarrow{\beta_3} \ldots \xrightarrow{\beta_{n-1}} t_{n-1} \xrightarrow{\beta_n} t_n = t \]

\( \alpha \) is a stutter action if \( L(s) = L(\alpha(s)) \) for all \( s \) where \( \alpha \in \text{enabled}(s) \).

\[ s_0 \xrightarrow{\beta_1} s_1 \xrightarrow{\beta_2} \ldots \xrightarrow{\beta_n} s_n \xrightarrow{\alpha} t, \text{ and} \]

\[ s_0 \xrightarrow{\alpha} t_0 \xrightarrow{\beta_1} \ldots \xrightarrow{\beta_{n-1}} t_{n-1} \xrightarrow{\beta_n} t \]

are stutter equivalent.
**STUTTER/INVISIBLE ACTION**

\[ s = s_0 \xrightarrow{\beta_1} s_1 \xrightarrow{\beta_2} s_2 \xrightarrow{\beta_3} \ldots \xrightarrow{\beta_{n-1}} s_{n-1} \xrightarrow{\beta_n} s_n \]

\[ t_0 \xrightarrow{\beta_1} t_1 \xrightarrow{\beta_2} t_2 \xrightarrow{\beta_3} \ldots \xrightarrow{\beta_{n-1}} t_{n-1} \xrightarrow{\beta_n} t_n = t \]

\[ \alpha \] is a stutter action if \( L(s) = L(\alpha(s)) \) for all \( s \) where \( \alpha \in \text{enabled}(s) \).

\[ s_0 \xrightarrow{\beta_1} s_1 \xrightarrow{\beta_2} \ldots \xrightarrow{\beta_n} s_n \xrightarrow{\alpha} t, \text{ and} \]

\[ s_0 \xrightarrow{\alpha} t_0 \xrightarrow{\beta_1} \ldots \xrightarrow{\beta_{n-1}} t_{n-1} \xrightarrow{\beta_n} t \]

are stutter equivalent.

This is true for infinite paths as well.
Infinite paths $\pi_1$ and $\pi_2$ are stutter equivalent if there exists an infinite sequence $A_0, A_1, \ldots$ (where $A_i \subseteq AP$) and sequences of natural numbers $n_0, n_1, \ldots$ and $m_0, m_1, \ldots$ such that

$$L(\pi_1) = \underbrace{A_0 \ldots A_0}_{n_0\text{-times}} \underbrace{A_1 \ldots A_1}_{n_1\text{-times}} \underbrace{A_2 \ldots A_2}_{n_2\text{-times}} \ldots$$

$$L(\pi_2) = \underbrace{A_0 \ldots A_0}_{m_0\text{-times}} \underbrace{A_1 \ldots A_1}_{m_1\text{-times}} \underbrace{A_2 \ldots A_2}_{m_2\text{-times}} \ldots$$

Finite path fragments $\hat{\pi}_1$ in $TS_1$ and $\hat{\pi}_2$ in $TS_2$ are stutter equivalent, denoted $\hat{\pi}_1 \equiv \hat{\pi}_2$, if there exists a finite sequence $A_0, \ldots, A_n \in (2^{AP})^+$ such that $\text{trace}(\hat{\pi}_1)$ and $\text{trace}(\hat{\pi}_2)$ are contained in the language given by the regular expression $A_0 \ldots A_0 A_1 \ldots A_1 A_2 \ldots A_2 \ldots$.
Infinite paths $\pi_1$ and $\pi_2$ are stutter equivalent if there exists an infinite sequence $A_0, A_1, \ldots$ (where $A_i \subseteq AP$) and a sequence of natural numbers $n_0, n_1, \ldots$ and $m_0, m_1, \ldots$ such that

$$L(\pi_1) = A_0 \ldots A_0 A_1 \ldots A_1 A_2 \ldots A_2 \ldots$$

$$L(\pi_2) = A_0 \ldots A_0 A_1 \ldots A_1 A_2 \ldots A_2 \ldots$$

where $n_0$-times, $n_1$-times, $n_2$-times, $m_0$-times, $m_1$-times, $m_2$-times, etc.

If $\pi_1$ and $\pi_2$ are stutter equivalent, then they satisfy the same set of $LTL_\Box$ formulas.
The Linear-Time Ample Set Approach

Lemma 8.11. Adding an Independent Stutter Action

Let $TS$ be an action-deterministic transition system, $s$ as a state in $TS$, and $\rho$ and $\rho'$ be infinite execution fragments starting in $s$ with the action sequences $\beta_1 \beta_2 \beta_3 \ldots$ and $\alpha \beta_1 \beta_2 \beta_3 \ldots$, respectively, such that $\alpha$ is a stutter action which is independent of $\{\beta_1, \beta_2, \beta_3, \ldots\}$.

Then $\rho \equiv \rho'$.

Proof: Let $\rho = s_0 \xrightarrow{\alpha} s_1 \xrightarrow{\beta_1} s_2 \xrightarrow{\beta_2} s_3 \xrightarrow{\beta_3} \ldots$ and $\rho' = s_0 \xrightarrow{\alpha} t_0 \xrightarrow{\beta_1} t_1 \xrightarrow{\beta_2} t_2 \xrightarrow{\beta_3} \ldots$ where $s_0 = s$.

Then, $s_i = \alpha(t_i)$ for all $i \geq 0$. Since $\alpha$ is a stutter action we have $L(s_i) = L(t_i)$ for all $i \geq 0$. With $A_i = L(s_i)$ we get

$$\text{trace}(\rho) = L(s_0) L(s_1) L(s_2) \ldots \equiv A_0 A_1 A_2 \ldots$$

$$\text{trace}(\rho') = L(s_0) L(t_0) L(t_1) L(t_2) \ldots \equiv A_0 A_0 A_1 A_2 \ldots$$

Thus, both traces have the form $A + A_1 A_2 \ldots$ which yields $\rho \equiv \rho'$.

Lemmas 8.10 and 8.11 yield the basis of the partial order reduction approach. During partial order reduction, any stutter equivalence class of executions in the full system $TS$ is represented by at least one execution in the reduced system $\hat{TS}$. (One might say that partial order reduction amounts to model checking using representative executions.) The representatives in $\hat{TS}$ of $TS$’s stutter equivalence classes arise by permuting independent actions and adding independent stutter actions.

8.2 The Linear-Time Ample Set Approach

We consider partial order reduction for LTL using so-called ample sets. The basic idea is the following. Consider a high-level specification of an asynchronous system. Using traditional state space generation, for each encountered state all direct successors are explored. That is, for each action $\alpha \in \text{Act}(s)$, the successor state $\alpha(s)$ is determined, and when encountered for the first time, generated. With partial order reduction using ample sets, the set $\text{ample}(s) \subseteq \text{Act}(s)$ will be explored instead of the entire set $\text{Act}(s)$. That is, all direct successors in $\text{Act}(s) \setminus \text{ample}(s)$ are not explored, and possibly not generated at all. By choosing appropriate action sets $\text{ample}(\cdot)$, this approach yields a—hopefully small—fragment of the full transition system $TS = (S, \text{Act}, \rightarrow, \text{I}, \text{AP}, L)$.

As $TS$ will never be generated, the peak memory requirements are determined by the size of the fragment $\hat{TS}$ rather than by $TS$. The reduced transition system $\hat{TS}$ results from the transition relation $\Rightarrow$ which is defined by

$$s \xrightarrow{\alpha} s' \land \alpha \in \text{ample}(s) \Rightarrow s \xrightarrow{\alpha} s'.$$
We a transition system by replacing its transition relation with a reduced one:

\[ s \xrightarrow{\alpha} s' \land \alpha \in \text{ample}(s) \]

\[ s \xrightarrow{\alpha} s' \]

when is it correct to check the reduced TS for a property?
We a transition system by replacing its transition relation with a reduced one:

\[
\begin{align*}
    s & \xrightarrow{\alpha} s' \land \alpha \in \text{ample}(s) \\
    \implies & \\
    s & \xrightarrow{\alpha} s'
\end{align*}
\]

when is it correct to check the reduced TS for a property?

If for every execution \( \pi \) of \( TS \), we find a stutter equivalent execution \( \pi' \) of the reduced \( \hat{TS} \), then we have:

\[
\hat{TS} \models \varphi \iff TS \models \varphi
\]

for all \( \varphi \in LTL_{\neg \Box} \).
Every path the reduced TS is by definition a path in the original.

Under what conditions is the reduced one good enough?
Every path the reduced TS is by definition a path in the original.

Under what conditions is the reduced one good enough?

Consider a path $\rho_0$ that is a path in $TS$ but not path in $\hat{TS}$.

$$\rho_0 = \underbrace{u \xrightarrow{\gamma_1} \ldots \xrightarrow{\gamma_m} s}_{\text{prefix } \rho_0} \underbrace{s \xrightarrow{\beta_1} s_1 \xrightarrow{\beta_2} s_2 \ldots}_{\text{suffix } \rho \text{ with } \beta_1 \notin \text{ample}(s)}$$

We need to argue that it is stutter equivalent to a path $\rho_1$ of $\hat{TS}$. 
WHAT CAN GUARANTEE THIS?

Consider a path $\rho_0$ that is a path in $TS$ but not path in $\hat{TS}$.

$$\rho_0 = \begin{array}{c}
\text{prefix } \varrho_0 \quad \begin{array}{c}
\xrightarrow{u} \gamma_1 \quad \ldots \quad \xrightarrow{\gamma_m} s
\end{array}
\end{array}
\begin{array}{c}
\text{suffix } \rho \text{ with } \beta_1 \notin \text{ample}(s)
\end{array}
\begin{array}{c}
\xrightarrow{s} \beta_1 \rightarrow s_1 \xrightarrow{\beta_2} s_2 \ldots
\end{array}
\quad \text{for } m \geq 0.
$$

We need to argue that it is stutter equivalent to a path $\rho_1$ of $\hat{TS}$.
WHAT CAN GUARANTEE THIS?

Consider a path $\rho_0$ that is a path in $TS$ but not path in $\hat{TS}$.

$$\rho_0 = u \xrightarrow{\gamma_1} \ldots \xrightarrow{\gamma_m} s \xrightarrow{\beta_1} s_1 \xrightarrow{\beta_2} s_2 \ldots$$

prefix $\rho_0$  suffix $\rho$ with $\beta_1 \notin \text{ample}(s)$

for $m \geq 0$.

We need to argue that it is stutter equivalent to a path $\rho_1$ of $\hat{TS}$

\[ s \xrightarrow{\beta_1} \ldots \xrightarrow{\beta_n} s_n \xrightarrow{\alpha} t \xrightarrow{\beta_{n+2}} s_{n+2} \xrightarrow{\beta_{n+3}} \ldots \]

\[ s \xrightarrow{\alpha} t_0 \xrightarrow{\beta_1} \ldots \xrightarrow{\beta_n} t \xrightarrow{\beta_{n+2}} s_{n+2} \xrightarrow{\beta_{n+3}} \ldots \]

Case 1: there is an action in the suffix that belongs to ample(s)
WHAT CAN GUARANTEE THIS?

Consider a path $\rho_0$ that is a path in $TS$ but not path in $\hat{TS}$.

$$\rho_0 = u \gamma_1 \ldots \gamma_m s$$

prefix $\rho_0$  suffix $\rho$ with $\beta_1 \not\in$ ample($s$)

for $m \geq 0$.

We need to argue that it is stutter equivalent to a path $\rho_1$ of $\hat{TS}$

**Case 1:** there is an action in the suffix that belongs to ample($s$)

**Case 2:** no action in the suffix belongs to ample($s$)
(A1) **Nonemptiness condition**
\[ \emptyset \neq \text{ample}(s) \subseteq \text{Act}(s) \]

(A2) **Dependency condition**
Let \( s \xrightarrow{\beta_1} \ldots \xrightarrow{\beta_n} s_n \xrightarrow{\alpha} t \) be a finite execution fragment in \( TS \).
If \( \alpha \) depends on \( \text{ample}(s) \), then \( \beta_i \in \text{ample}(s) \) for some \( 0 < i \leq n \).

(A3) **Stutter condition**
If \( \text{ample}(s) \neq \text{Act}(s) \) then any \( \alpha \in \text{ample}(s) \) is a stutter action.

(A4) **Cycle condition**
For any cycle \( s_0 \ s_1 \ldots \ s_n \) in \( \hat{TS} \) and \( \alpha \in \text{Act}(s_i) \), for some \( 0 < i \leq n \), there exists \( j \in \{1, \ldots, n\} \) such that \( \alpha \in \text{ample}(s_j) \).
(A1) **Nonemptiness condition**
\[ \emptyset \neq \text{ample}(s) \subseteq \text{Act}(s) \]

(A2) **Dependency condition**

Let \( s \xrightarrow{\beta_1} \ldots \xrightarrow{\beta_n} s_n \xrightarrow{\alpha} t \) be a finite execution fragment in \( TS \).

If \( \alpha \) depends on \( \text{ample}(s) \), then \( \beta_i \in \text{ample}(s) \) for some \( 0 < i \leq n \).

(A3) **Stutter condition**

If \( \text{ample}(s) \neq \text{Act}(s) \) then any \( \alpha \in \text{ample}(s) \) is a stutter action.

(A4) **Cycle condition**

For any cycle \( s_0 s_1 \ldots s_n \) in \( \hat{TS} \) and \( \alpha \in \text{Act}(s_i) \), for some \( 0 < i \leq n \), there exists \( j \in \{1, \ldots, n\} \) such that \( \alpha \in \text{ample}(s_j) \).
### Constraints on Ample Sets

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>(A1) Nonemptiness condition</strong></td>
<td>$\emptyset \neq \text{ample}(s) \subseteq \text{Act}(s)$</td>
</tr>
<tr>
<td><strong>(A2) Dependency condition</strong></td>
<td>Let $s \xrightarrow{\beta_1} \ldots \xrightarrow{\beta_n} s_n \xrightarrow{\alpha} t$ be a finite execution fragment in $TS$. If $\alpha$ depends on $\text{ample}(s)$, then $\beta_i \in \text{ample}(s)$ for some $0 &lt; i \leq n$.</td>
</tr>
<tr>
<td><strong>(A3) Stutter condition</strong></td>
<td>If $\text{ample}(s) \neq \text{Act}(s)$ then any $\alpha \in \text{ample}(s)$ is a stutter action.</td>
</tr>
<tr>
<td><strong>(A4) Cycle condition</strong></td>
<td>For any cycle $s_0 s_1 \ldots s_n$ in $\hat{TS}$ and $\alpha \in \text{Act}(s_i)$, for some $0 &lt; i \leq n$, there exists $j \in {1, \ldots, n}$ such that $\alpha \in \text{ample}(s_j)$.</td>
</tr>
</tbody>
</table>

**Example 8.12. Ample Set Conditions**

Consider the transition system $TS$ in Figure 8.7 (left part) over $AP = \{a\}$. Action $\beta$ is a stutter action, and is independent of $\{\alpha, \gamma, \delta\}$. Let $\text{ample}(s_0) = \{\beta\}$. This choice satisfies constraints (A1) through (A3). Consider now state $s_2$. The choice $\text{ample}(s_2) = \{\alpha\}$ violates (A3), as $\alpha$ is not a stutter action. $\text{ample}(s_2) = \{\delta\}$ violates the cycle condition (A4): the reduced transition system $\hat{TS}$ would contain the cycle $s_0 s_2 s_2$ with $\alpha \in \text{Act}(s_2)$, but $\alpha \not\in \text{ample}(s_2)$. Thus, we select $\text{ample}(s_2) = \{\alpha, \delta\}$. The nonemptiness condition (A1) then leaves no freedom for $s_3$: $\text{ample}(s_3) = \{\gamma\}$. The resulting reduced transition system $\hat{TS}$ is depicted in Figure 8.7 (right part). The traces of $TS$ and $\hat{TS}$ are either of the form $(\emptyset + \{a\} + \emptyset)^\omega$ or $(\emptyset + \{a\} + \emptyset)^* \emptyset^\omega$. Hence, $TS \approx \hat{TS}$.

We now state the main result of this section. The proof of this result is provided by a series of lemmas, presented in the remainder of this section.

**Theorem 8.13. Correctness of the Ample Set Approach**

Let $TS$ be an action-deterministic, finite transition system without terminal states. Then if conditions (A1) through (A4) are satisfied, then $\hat{TS} = TS$.

This theorem asserts that whenever $\hat{TS}$ is constructed from $TS$ using ample sets that all ample actions are independent of all non-ample actions in any reachable state.
CONSTRAINTS ON AMPLE SETS

(A1) **Nonemptiness condition**
\[ \emptyset \neq \text{ample}(s) \subseteq \text{Act}(s) \]

(A2) **Dependency condition**
Let \( s \xrightarrow{\beta_1} \ldots \xrightarrow{\beta_n} s_n \xrightarrow{\alpha} t \) be a finite execution fragment in \( TS \).
If \( \alpha \) depends on \( \text{ample}(s) \), then \( \beta_i \in \text{ample}(s) \) for some \( 0 < i \leq n \).

(A3) **Stutter condition**
If \( \text{ample}(s) \neq \text{Act}(s) \) then any \( \alpha \in \text{ample}(s) \) is a stutter action.

(A4) **Cycle condition**
For any cycle \( s_0 s_1 \ldots s_n \) in \( \hat{TS} \) and \( \alpha \in \text{Act}(s_i) \), for some \( 0 < i \leq n \),
there exists \( j \in \{ 1, \ldots, n \} \) such that \( \alpha \in \text{ample}(s_j) \).
CONSTRAINTS ON AMPLE SETS

(A1) Nonemptiness condition
\( \emptyset \neq \text{ample}(s) \subseteq \text{Act}(s) \)

(A2) Dependency condition
Let \( s \xrightarrow{\beta_1} \ldots \xrightarrow{\beta_n} s_n \xrightarrow{\alpha} t \) be a finite execution fragment in TS. If \( \alpha \) depends on \( \text{ample}(s) \), then \( \beta_i \in \text{ample}(s) \) for some \( 0 < i \leq n \).

(A3) Stutter condition
If \( \text{ample}(s) \neq \text{Act}(s) \) then any \( \alpha \in \text{ample}(s) \) is a stutter action.

(A4) Cycle condition
For any cycle \( s_0 s_1 \ldots s_n \) in TS and \( \alpha \in \text{Act}(s_i) \), for some \( 0 < i \leq n \), there exists \( j \in \{1, \ldots, n\} \) such that \( \alpha \in \text{ample}(s_j) \).
Finite case:

\[ s \xrightarrow{\beta_1} s_1 \xrightarrow{\beta_2} \ldots \xrightarrow{\beta_n} s_n \xrightarrow{\alpha} t \]

if \( \alpha \in \text{ample}(s) \) and \( \beta_i \notin \text{ample}(s) \) (for all \( i \)), and Conditions A1-3 are satisfied, then there exists a stutter equivalent execution:

\[ s \xrightarrow{\alpha} t_0 \xrightarrow{\beta_1} t_1 \xrightarrow{\beta_2} \ldots \xrightarrow{\beta_{n-1}} t_{n-1} \xrightarrow{\beta_n} t \]
WHY DOES THIS WORK?

Finite case:

\[
\begin{align*}
  s \xrightarrow{\beta_1} s_1 \xrightarrow{\beta_2} \ldots \xrightarrow{\beta_n} s_n \xrightarrow{\alpha} t
\end{align*}
\]

if \( \alpha \in \text{ample}(s) \) and \( \beta_i \notin \text{ample}(s) \) (for all \( i \)), and Conditions A1-3 are satisfied, then there exists a stutter equivalent execution:

\[
\begin{align*}
  s \xrightarrow{\alpha} t_0 \xrightarrow{\beta_1} t_1 \xrightarrow{\beta_2} \ldots \xrightarrow{\beta_{n-1}} t_{n-1} \xrightarrow{\beta_n} t
\end{align*}
\]

Infinite case:

\[
\begin{align*}
  s \xrightarrow{\beta_1} s_1 \xrightarrow{\beta_2} s_2 \xrightarrow{\beta_3} \ldots
\end{align*}
\]

where \( \beta_i \notin \text{ample}(s) \) (for all \( i \)), and Conditions A1-3 are satisfied, then there exists a stutter equivalent execution:

\[
\begin{align*}
  s \xrightarrow{\alpha} t_0 \xrightarrow{\beta_1} t_1 \xrightarrow{\beta_2} t_2 \xrightarrow{\beta_3} \ldots
\end{align*}
\]

where \( \alpha \in \text{ample}(s) \).
CYCLE CONDITION
Let us explain why the above-mentioned replacement process, which should transform the transition system $TS$ into a stutter-equivalent execution of $\omega$, succeeds. Hence, case 2 might generate a series of traces $\rho$. In fact, $\rho$ with action sequences $\alpha\beta\gamma\ldots$ is never performed.

The following example illustrates the necessity of cycle condition (A4). Consider the action sequence $\alpha\beta\gamma\ldots$ that is not stutter-equivalent to the original execution $s_0 \rightarrow s_1 \rightarrow \ldots \rightarrow t_0 \rightarrow t_1 \rightarrow t_2 \rightarrow \ldots$. Therefore, the transformation according to case 2 may fail. In this example, case 1 also never generates a stutter-equivalent execution $s_0 \rightarrow s_1 \rightarrow \ldots \rightarrow t_0 \rightarrow t_1 \rightarrow t_2 \rightarrow \ldots$. Hence, the reduced transition system is depicted in the left part of Figure 8.10. The associated trace is never performed.
Consider the transition systems

Example 8.20. Necessity of Cycle Condition (A4)

In absence of condition (A4), the transformation according to case 2 may ignore forever in $\hat{\alpha}$. The associated trace is $\hat{\alpha}$, however, the does $\hat{\alpha}_0$. The reduced transition system $\langle s_0, t_0 \rangle \emptyset$ is depicted in the left part of Figure 8.10. The reduced transition system $\langle s_0, t_1 \rangle \emptyset$, $\langle s_1, t_0 \rangle \{a\}$, $\langle s_1, t_1 \rangle \emptyset$, and $\langle s_1, t_2 \rangle \{a\}$ have the action sequence $\hat{\alpha}_1$, $\hat{\alpha}_2$, $\hat{\alpha}_3$, $\beta$, $\gamma$, and $\alpha_1$ respectively.

Partial Order Reduction $\approx \rho_0$. The case 2 might generate a series of executions that is not stutter-equivalent to the original execution $\rho$. Hence, $\hat{\alpha}_1 \rho \hat{\alpha}_2$. Actually, $\hat{\alpha}_1 \rho \hat{\alpha}_2$ (This can be seen by considering the LTL formula $\Box (s_0 \lor \alpha) \land \Box (s_1 \lor \beta)$). In this example, case 1 fails. Hence, $\alpha_2 \rho \alpha_3$.
Consider the transition systems also be seen by considering the LTL executions of

Let us explain why the above-mentioned replacement process, which should transform the

\[ \rho \rightarrow \] executions of

\[ \alpha \]

\[ \beta \]

\[ \alpha \]

\[ \gamma \]

In a similar way, in absence of condition (A4), the transformation according to case 2 may

The associated trace is depicted in the left part of Figure 8.10. The reduced transition system

\[ \rho \rightarrow \] is never performed. Hence, this example, case 1 fails. Hence,

\[ \alpha \rightarrow \] \[ \beta \]

\[ \alpha \rightarrow \]

\[ \alpha \rightarrow \] \[ \beta \]

\[ \alpha \rightarrow \]

\[ \alpha \rightarrow \]

\[ \alpha \rightarrow \]

\[ \alpha \rightarrow \]

\[ \alpha \rightarrow \]

\[ \alpha \rightarrow \]

\[ \alpha \rightarrow \]

\[ \alpha \rightarrow \]

\[ \alpha \rightarrow \]

\[ \alpha \rightarrow \]

\[ \alpha \rightarrow \]

\[ \alpha \rightarrow \]

\[ \alpha \rightarrow \]

\[ \alpha \rightarrow \]

\[ \alpha \rightarrow \]

\[ \alpha \rightarrow \]

\[ \alpha \rightarrow \]
Consider the transition systems also be seen by considering the LTL

\[ \alpha_i \mapsto \alpha_2 \alpha_3 \alpha_1 \alpha_2 \alpha_3 \ldots \]
\[ \alpha_1 \alpha_2 \beta \alpha_3 \alpha_1 \alpha_2 \alpha_3 \ldots \]
\[ \alpha_1 \alpha_2 \alpha_3 \beta \alpha_1 \alpha_2 \alpha_3 \ldots \]
\[ \alpha_1 \alpha_2 \alpha_3 \alpha_1 \beta \alpha_2 \alpha_3 \ldots \]
\[ \ldots \]
HOW TO INTEGRATE IT IN THE MODEL CHECKING ALGORITHM?
CHEAP CONDITION CHECKING
A1 and A3 are relatively easy (at least their syntactic version)
A1 and A3 are relatively easy (at least their syntactic version).

A2 can be as complex as checking the validity of an eventually formula for the entire TS.
A1 and A3 are relatively easy (at least their syntactic version)

A2 can be as complex as checking the validity of an eventually formula for the entire TS.

Practically, cheaper but over-approximating static analyses are used to compute an ample set that satisfies A2.
CHEAP CONDITION CHECKING

☐ A1 and A3 are relatively easy (at least their syntactic version)

☐ A2 can be as complex as checking the validity of an eventually formula for the entire TS.

☐ Practically, cheaper but over-approximating static analyses are used to compute an ample set that satisfies A2.

☐ A4 is replaced by a stronger condition that implies it:
CHEAP CONDITION CHECKING

- A1 and A3 are relatively easy (at least their syntactic version)
- A2 can be as complex as checking the validity of an eventually formula for the entire TS.
- Practically, cheaper but over-approximating static analyses are used to compute an ample set that satisfies A2.
- A4 is replaced by a stronger condition that implies it:
  - A’4: in each cycle, in at least one state ample(s)=enabled(s).
CHEAP CONDITION CHECKING

- A1 and A3 are relatively easy (at least their syntactic version)

- A2 can be as complex as checking the validity of an eventually formula for the entire TS.

- Practically, cheaper but over-approximating static analyses are used to compute an ample set that satisfies A2.

- A4 is replaced by a stronger condition that implies it:

  - A'4: in each cycle, in at least one state ample(s)=enabled(s).

  - can be easily integrated in a depth-first search algorithm.
CHEAP CONDITION CHECKING

- A1 and A3 are relatively easy (at least their syntactic version).

- A2 can be as complex as checking the validity of an eventually formula for the entire TS.

- Practically, cheaper but over-approximating static analyses are used to compute an ample set that satisfies A2.

- A4 is replaced by a stronger condition that implies it:

  - A’4: in each cycle, in at least one state ample(s)=enabled(s).

  - can be easily integrated in a depth-first search algorithm.

See the book for the detailed algorithm!
MORE ON MODEL CHECKING
MAKING MODEL CHECKING SCALABLE

- Symbolic model checking
- Binary Decision Diagrams (BDDs)
- Bounded model checking
- Symmetry reduction
- Abstraction: we’ll see more on this
Symbolic Model Checking

Ordered binary decision diagrams (OBDDs) are a canonical form for boolean formulas.

\[ f(a_1, a_2, b_1, b_2) = (a_1 \leftrightarrow b_1) \land (a_2 \leftrightarrow b_2) \]
A transition system can be represented using OBDDs:
A transition system can be represented using OBDDs:

Assume that states are represented using $n$ boolean variables.

\[(v_1, \ldots, v_n) \rightarrow (v'_1, \ldots, v'_n)\]

can be replaced with a boolean formula representation:

\[T(v_1, \ldots, v_n, v'_1, \ldots, v'_n)\]
A transition system can be represented using OBDDs:

Assume that states are represented using \( n \) boolean variables.

\[
(v_1, \ldots, v_n) \rightarrow (v'_1, \ldots, v'_n)
\]

can be replaced with a boolean formula representation:

\[
T(v_1, \ldots, v_n, v'_1, \ldots, v'_n)
\]

Similarly, sets of states can be represented by OBDDs, including initial states.

\[
I_0(v_1, \ldots, v_n)
\]
A transition system can be represented using OBDDs:

Assume that states are represented using n boolean variables.

\[(v_1, \ldots, v_n) \rightarrow (v'_1, \ldots, v'_n)\]

can be replaced with a boolean formula representation:

\[T(v_1, \ldots, v_n, v'_1, \ldots, v'_n)\]

Similarly, sets of states can be represented by OBDDs, including initial states.

\[I_0(v_1, \ldots, v_n)\]

Invariants (set of reachable states) can be computed as the fixed point of the following equation:

\[I_n(v') = I_{n-1}(v') \lor [I_{n-1}(v') \land T(v, v')]\]
BOUNDED MODEL CHECKING
Now imagine not computing the fixed point for the expansion formula below:

\[ I_n(\vec{v}') = I_{n-1}(\vec{v}') \lor [I_{n-1}(\vec{v}) \land T(\vec{v}, \vec{v}')] \]
Now imagine not computing the fixed point for the expansion formula below:

\[ I_n(\vec{v}') = I_{n-1}(\vec{v}') \lor [I_{n-1}(\vec{v}) \land T(\vec{v}, \vec{v}')] \]

But, instead unrolling it for a given fixed depth.
Now imagine not computing the fixed point for the expansion formula below:

\[
I_n(\vec{v}') = I_{n-1}(\vec{v}') \lor [I_{n-1}(\vec{v}) \land T(\vec{v}, \vec{v}')] 
\]

But, instead unrolling it for a given fixed depth.

This under-approximates the set of reachable states.
Now imagine not computing the fixed point for the expansion formula below:

$$I_n(\vec{v}') = I_{n-1}(\vec{v}') \lor [I_{n-1}(\vec{v}) \land T(\vec{v}, \vec{v}')]$$

But, instead unrolling it for a given fixed depth.

This under-approximates the set of reachable states.

For certain properties (e.g. invariance checking), there are depths that guarantee the completeness of the check: diameter of the transition system.
Now imagine not computing the fixed point for the expansion formula below:

\[ I_n(\vec{v}') = I_{n-1}(\vec{v}') \lor [I_{n-1}(\vec{v}) \land T(\vec{v}, \vec{v}')] \]

But, instead unrolling it for a given fixed depth.

This under-approximates the set of reachable states.

For certain properties (e.g. invariance checking), there are depths that guarantee the completeness of the check: diameter of the transition system

SAT-solvers and standard boolean formulas are typically used (in place of OBBDs) for BMC.
A system with symmetry
SYMMETRY REDUCTION

A system with symmetry
A system with symmetry
SYMMETRY REDUCTION

Automorphism $h$ on state graph $G$ induces quotient graph $G_0$.

Original Transition System

Quotient Transition System