COMP2111 Week 8
Term 1, 2019
Regular languages and beyond II

Summary

- Regular expressions
- Myhill-Nerode theorem
- Context-free languages
- Mealy machines
- LTL: Logic for transition systems

Context-free grammars

Regular languages can be specified in terms of finite automata that accept or reject strings, equivalently, in terms of regular expressions, which strings are to match.
Grammars are a *generative* means of specifying sets of strings.

Context-free grammars (CFG): A way of generating words

Ingredients of a CFG:

$$(\{\text{variables}\}, \{\text{terminals}\}, \{\text{productions (or rules)}\}, \text{start symbol})$$

The start symbol is a special variable.
A CFG generates strings over the alphabet $\Sigma = \{\text{terminals}\}$.

Example

$G = (\{A, B\}, \{0, 1\}, \mathcal{R}, A)$ where $\mathcal{R}$ consists of three rules:

$$\begin{cases} A & \rightarrow & 0\,A\,1 \\ A & \rightarrow & B \\ B & \rightarrow & \epsilon \end{cases}$$

How to generate strings using a CFG

1. Set w to be the start symbol.
2. Choose an occurrence of a variable X in w if any, otherwise STOP.
3. Pick a production whose lhs is X, replace the chosen occurrence of X in w by the rhs.
4. GOTO 2.

Example

$G = (\{A, B\}, \{0, 1\}, \{A \to 0\,A\,1 \mid B, \quad B \to \epsilon\}, A)$ generates $\{0^i\,1^i : i \geq 0\}$.

$$\begin{aligned} A &\Rightarrow 0\,A\,1 \\ &\Rightarrow 0\,0\,A\,1\,1 \\ &\Rightarrow 0\,0\,B\,1\,1 \\ &\Rightarrow 0\,0\,\epsilon\,1\,1 = 0^2\,1^2 \end{aligned}$$

Such sequences are called **derivations**.

Formal definition

A **context-free grammar** is a 4-tuple $G = (V, \Sigma, \mathcal{R}, S)$ where

- V is a finite set of *variables* (or *non-terminals*)
- Σ (the alphabet) is a finite set of *terminals*
- $\mathcal{R}$ is a finite set of *productions*. A *production* (or *rule*) is an element of $V \times (V \cup \Sigma)^*$, written $A \to w$.
- $S \in V$ is the *start symbol*.

We define a binary relation $\Rightarrow$ over $(\{V \cup \Sigma\})^*$ by: for each $u, v \in (\{V \cup \Sigma\})^*$, for each $A \to w$ in $\mathcal{R}$

$$u\,A\,v \Rightarrow u\,w\,v$$

The **language generated by the grammar**, $L(G)$, is $\{w \in \Sigma^* : S \Rightarrow^* w\}$.

A language is **context-free** if it can be generated by a CFG.

Examples

Example

Well-balanced parentheses: generated by $(\{S\}, \{\,(\,,\,)\,\}, \mathcal{R}, S)$ where $\mathcal{R}$ consists of

$$S \;\rightarrow\; (\;S\;)\;|\,S\,S\,|\,\epsilon$$

E.g. $(\;(\;)\;(\;(\;)\;)\;)\;(\;)$

Examples

Example

Inductively defined syntax:

- Well-formed formulas
- $\mathcal{L}$
- Regular expressions
- Code specifications

WFFs: Generated by $(\{\varphi\}, \Sigma, \mathcal{R}, \varphi)$ where $\Sigma = \textsc{Prop} \cup \{\top, \bot, (,), \neg, \wedge, \vee, \rightarrow, \leftrightarrow\}$ and $\mathcal{R}$ consists of

$$\varphi \rightarrow \top \mid \bot \mid P \mid \neg\varphi \mid (\varphi \wedge \varphi) \mid (\varphi \vee \varphi) \mid (\varphi \rightarrow \varphi) \mid (\varphi \leftrightarrow \varphi)$$

Examples

Example

Inductively defined syntax:

- Well-formed formulas
- $\mathcal{L}$
- Regular expressions
- Code specifications

WFFs: Generated by $(\{\varphi\}, \Sigma, \mathcal{R}, \varphi)$ where
$\Sigma = \textsc{Prop} \cup \{\top, \bot, (,), \neg, \wedge, \vee, \to, \leftrightarrow\}$ and $\mathcal{R}$ consists of

$$\varphi \;\to\; \top \mid \bot \mid P \mid \neg\varphi \mid (\varphi \wedge \varphi) \mid (\varphi \vee \varphi) \mid (\varphi \to \varphi) \mid (\varphi \leftrightarrow \varphi)$$

Examples

Example

A small English language

$$
\begin{array}{rcl}
\langle\text{SENTENCE}\rangle & \rightarrow & \langle\text{NOUN-PHRASE}\rangle\ \langle\text{VERB-PHRASE}\rangle \\
\langle\text{NOUN-PHRASE}\rangle & \rightarrow & \langle\text{CMPLX-NOUN}\rangle \quad | \quad \langle\text{CMPLX-NOUN}\rangle\ \langle\text{PREP-PHRASE} \\
\langle\text{VERB-PHRASE}\rangle & \rightarrow & \langle\text{CMPLX-VERB}\rangle \quad | \quad \langle\text{CMPLX-VERB}\rangle\ \langle\text{PREP-PHRASE}\rangle \\
\langle\text{PREP-PHRASE}\rangle & \rightarrow & \langle\text{PREP}\rangle\ \langle\text{CMPLX-NOUN}\rangle \\
\langle\text{CMPLX-NOUN}\rangle & \rightarrow & \langle\text{ARTICLE}\rangle\ \langle\text{NOUN}\rangle \\
\langle\text{CMPLX-VERB}\rangle & \rightarrow & \langle\text{VERB}\rangle \quad | \quad \langle\text{VERB}\rangle\ \langle\text{NOUN-PHRASE}\rangle \\
\langle\text{ARTICLE}\rangle & \rightarrow & \mathsf{a} \quad | \quad \mathsf{the} \\
\langle\text{NOUN}\rangle & \rightarrow & \mathsf{boy} \quad | \quad \mathsf{girl} \quad | \quad \mathsf{flower} \\
\langle\text{VERB}\rangle & \rightarrow & \mathsf{touches} \quad | \quad \mathsf{like} \quad | \quad \mathsf{see} \\
\langle\text{PREP}\rangle & \rightarrow & \mathsf{with}
\end{array}
$$

Examples

Example

A small English language

$$
\begin{aligned}
\langle \text{SENTENCE} \rangle &\Rightarrow \langle \text{NOUN-PHRASE} \rangle \ \langle \text{VERB-PHRASE} \rangle \\
&\Rightarrow \langle \text{CMPLX-NOUN} \rangle \ \langle \text{PREP-PHRASE} \rangle \ \langle \text{VERB-PHRASE} \rangle \\
&\Rightarrow \langle \text{ARTICLE} \rangle \ \langle \text{NOUN} \rangle \ \langle \text{PREP-PHRASE} \rangle \ \langle \text{VERB-PHRASE} \rangle \\
&\Rightarrow \text{a girl } \langle \text{PREP} \rangle \ \langle \text{CMPLX-NOUN} \rangle \ \langle \text{VERB-PHRASE} \rangle \\
&\Rightarrow \text{a girl with } \langle \text{CMPLX-NOUN} \rangle \ \langle \text{VERB-PHRASE} \rangle \\
&\Rightarrow \text{a girl with } \langle \text{ARTICLE} \rangle \ \langle \text{NOUN} \rangle \ \langle \text{VERB-PHRASE} \rangle \\
&\Rightarrow \text{a girl with a flower } \langle \text{VERB-PHRASE} \rangle \\
&\Rightarrow \text{a girl with a flower } \langle \text{CMPLX-VERB} \rangle \\
&\Rightarrow \text{a girl with a flower } \langle \text{VERB} \rangle \ \langle \text{NOUN-PHRASE} \rangle \\
&\Rightarrow \text{a girl with a flower likes } \langle \text{CMPLX-NOUN} \rangle \\
&\Rightarrow \text{a girl with a flower likes } \langle \text{ARTICLE} \rangle \ \langle \text{NOUN} \rangle \\
&\Rightarrow \text{a girl with a flower likes the boy}
\end{aligned}
$$

Regular languages vs Context-free languages

A CFG is **right-linear** if every rule is either of the form $R \to wT$ or of the form $R \to w$ where w ranges over strings of terminals, and R and T over variables.

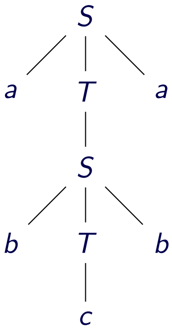
Theorem

A language is regular if and only if it is generated by a right-linear CFG.

Parse trees

Each derivation determines a **parse tree**.
Parse trees are *ordered* trees: the children at each node are ordered.
The parse tree of a derivation abstracts away from the order in which variables are replaced in the sequence.

$$
\begin{aligned}
S &\Rightarrow a\,T\,a \\
  &\Rightarrow a\,S\,a \\
  &\Rightarrow a\,b\,T\,b\,a \\
  &\Rightarrow a\,b\,c\,b\,a
\end{aligned}
$$



Properties of CFLs

Context-free languages are closed under union

Context-free languages are **not** closed under complement nor intersection

Properties of CFLs

Context-free languages are closed under union

Context-free languages are **not** closed under complement nor intersection

Pushdown automata

CFLs can be recognized by **Pushdown automata**:

- Non-deterministic finite automaton, PLUS
- Stack memory:
  - Infinite capacity for storing inputs
  - Can recover top-most memory item to influence transitions

Summary

- Regular expressions
- Myhill-Nerode theorem
- Context-free languages
- Mealy machines
- LTL: Logic for transition systems

Mealy machines

A **Mealy machine** is a finite state deterministic transducer.
Formally, a tuple $(Q, \Sigma, \Gamma, \delta, q_0)$ where

- Q is a finite set of states
- Σ is the input alphabet
- Γ is the output alphabet
- $\delta : Q \times \Sigma \to Q \times \Gamma$ is the transition function
- $q_0 \in Q$ is the start state.

DFAs accept languages, Mealy machines compute (length-preserving) functions $f(M) : \Sigma^* \to \Gamma^*$

Mealy machines

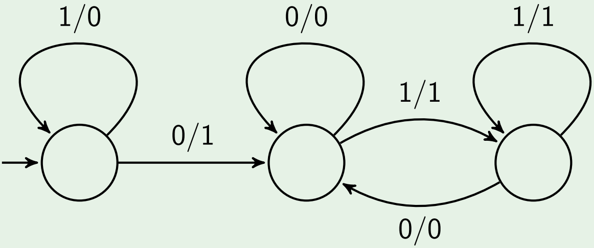
A **Mealy machine** is a finite state deterministic transducer.
Formally, a tuple $(Q, \Sigma, \Gamma, \delta, q_0)$ where

- Q is a finite set of states
- Σ is the input alphabet
- Γ is the output alphabet
- $\delta : Q \times \Sigma \to Q \times \Gamma$ is the transition function
- $q_0 \in Q$ is the start state.

DFAs accept languages, Mealy machines compute (length-preserving) functions $f(M) : \Sigma^* \to \Gamma^*$

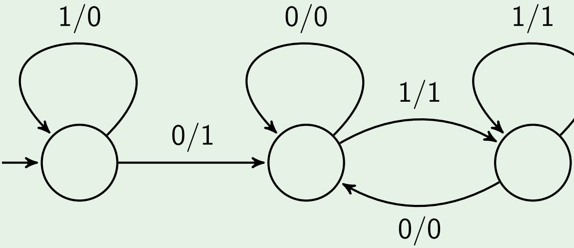
Example

Example



Example

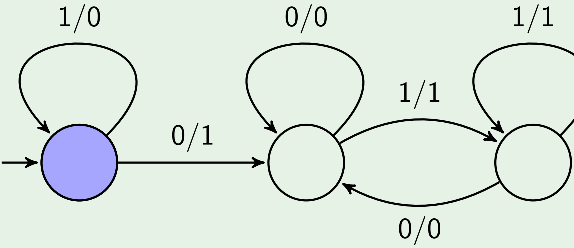
Example



1 1 0 1

Example

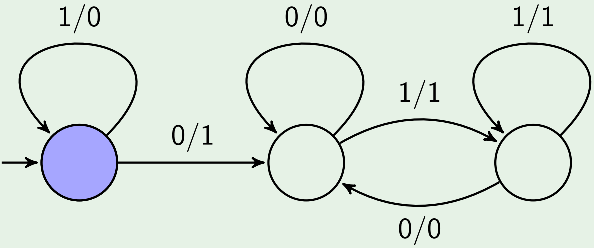
Example



1 1 0 1

Example

Example

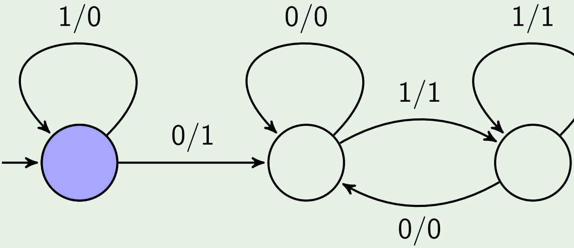


1 1 0 1

0

Example

Example

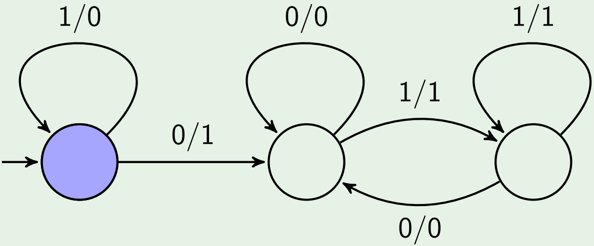


1 1 0 1

0

Example

Example

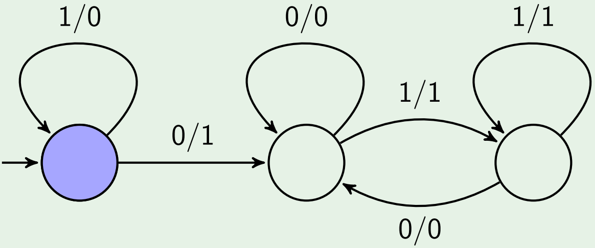


1 1 0 1

0 0

Example

Example

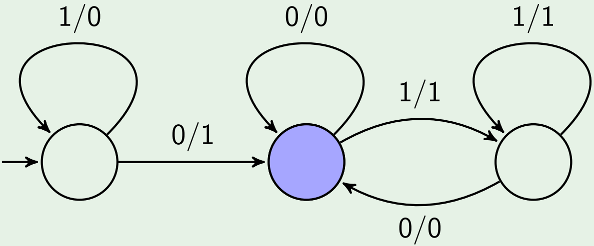


1 1 0 1

0 0

Example

Example

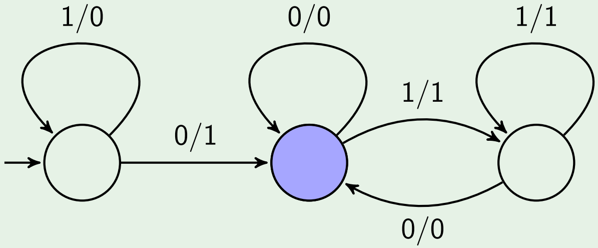


1 1 0 1

0 0 1

Example

Example

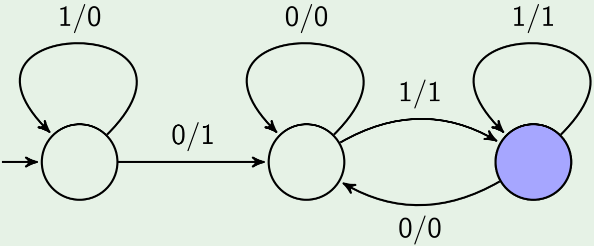


1 1 0 1

0 0 1

Example

Example



1 1 0 1

0 0 1 1

Applications

Mealy machines model input/output systems:

- “Black-box” investigation
- Substitution cipher
- Circuit analysis

Moore machines

Moore machines historically predate Mealy machines

Outputs occur at states rather than transitions

$\Rightarrow$ Better for synchronicity

Summary

- Regular expressions
- Myhill-Nerode theorem
- Context-free languages
- Mealy machines
- LTL: Logic for transition systems

Models of computation

Transition systems model a rudimentary form of computation

We would like to reason about them:

- Every request is eventually responded to
- The traffic light is not always red
- The brakes are applied until the pedal is released
- If φ holds in a state, then ψ will hold in the successor state

Specifications for transition systems

Model:

- Transition system with start state $(S, \rightarrow, s_0)$
- Set $\textsc{Prop}$ of propositional variables
- A labelling $\Lambda : S \rightarrow \mathrm{Pow}(\textsc{Prop})$ identifying which propositions hold in which state

NB

With propositions such as "just took transition labelled x" or "in a final state" we can cover other types of transition systems

First-order logic approach

Vocabulary:

- Domain of discourse: States
- A binary predicate, T
- A constant, s_0
- Unary predicates $P(x)$ for each $P \in \textsc{Prop}$

Example

The traffic light is never always red

could be specified as:

$$\forall x \exists y((y > x) \wedge \neg \mathrm{isRed}(y))$$

where

$$y > x \quad := \quad (x \to y) \vee \forall z.(T(x,z) \to (y > z))$$

First-order logic pros and cons

Pros:

- Very expressive specification language
- Satisfiability is decidable

Cons:

- “Readability” can be an issue
- Satisfiability checking is of non-elementary complexity

Linear Temporal Logic

Temporal logics, for reasoning about time, were developed around 1900

Linear Temporal Logic (LTL) introduced in 1970s by Pnueli.

LTL is an extension of Propositional Logic with the ability to work with state transitions

LTL is a restriction of Predicate Logic with limitations on various constructs such as quantifiers

LTL pros and cons

Pros:

- Logical language close to English
- Satisfiability checking is (relatively) efficient
- Quite expressive (same as FO on paths)

Cons:

- Does not take “branching” into account (see CTL)
- Satifiability is still computationally difficult

LTL Syntax

The **formulas of LTL** are defined recursively as follows:

- $\top$, $\bot$, p ($p \in \textsc{Prop}$) are all formulas;
- If φ, ψ are formulas then so are:
  - $\neg\varphi$
  - $(\varphi \wedge \psi), (\varphi \vee \psi), (\varphi \rightarrow \psi), (\varphi \leftrightarrow \psi)$
  - $\mathbf{X}\varphi$ [φ holds in the ne**X**t state]
  - $\varphi\mathbf{U}\psi$ [φ holds **U**ntil ψ holds]

NB

X and U are known as **temporal operators**.

Example

Some formulas:

- $\top\mathbf{U}(p \wedge q)$
- $\mathbf{X}(p \vee (q\mathbf{U}\neg p))$

LTL Syntax

The **formulas of LTL** are defined recursively as follows:

- $\top$, $\bot$, p ($p \in \textsc{Prop}$) are all formulas;
- If φ, ψ are formulas then so are:
  - $\neg\varphi$
  - $(\varphi \wedge \psi), (\varphi \vee \psi), (\varphi \rightarrow \psi), (\varphi \leftrightarrow \psi)$
  - $\mathbf{X}\varphi$ [φ holds in the ne**X**t state]
  - $\varphi\mathbf{U}\psi$ [φ holds **U**ntil ψ holds]

NB

X and U are known as **temporal operators**.

Example

Some formulas:

- $\top\mathbf{U}(p \wedge q)$
- $\mathbf{X}(p \vee (q\mathbf{U}\neg p))$

LTL Syntax

The **formulas of LTL** are defined recursively as follows:

- $\top$, $\bot$, p ($p \in \textsc{Prop}$) are all formulas;
- If φ, ψ are formulas then so are:
  - $\neg\varphi$
  - $(\varphi \wedge \psi), (\varphi \vee \psi), (\varphi \rightarrow \psi), (\varphi \leftrightarrow \psi)$
  - $\mathbf{X}\varphi$ [φ holds in the ne**X**t state]
  - $\varphi\mathbf{U}\psi$ [φ holds **U**ntil ψ holds]

NB

X and U are known as **temporal operators**.

Example

Some formulas:

- $\top\mathbf{U}(p \wedge q)$
- $\mathbf{X}(p \vee (q\mathbf{U}\neg p))$

LTL Syntax

The **formulas of LTL** are defined recursively as follows:

- $\top$, $\bot$, p ($p \in \textsc{Prop}$) are all formulas;
- If φ, ψ are formulas then so are:
  - $\neg\varphi$
  - $(\varphi \wedge \psi), (\varphi \vee \psi), (\varphi \to \psi), (\varphi \leftrightarrow \psi)$
  - $\mathbf{X}\varphi$ [φ holds in the ne**X**t state]
  - $\varphi\mathbf{U}\psi$ [φ holds **U**ntil ψ holds]

NB

X and U are known as ***temporal operators****.*

Example

Some formulas:

- $\top\mathbf{U}(p \wedge q)$
- $\mathbf{X}(p \vee (q\mathbf{U}\neg p))$

LTL Syntax: Derived operators

Three additional common operators:

- $\mathbf{F}\varphi$: eventually (in the **F**uture) φ, for $\top\mathbf{U}\varphi$
- $\mathbf{G}\varphi$: always (**G**lobally) φ, for $\neg(\top\mathbf{U}(\neg\varphi))$
- $\varphi\mathbf{W}\psi$: φ **W**eakly until ψ, for $(\mathbf{G}\varphi) \vee (\varphi\mathbf{U}\psi)$
- $\varphi\mathbf{R}\psi$: φ **R**eleases ψ, for $\neg(\neg\varphi\mathbf{U}\neg\psi)$

More examples

Example

- Every request is eventually responded to: $\mathbf{G}(\mathsf{req} \rightarrow \mathbf{F}\mathsf{resp})$
- The traffic light is not always red: $\neg\mathbf{G}\mathsf{red}$
- The brakes are applied until the pedal is released: $\mathsf{brake}\mathbf{U}(\neg\mathsf{pedal})$
- If φ holds in a state, then ψ will hold in the successor state: $\varphi \rightarrow \mathbf{X}\psi$

LTL Semantics

LTL is *linear*: it is interpreted by runs in the system.

Let $\rho = s_1, s_2, \ldots$ be a run in the transition system starting at s_1.
Let ρ_i denote the sub-run of ρ starting at i.
We define what it means for ρ to **satisfy** an LTL formula φ, written $\rho \models \varphi$, recursively as follows:

- $\rho \models \top$ for all runs, $\rho \not\models \bot$ for any run
- $\rho \models p$ if p is true in s_1
- $\rho \models \neg\varphi$ if it is not the case that $\rho \models \varphi$
- $\rho \models \varphi \wedge \psi$ if $\rho \models \varphi$ and $\rho \models \psi$
- $\rho \models \varphi \vee \psi$ if $\rho \models \varphi$ or $\rho \models \psi$
- $\rho \models \varphi \rightarrow \psi$ if it is the case that if $\rho \models \varphi$ then $\rho \models \psi$
- $\rho \models \varphi \leftrightarrow \psi$ if $\rho \models \varphi$ if and only if $\rho \models \psi$
- $\rho \models \mathbf{X}\varphi$ if $\rho_2 \models \varphi$
- $\rho \models \varphi \mathbf{U} \psi$ if there exists an i such that $\rho_j \models \varphi$ for all $1 \leq j < i$ and $\rho_i \models \psi$

LTL Semantics

LTL is *linear*: it is interpreted by runs in the system.

Let $\rho = s_1, s_2, \ldots$ be a run in the transition system starting at s_1.
Let ρ_i denote the sub-run of ρ starting at i.
We define what it means for ρ to **satisfy** an LTL formula φ, written $\rho \models \varphi$, recursively as follows:

- $\rho \models \top$ for all runs, $\rho \not\models \bot$ for any run
- $\rho \models p$ if p is true in s_1
- $\rho \models \neg\varphi$ if it is not the case that $\rho \models \varphi$
- $\rho \models \varphi \wedge \psi$ if $\rho \models \varphi$ and $\rho \models \psi$
- $\rho \models \varphi \vee \psi$ if $\rho \models \varphi$ or $\rho \models \psi$
- $\rho \models \varphi \rightarrow \psi$ if it is the case that if $\rho \models \varphi$ then $\rho \models \psi$
- $\rho \models \varphi \leftrightarrow \psi$ if $\rho \models \varphi$ if and only if $\rho \models \psi$
- $\rho \models \mathbf{X}\varphi$ if $\rho_2 \models \varphi$
- $\rho \models \varphi\mathbf{U}\psi$ if there exists an i such that $\rho_j \models \varphi$ for all $1 \leq j < i$ and $\rho_i \models \psi$

LTL Semantics

LTL is *linear*: it is interpreted by runs in the system.

Let $\rho = s_1, s_2, \ldots$ be a run in the transition system starting at s_1.
Let ρ_i denote the sub-run of ρ starting at i.
We define what it means for ρ to **satisfy** an LTL formula φ, written $\rho \models \varphi$, recursively as follows:

- $\rho \models \top$ for all runs, $\rho \not\models \bot$ for any run
- $\rho \models p$ if p is true in s_1
- $\rho \models \neg\varphi$ if it is not the case that $\rho \models \varphi$
- $\rho \models \varphi \wedge \psi$ if $\rho \models \varphi$ and $\rho \models \psi$
- $\rho \models \varphi \vee \psi$ if $\rho \models \varphi$ or $\rho \models \psi$
- $\rho \models \varphi \rightarrow \psi$ if it is the case that if $\rho \models \varphi$ then $\rho \models \psi$
- $\rho \models \varphi \leftrightarrow \psi$ if $\rho \models \varphi$ if and only if $\rho \models \psi$
- $\rho \models \mathbf{X}\varphi$ if $\rho_2 \models \varphi$
- $\rho \models \varphi\mathbf{U}\psi$ if there exists an i such that $\rho_j \models \varphi$ for all $1 \leq j < i$ and $\rho_i \models \psi$

LTL Semantics

LTL is *linear*: it is interpreted by runs in the system.

Let $\rho = s_1, s_2, \ldots$ be a run in the transition system starting at s_1.
Let ρ_i denote the sub-run of ρ starting at i.
We define what it means for ρ to **satisfy** an LTL formula φ, written $\rho \models \varphi$, recursively as follows:

- $\rho \models \top$ for all runs, $\rho \not\models \bot$ for any run
- $\rho \models p$ if p is true in s_1
- $\rho \models \neg\varphi$ if it is not the case that $\rho \models \varphi$
- $\rho \models \varphi \wedge \psi$ if $\rho \models \varphi$ and $\rho \models \psi$
- $\rho \models \varphi \vee \psi$ if $\rho \models \varphi$ or $\rho \models \psi$
- $\rho \models \varphi \rightarrow \psi$ if it is the case that if $\rho \models \varphi$ then $\rho \models \psi$
- $\rho \models \varphi \leftrightarrow \psi$ if $\rho \models \varphi$ if and only if $\rho \models \psi$
- $\rho \models \mathbf{X}\varphi$ if $\rho_2 \models \varphi$
- $\rho \models \varphi\mathbf{U}\psi$ if there exists an i such that $\rho_j \models \varphi$ for all $1 \leq j < i$ and $\rho_i \models \psi$

LTL Semantics

LTL is *linear*: it is interpreted by runs in the system.

Let $\rho = s_1, s_2, \ldots$ be a run in the transition system starting at s_1.
Let ρ_i denote the sub-run of ρ starting at i.
We define what it means for ρ to **satisfy** an LTL formula φ, written $\rho \models \varphi$, recursively as follows:

- $\rho \models \top$ for all runs, $\rho \not\models \bot$ for any run
- $\rho \models p$ if p is true in s_1
- $\rho \models \neg\varphi$ if it is not the case that $\rho \models \varphi$
- $\rho \models \varphi \wedge \psi$ if $\rho \models \varphi$ and $\rho \models \psi$
- $\rho \models \varphi \vee \psi$ if $\rho \models \varphi$ or $\rho \models \psi$
- $\rho \models \varphi \rightarrow \psi$ if it is the case that if $\rho \models \varphi$ then $\rho \models \psi$
- $\rho \models \varphi \leftrightarrow \psi$ if $\rho \models \varphi$ if and only if $\rho \models \psi$
- $\rho \models \mathbf{X}\varphi$ if $\rho_2 \models \varphi$
- $\rho \models \varphi\mathbf{U}\psi$ if there exists an i such that $\rho_j \models \varphi$ for all $1 \leq j < i$ and $\rho_i \models \psi$

LTL Semantics

LTL is *linear*: it is interpreted by runs in the system.

Let $\rho = s_1, s_2, \ldots$ be a run in the transition system starting at s_1.
Let ρ_i denote the sub-run of ρ starting at i.
We define what it means for ρ to **satisfy** an LTL formula φ, written $\rho \models \varphi$, recursively as follows:

- $\rho \models \top$ for all runs, $\rho \not\models \bot$ for any run
- $\rho \models p$ if p is true in s_1
- $\rho \models \neg\varphi$ if it is not the case that $\rho \models \varphi$
- $\rho \models \varphi \wedge \psi$ if $\rho \models \varphi$ and $\rho \models \psi$
- $\rho \models \varphi \vee \psi$ if $\rho \models \varphi$ or $\rho \models \psi$
- $\rho \models \varphi \rightarrow \psi$ if it is the case that if $\rho \models \varphi$ then $\rho \models \psi$
- $\rho \models \varphi \leftrightarrow \psi$ if $\rho \models \varphi$ if and only if $\rho \models \psi$
- $\rho \models \mathbf{X}\varphi$ if $\rho_2 \models \varphi$
- $\rho \models \varphi \mathbf{U} \psi$ if there exists an i such that $\rho_j \models \varphi$ for all $1 \leq j < i$ and $\rho_i \models \psi$

LTL Semantics

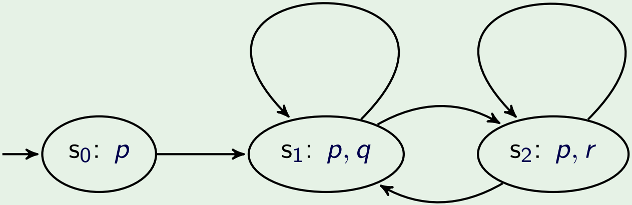
LTL is *linear*: it is interpreted by runs in the system.

Let $\rho = s_1, s_2, \ldots$ be a run in the transition system starting at s_1.
Let ρ_i denote the sub-run of ρ starting at i.
We define what it means for ρ to **satisfy** an LTL formula φ, written $\rho \models \varphi$, recursively as follows:

- $\rho \models \top$ for all runs, $\rho \not\models \bot$ for any run
- $\rho \models p$ if p is true in s_1
- $\rho \models \neg\varphi$ if it is not the case that $\rho \models \varphi$
- $\rho \models \varphi \wedge \psi$ if $\rho \models \varphi$ and $\rho \models \psi$
- $\rho \models \varphi \vee \psi$ if $\rho \models \varphi$ or $\rho \models \psi$
- $\rho \models \varphi \rightarrow \psi$ if it is the case that if $\rho \models \varphi$ then $\rho \models \psi$
- $\rho \models \varphi \leftrightarrow \psi$ if $\rho \models \varphi$ if and only if $\rho \models \psi$
- $\rho \models \mathbf{X}\varphi$ if $\rho_2 \models \varphi$
- $\rho \models \varphi\mathbf{U}\psi$ if there exists an i such that $\rho_j \models \varphi$ for all $1 \leq j < i$ and $\rho_i \models \psi$

Example

Example

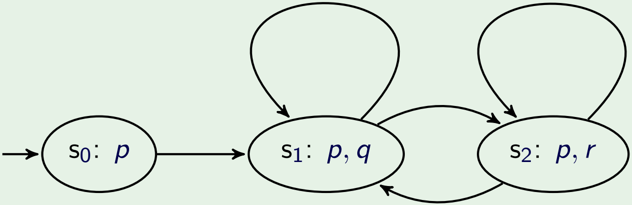


Consider the run $\rho = s_0, s_1, s_2, s_1, s_2, s_1, \ldots$. Does ρ satisfy the following formulas:

- $\mathbf{G}p$? Yes
- $\mathbf{X}q$? Yes
- $q\mathbf{U}r$? No
- $(\mathbf{X}q)\mathbf{W}r$? Yes
- $\mathbf{FG}q$? No

Example

Example

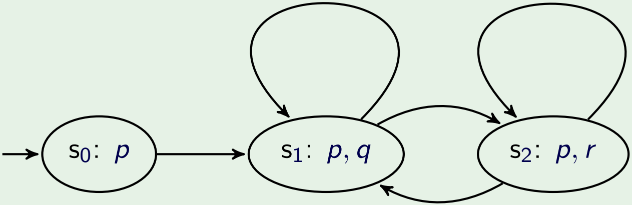


Consider the run $\rho = s_0, s_1, s_2, s_1, s_2, s_1, \ldots$. Does ρ satisfy the following formulas:

- $\mathbf{G}p$? Yes
- $\mathbf{X}q$? Yes
- $q\mathbf{U}r$? No
- $(\mathbf{X}q)\mathbf{W}r$? Yes
- $\mathbf{FG}q$? No

Example

Example

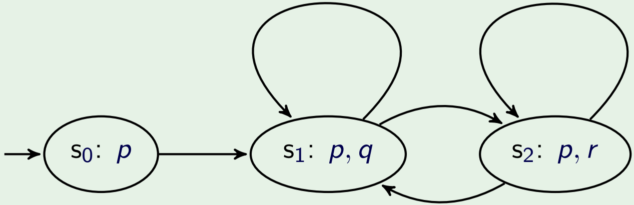


Consider the run $\rho = s_0, s_1, s_2, s_1, s_2, s_1, \ldots$. Does ρ satisfy the following formulas:

- $\mathbf{G}p$? Yes
- $\mathbf{X}q$? Yes
- $q\mathbf{U}r$? No
- $(\mathbf{X}q)\mathbf{W}r$? Yes
- $\mathbf{FG}q$? No

Example

Example

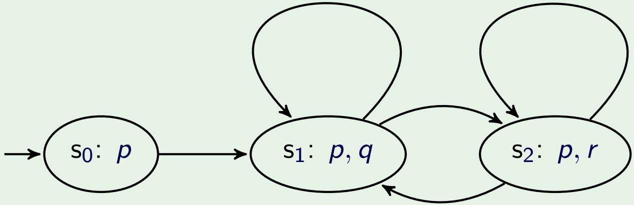


Consider the run $\rho = s_0, s_1, s_2, s_1, s_2, s_1, \ldots$. Does ρ satisfy the following formulas:

- $\mathbf{G}p$? Yes
- $\mathbf{X}q$? Yes
- $q\mathbf{U}r$? No
- $(\mathbf{X}q)\mathbf{W}r$? Yes
- $\mathbf{FG}q$? No

Example

Example

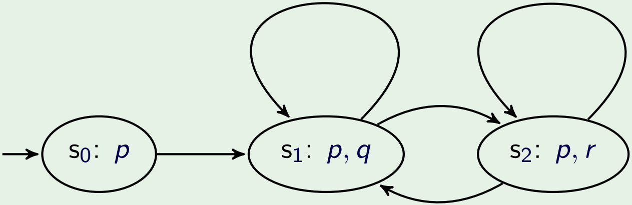


Consider the run $\rho = s_0, s_1, s_2, s_1, s_2, s_1, \ldots$. Does ρ satisfy the following formulas:

- $\mathbf{G}p$? Yes
- $\mathbf{X}q$? Yes
- $q\mathbf{U}r$? No
- $(\mathbf{X}q)\mathbf{W}r$? Yes
- $\mathbf{FG}q$? No

Example

Example

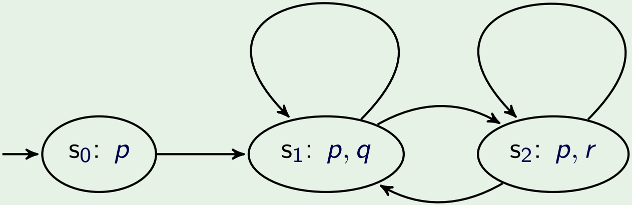


Consider the run $\rho = s_0, s_1, s_2, s_1, s_2, s_1, \ldots$. Does ρ satisfy the following formulas:

- $\mathbf{G}p$? Yes
- $\mathbf{X}q$? Yes
- $q\mathbf{U}r$? No
- $(\mathbf{X}q)\mathbf{W}r$? Yes
- $\mathbf{FG}q$? No

Example

Example

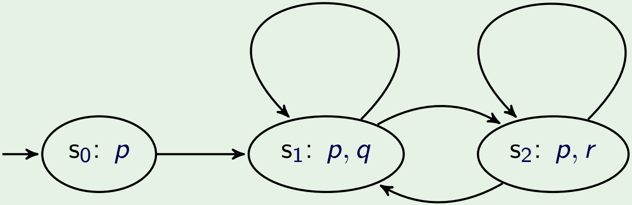


Consider the run $\rho = s_0, s_1, s_2, s_1, s_2, s_1, \ldots$. Does ρ satisfy the following formulas:

- $\mathbf{G}p$? Yes
- $\mathbf{X}q$? Yes
- $q\mathbf{U}r$? No
- $(\mathbf{X}q)\mathbf{W}r$? Yes
- $\mathbf{FG}q$? No

Example

Example



Consider the run $\rho = s_0, s_1, s_2, s_1, s_2, s_1, \ldots$. Does ρ satisfy the following formulas:

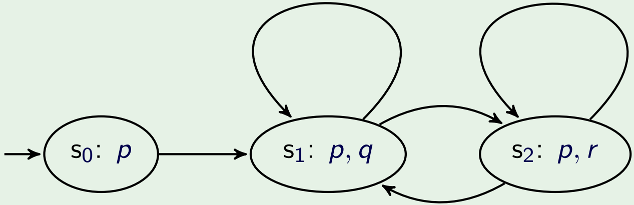
- $\mathbf{G}p$? Yes
- $\mathbf{X}q$? Yes
- $q\mathbf{U}r$? No
- $(\mathbf{X}q)\mathbf{W}r$? Yes
- $\mathbf{FG}q$? No

Semantics continued

A transition system with start state s_0 **satisfies** an LTL formula φ if $\rho \models \varphi$ for **all** runs starting at s_0.

Example

Example

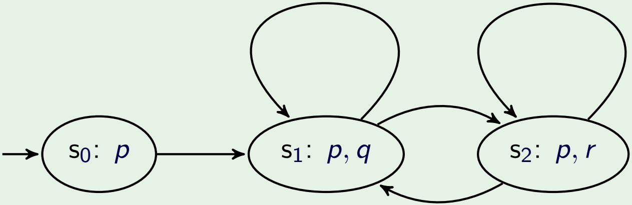


Does the system satisfy the following formulas:

- $\mathbf{G}p$? Yes
- $\mathbf{X}q$? Yes
- $(\mathbf{X}q)\mathbf{W}r$? Yes
- $\mathbf{GF}q$? No: e.g. $s_0, s_1, s_2, s_2, s_2 \ldots$
- $\mathbf{XGXX}q$? No

Example

Example

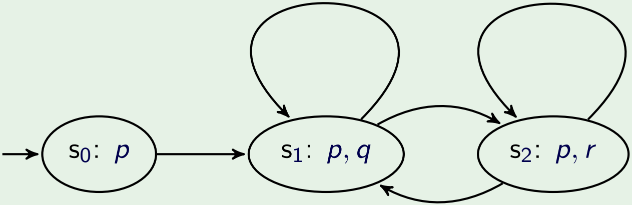


Does the system satisfy the following formulas:

- $\mathbf{G}p$? Yes
- $\mathbf{X}q$? Yes
- $(\mathbf{X}q)\mathbf{W}r$? Yes
- $\mathbf{GF}q$? No: e.g. $s_0, s_1, s_2, s_2, s_2 \ldots$
- $\mathbf{XGXX}q$? No

Example

Example

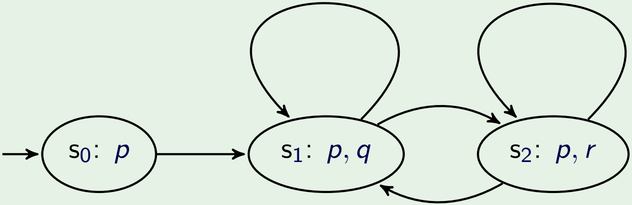


Does the system satisfy the following formulas:

- $\mathbf{G}p$? Yes
- $\mathbf{X}q$? Yes
- $(\mathbf{X}q)\mathbf{W}r$? Yes
- $\mathbf{GF}q$? No: e.g. $s_0, s_1, s_2, s_2, s_2 \ldots$
- $\mathbf{XGXX}q$? No

Example

Example

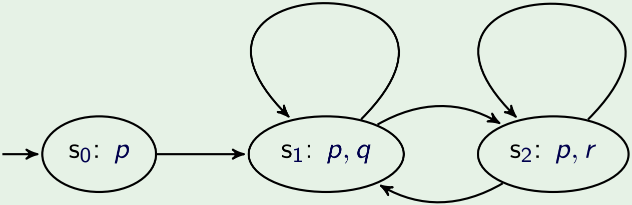


Does the system satisfy the following formulas:

- $\mathbf{G}p$? Yes
- $\mathbf{X}q$? Yes
- $(\mathbf{X}q)\mathbf{W}r$? Yes
- $\mathbf{GF}q$? No: e.g. $s_0, s_1, s_2, s_2, s_2 \ldots$
- $\mathbf{XGXX}q$? No

Example

Example

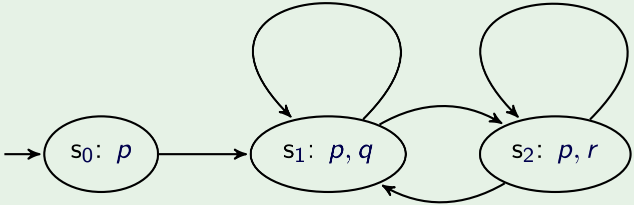


Does the system satisfy the following formulas:

- $\mathbf{G}p$? Yes
- $\mathbf{X}q$? Yes
- $(\mathbf{X}q)\mathbf{W}r$? Yes
- $\mathbf{GF}q$? No: e.g. $s_0, s_1, s_2, s_2, s_2 \ldots$
- $\mathbf{XGXX}q$? No

Example

Example



Does the system satisfy the following formulas:

- $\mathbf{G}p$? Yes
- $\mathbf{X}q$? Yes
- $(\mathbf{X}q)\mathbf{W}r$? Yes
- $\mathbf{GF}q$? No: e.g. $s_0, s_1, s_2, s_2, s_2 \ldots$
- $\mathbf{XGXX}q$? No

Relation to earlier concepts

- φ is a preserved invariant: $\varphi \to \mathbf{X}\varphi$
- The invariant principle: $(\varphi \to \mathbf{X}\varphi) \to \mathbf{G}\varphi$
- Safety: $\mathbf{G}\varphi$ (also $\mathbf{FG}\varphi$)
- Liveness: $\mathbf{F}\varphi$ (also $\mathbf{GF}\varphi$)

Deciding satisfiability I

On finitely presented transition systems, to decide if the system satisfies φ:

- Use φ to create a Büchi automaton (NFA for infinite words)
- Check if all runs of the system are accepted by the automaton

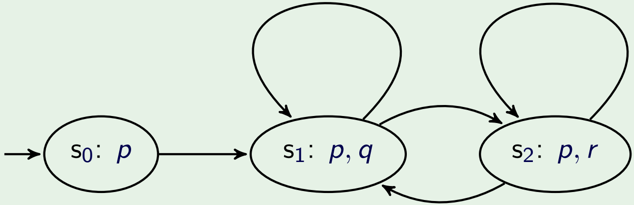
Deciding satisfiability II

On finitely presented transition systems, to decide if the system satisfies φ:

- Set up a two-player game: Verifier (System) vs Falsifier (Environment):
  - Verifier is trying to show the system satisfies the formula,
  - Falsifier is trying to show the system does not satisfy the formula
- Game is played on $\textsc{States} \times \textsc{Subformulas}$
- Players choose the next state or subformula to move to, which player chooses depends on the formula type, e.g.
  - Current formula is $X\varphi$, then Falsifier chooses a successor state and the formula becomes φ.
  - Current formula is $\varphi \vee \psi$, then Verifier chooses a subformula φ or ψ and the state remains the same.
  - Current formula is $\neg\varphi$, then Verifier and Falsifier swap roles and the formula becomes φ in the current state.
- Game continues until a propositional variable is reached, winner determined by whether that variable holds in the current state.

Example

Example

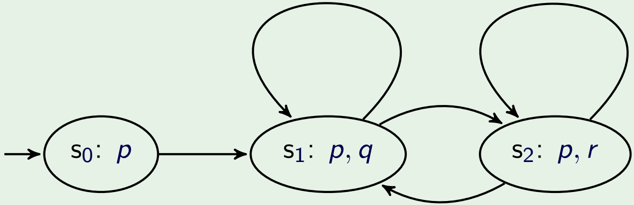


Show that this system does not satisfy $(\mathbf{X}q) \wedge (\mathbf{XX}r)$

- Current state: s_0
- $\varphi \wedge \psi$: Falsifier chooses subformula:
- $\mathbf{X}\varphi$: Falsifier chooses successor:
- $\mathbf{X}\varphi$: Falsifier chooses successor:
- r is not true in s_1 so Falsifier wins

Example

Example

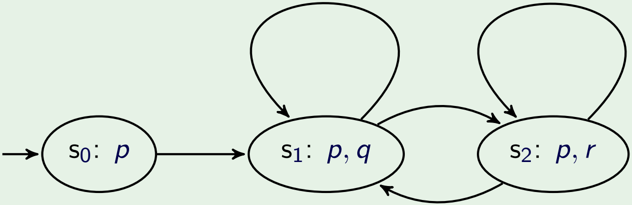


Show that this system does not satisfy $(\mathbf{X}q) \wedge (\mathbf{XX}r)$

- Current state: s_0
- $\varphi \wedge \psi$: Falsifier chooses subformula: say $(\mathbf{XX}r)$
- $\mathbf{X}\varphi$: Falsifier chooses successor:
- $\mathbf{X}\varphi$: Falsifier chooses successor:
- r is not true in s_1 so Falsifier wins

Example

Example

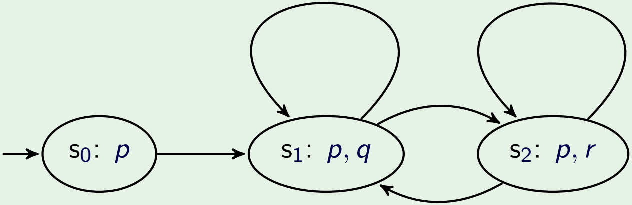


Show that this system does not satisfy $(\mathbf{X}q) \wedge (\mathbf{XX}r)$

- Current state: s_0
- $\varphi \wedge \psi$: Falsifier chooses subformula: say $(\mathbf{XX}r)$
- $\mathbf{X}\varphi$: Falsifier chooses successor:
- $\mathbf{X}\varphi$: Falsifier chooses successor:
- r is not true in s_1 so Falsifier wins

Example

Example

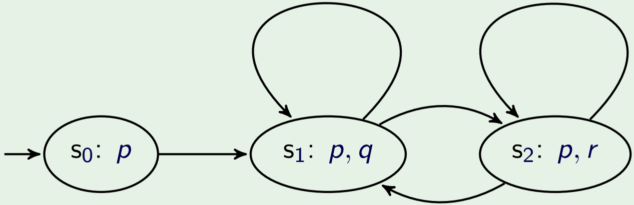


Show that this system does not satisfy $(\mathbf{X}q) \wedge (\mathbf{XX}r)$

- Current state: s_0
- $\varphi \wedge \psi$: Falsifier chooses subformula: say $(\mathbf{XX}r)$
- $\mathbf{X}\varphi$: Falsifier chooses successor: say s_1. Current formula: $\mathbf{X}r$
- $\mathbf{X}\varphi$: Falsifier chooses successor:
- r is not true in s_1 so Falsifier wins

Example

Example

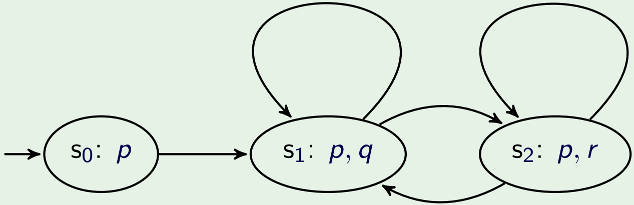


Show that this system does not satisfy $(\mathbf{X}q) \wedge (\mathbf{XX}r)$

- Current state: s_0
- $\varphi \wedge \psi$: Falsifier chooses subformula: say $(\mathbf{XX}r)$
- $\mathbf{X}\varphi$: Falsifier chooses successor: say s_1. Current formula: $\mathbf{X}r$
- $\mathbf{X}\varphi$: Falsifier chooses successor:
- r is not true in s_1 so Falsifier wins

Example

Example

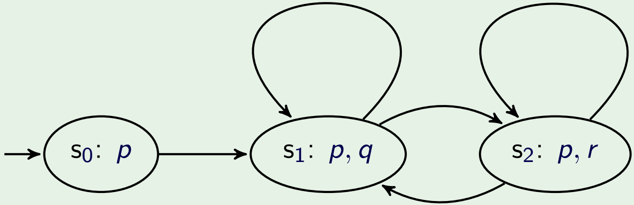


Show that this system does not satisfy $(\mathbf{X}q) \wedge (\mathbf{X}\mathbf{X}r)$

- Current state: s_0
- $\varphi \wedge \psi$: Falsifier chooses subformula: say $(\mathbf{X}\mathbf{X}r)$
- $\mathbf{X}\varphi$: Falsifier chooses successor: say s_1. Current formula: $\mathbf{X}r$
- $\mathbf{X}\varphi$: Falsifier chooses successor: say s_1. Current formula: r
- r is not true in s_1 so Falsifier wins

Example

Example

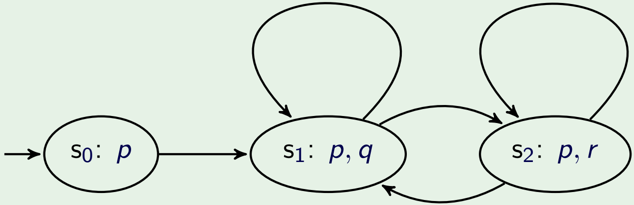


Show that this system does not satisfy $(\mathbf{X}q) \wedge (\mathbf{XX}r)$

- Current state: s_0
- $\varphi \wedge \psi$: Falsifier chooses subformula: say $(\mathbf{XX}r)$
- $\mathbf{X}\varphi$: Falsifier chooses successor: say s_1. Current formula: $\mathbf{X}r$
- $\mathbf{X}\varphi$: Falsifier chooses successor: say s_1. Current formula: r
- r is not true in s_1 so Falsifier wins

Example

Example

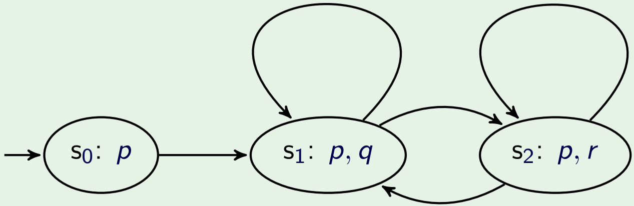


Show that this system does not satisfy $(\mathbf{X}q) \wedge (\mathbf{X}\mathbf{X}r)$

- Current state: s_0
- $\varphi \wedge \psi$: Falsifier chooses subformula: say $(\mathbf{X}\mathbf{X}r)$
- $\mathbf{X}\varphi$: Falsifier chooses successor: say s_1. Current formula: $\mathbf{X}r$
- $\mathbf{X}\varphi$: Falsifier chooses successor: say s_1. Current formula: r
- r is not true in s_1 so Falsifier wins

Example

Example

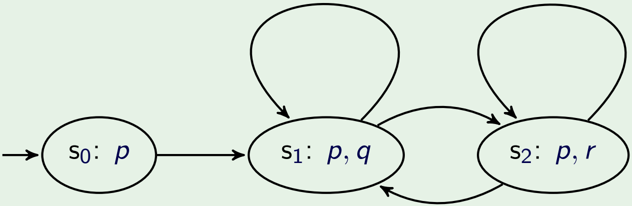


Show that this system does satisfy $(\mathbf{X}q) \vee (\mathbf{XX}r)$

- Current state: s_0
- $\varphi \vee \psi$: Verifier chooses subformula: say $(\mathbf{X}q)$
- $\mathbf{X}\varphi$: Falsifier chooses successor:
- q is true in s_1 so Verifier wins

Example

Example

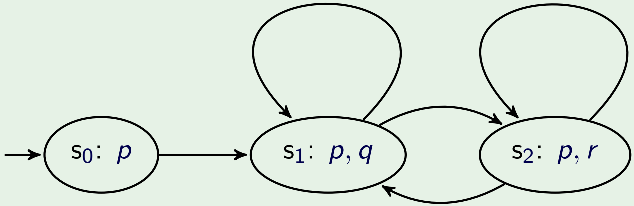


Show that this system does satisfy $(\mathbf{X}q) \vee (\mathbf{XX}r)$

- Current state: s_0
- $\varphi \vee \psi$: Verifier chooses subformula: say $(\mathbf{X}q)$
- $\mathbf{X}\varphi$: Falsifier chooses successor:
- q is true in s_1 so Verifier wins

Example

Example

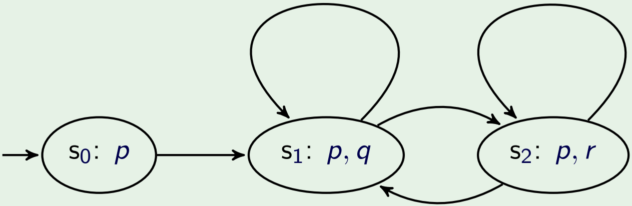


Show that this system does satisfy $(\mathbf{X}q) \vee (\mathbf{XX}r)$

- Current state: s_0
- $\varphi \vee \psi$: Verifier chooses subformula: say $(\mathbf{X}q)$
- $\mathbf{X}\varphi$: Falsifier chooses successor: say s_1. Current formula: q
- q is true in s_1 so Verifier wins

Example

Example

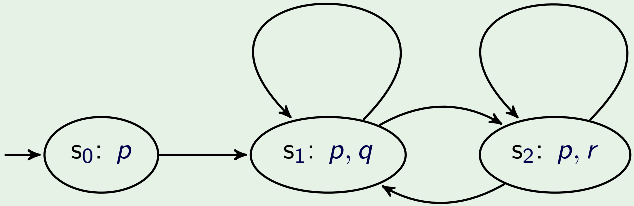


Show that this system does satisfy $(\mathbf{X}q) \vee (\mathbf{XX}r)$

- Current state: s_0
- $\varphi \vee \psi$: Verifier chooses subformula: say $(\mathbf{X}q)$
- $\mathbf{X}\varphi$: Falsifier chooses successor: say s_1. Current formula: q
- q is true in s_1 so Verifier wins

Example

Example



Show that this system does satisfy $(\mathbf{X}q) \vee (\mathbf{XX}r)$

- Current state: s_0
- $\varphi \vee \psi$: Verifier chooses subformula: say $(\mathbf{X}q)$
- $\mathbf{X}\varphi$: Falsifier chooses successor: say s_1. Current formula: q
- q is true in s_1 so Verifier wins

Expressiveness

In general, first-order logic is more expressive than LTL: e.g. LTL cannot express

on all runs: $p \rightarrow$ (there is a successor such that: q)

On transition systems that are just paths:

Theorem (Kamp,Pnueli)

On linear transition systems, every First-order formula is logically equivalent to a formula in LTL.

Expressiveness

In general, first-order logic is more expressive than LTL: e.g. LTL cannot express

on all runs: $p \to$ (there is a successor such that: q)

On transition systems that are just paths:

Theorem (Kamp,Pnueli)

On linear transition systems, every First-order formula is logically equivalent to a formula in LTL.

Summary

- Regular expressions
- Myhill-Nerode theorem
- Context-free languages
- Mealy machines
- LTL: Logic for transition systems