

Formal Methods and Functional Programming

Linear Temporal Logic

Peter Müller

Chair of Programming Methodology
ETH Zurich

The slides in this section are partly based on the course *Automata-based System Analysis* by
Felix Klaedtke

Motivation

- Many interesting properties relate several states
- Example: all opened files must be closed eventually

Motivation

- Many interesting properties relate several states
- Example: all opened files must be closed eventually
 - For a **terminating** program s

$$\langle s, \sigma \rangle \rightarrow_1^* \sigma' \text{ and } \sigma(o) = 0 \text{ then } \sigma'(o) = 0$$

Motivation

- Many interesting properties relate several states
- Example: all opened files must be closed eventually
 - For a **terminating** program s

$$\langle s, \sigma \rangle \rightarrow_1^* \sigma' \text{ and } \sigma(o) = 0 \text{ then } \sigma'(o) = 0$$

- For a **deterministic, non-terminating** program s

$$\langle s, \sigma \rangle \rightarrow_1^* \langle s', \sigma' \rangle \text{ and } \sigma(o) = 0 \text{ and } \sigma'(o) = 1 \text{ then there exist } s'', \sigma'' \text{ such that } \langle s', \sigma' \rangle \rightarrow_1^* \langle s'', \sigma'' \rangle \text{ and } \sigma''(o) = 0$$

Motivation

- Many interesting properties relate several states
- Example: all opened files must be closed eventually
 - For a **terminating** program s

$$\langle s, \sigma \rangle \rightarrow_1^* \sigma' \text{ and } \sigma(o) = 0 \text{ then } \sigma'(o) = 0$$

- For a **deterministic, non-terminating** program s

$$\langle s, \sigma \rangle \rightarrow_1^* \langle s', \sigma' \rangle \text{ and } \sigma(o) = 0 \text{ and } \sigma'(o) = 1 \text{ then there exist } s'', \sigma'' \text{ such that } \langle s', \sigma' \rangle \rightarrow_1^* \langle s'', \sigma'' \rangle \text{ and } \sigma''(o) = 0$$

- For a **non-deterministic, non-terminating** program s

$$wc : \text{Stm} \times \text{State} \times \mathbb{N} \rightarrow \text{Bool}$$

$$wc(s, \sigma, n) \Leftrightarrow \sigma(o) = 0 \vee$$

$$(\text{for all } s', \sigma' : \text{if } \langle s, \sigma \rangle \rightarrow_1 \langle s', \sigma' \rangle \text{ then there exists } m \in \mathbb{N} \text{ such that } m < n \text{ and } wc(s', \sigma', m))$$

$$\langle s, \sigma \rangle \rightarrow_1^* \langle s', \sigma' \rangle \text{ and } \sigma(o) = 0 \text{ and } \sigma'(o) = 1 \text{ then there exists } n \in \mathbb{N} \text{ such that } wc(s', \sigma', n)$$

5. Linear Temporal Logic

5.1 Linear-Time Properties

5.2 Linear Temporal Logic

5.3 LTL Model Checking

Transition Systems Revisited

- We use a slightly different definition here
- A finite transition system is a tuple $(\Gamma, \sigma_I, \rightarrow)$
 - Γ : a **finite** set of configurations
 - σ_I : an **initial configuration**, $\sigma_I \in \Gamma$
 - \rightarrow : a transition relation, $\rightarrow \subseteq \Gamma \times \Gamma$
- We add an initial configuration
 - Transition system models only one system, not all programs of a programming language
- We omit terminal configurations
 - Simplifies theory
 - Termination can be modelled by special sink state

Transition System of a Promela Model

- Configurations: states (see previous section)
 - Global variables, global channels
 - Per active process: local variables, local channels, location counter
- Initial configuration: initial state (see previous section)
- Transition relation: defined by operational semantics of statements
 - We keep semantics informal
- A Promela model has a finite number of states
 - Finite number of active processes (limited to 255)
 - Finite number of variables and channels
 - Finite ranges of variables
 - Finite buffers of channels
- Therefore, it is possible to enumerate all possible states
 - How many states are there?

State Space of Sequential Programs

- Number of states

$$\# \text{program locations} \times \prod_{\text{variable } x} | \text{dom}(x) |$$

- where $| \text{dom}(x) |$ denotes the number of possible values of variable x

State Space of Sequential Programs

- Number of states

$$\# \text{program locations} \times \prod_{\text{variable } x} | \text{dom}(x) |$$

- where $| \text{dom}(x) |$ denotes the number of possible values of variable x
- Example: sequential program with 10 locations and 3 boolean variables

$$10 \times 2 \times 2 \times 2 = 10 \times 2^3 = 80$$

- Adding two integer variables yields $80 \times 2^{32} \times 2^{32} = 80 \times 2^{64}$

State Space of Sequential Programs

- Number of states

$$\# \text{program locations} \times \prod_{\text{variable } x} | \text{dom}(x) |$$

- where $| \text{dom}(x) |$ denotes the number of possible values of variable x
- Example: sequential program with 10 locations and 3 boolean variables

$$10 \times 2 \times 2 \times 2 = 10 \times 2^3 = 80$$

- Adding two integer variables yields $80 \times 2^{32} \times 2^{32} = 80 \times 2^{64}$
- Number of states grows **exponentially** in the number of variables
- **State space explosion**

State Space of Concurrent Programs

- The number of states of $P \equiv P_1 \parallel \dots \parallel P_N$ is at most

$$\begin{aligned} & \# \text{states of } P_1 \times \dots \times \# \text{states of } P_N = \\ & \prod_{i=1}^N (\# \text{program locations}_i \times \prod_{\text{variable } x_i} | \text{dom}(x_i) |) \end{aligned}$$

State Space of Concurrent Programs

- The number of states of $P \equiv P_1 \parallel \dots \parallel P_N$ is at most

$$\begin{aligned} & \# \text{states of } P_1 \times \dots \times \# \text{states of } P_N = \\ & \prod_{i=1}^N (\# \text{program locations}_i \times \prod_{\text{variable } x_i} | \text{dom}(x_i) |) \end{aligned}$$

- Number of states grows **exponentially** in the number of processes
- **State space explosion**

State Space of Promela Models

- The number of states of a system with N processes and K channels is at most

$$\prod_{i=1}^N (\# \text{program locations}_i \times \prod_{\text{variable } x_i} | \text{dom}(x_i) |) \times \prod_{j=1}^K | \text{dom}(c_j) |^{cap(c_j)}$$

- $| \text{dom}(c) |$ denotes the number of possible messages of channel c
- $cap(c)$ is the capacity (buffer size) of channel c

State Space of Promela Models

- The number of states of a system with N processes and K channels is at most

$$\prod_{i=1}^N (\# \text{program locations}_i \times \prod_{\text{variable } x_i} | \text{dom}(x_i) |) \times \prod_{j=1}^K | \text{dom}(c_j) |^{cap(c_j)}$$

- $| \text{dom}(c) |$ denotes the number of possible messages of channel c
- $cap(c)$ is the capacity (buffer size) of channel c
- Number of states grows **exponentially** in the number and capacity of channels
- **State space explosion**

Computations

- Infinite sequences
 - S^ω is the set of infinite sequences of elements of set S
 - s_i denotes the i -th element of the sequence $s \in S^\omega$
- $\gamma \in \Gamma^\omega$ is a **computation** of a transition system if:
 - $\gamma_0 = \sigma_I$
 - $\gamma_i \rightarrow \gamma_{i+1}$
- $\mathcal{C}(TS)$ is the set of all computations of a transition system TS

Linear-Time Properties

- Linear-time properties (LT-properties) specify the **admissible computations of a transition system**
- A **linear-time property over Γ** is a subset of Γ^ω
- TS **satisfies** LT-property P (over Γ)

$$TS \models P \text{ if and only if } \mathcal{C}(TS) \subseteq P$$

- **All computations** of TS are admissible
- By contrast: branching-time properties can also express the **existence of a computation**
 - Example: “It is always possible to return to the initial state”

LT-Properties: Example

- All opened files must be closed eventually

$$P = \{\gamma \in \Gamma^\omega \mid \forall i \geq 0 : \gamma_i(o) = 1 \Rightarrow \exists n > 0 : \gamma_{i+n}(o) = 0\}$$

- LT-properties elegantly express properties of computations
 - Non-termination is handled by infinite sequences
 - Non-determinism is handled by considering each computation separately
- Logical formalism needed to simplify specification of LT-properties

From States to Propositions

- For a transition system TS , we specify the set AP of **atomic propositions**
 - An atomic proposition is a proposition containing no logical connectives
 - Example: $AP = \{open, closed\}$
- We define a **labeling function** that maps configurations to sets of atomic propositions
 - $L : \Gamma \rightarrow \mathcal{P}(AP)$
 - Example:
$$L(\sigma) = \begin{cases} \{open\} & \text{if } \sigma(o) = 1 \\ \{closed\} & \text{if } \sigma(o) = 0 \\ \{\} & \text{otherwise} \end{cases}$$
- We call $L(\sigma)$ an **abstract state**
- From now on, we consider AP and L to be part of the transition system

Traces

- A trace is an abstraction of a computation
 - Observe only the propositions of each state, not the concrete state itself
 - Infinite sequence of abstract states $(\mathcal{P}(AP)^\omega)$
- $t \in \mathcal{P}(AP)^\omega$ is a **trace of a transition system TS** if
 $t = L(\gamma_0)L(\gamma_1)L(\gamma_2), \dots$ and γ is a computation of TS
- $\mathcal{T}(TS)$ is the set of all traces of a transition system TS
- LT-properties are typically specified on traces

$$P = \{t \in \mathcal{P}(AP)^\omega \mid \forall i \geq 0 : open \in t_i \Rightarrow \exists n > 0 : closed \in t_{i+n}\}$$

Safety Properties

- Intuition
 - “nothing bad ever happens”
 - “if something bad happens then it is irremediable”
- An LT-property P is a safety property if for all traces $t \in \mathcal{P}(AP)^\omega$: if $t \notin P$ then there is a finite prefix \hat{t} of t such that for each trace t' with prefix \hat{t} , $t' \notin P$
 - \hat{t} is called a **bad prefix**
- Safety properties are **violated in finite time** and cannot be repaired
- Examples
 - State properties, for instance, invariants

$$P = \{t \in \mathcal{P}(AP)^\omega \mid \forall i \geq 0 : open \in t_i \vee closed \in t_i\}$$

- “Money can be withdrawn only after correct PIN has been entered”

Liveness Properties

- Intuition
 - “something good will happen eventually”
 - “if the good thing has not happened yet, it will happen in the future”
 - An LT-property P is a liveness property if every finite sequence $\hat{t} \in \mathcal{P}(AP)^*$ is a prefix of a trace $t \in P$
 - A liveness property does not rule out any prefix
 - Every finite prefix can be extended to a trace that is in P
 - Liveness properties are **violated in infinite time**
 - Examples
 - All opened files must be closed eventually
- $$P = \{t \in \mathcal{P}(AP)^\omega \mid \forall i \geq 0 : open \in t_i \Rightarrow \exists n > 0 : closed \in t_{i+n}\}$$
- “The program terminates eventually”

5. Linear Temporal Logic

5.1 Linear-Time Properties

5.2 Linear Temporal Logic

5.3 LTL Model Checking

Linear Temporal Logic

- Linear Temporal Logic (LTL) allows us to formalize LT-properties of traces
- We will discuss syntax and semantics, but not inference rules
- Whether the traces of a finite transition system satisfy an LTL formula is **decidable**

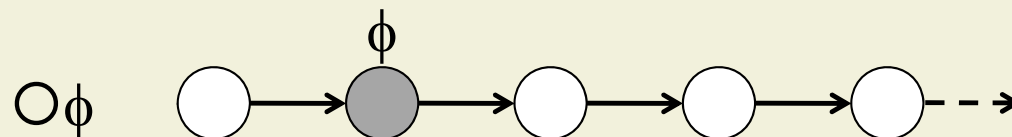
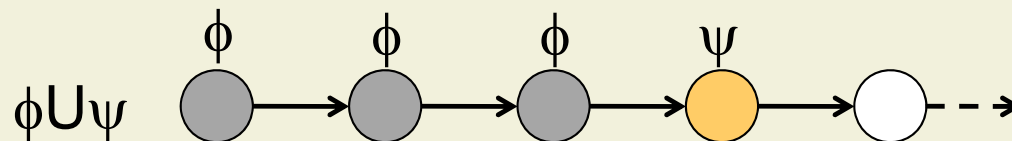
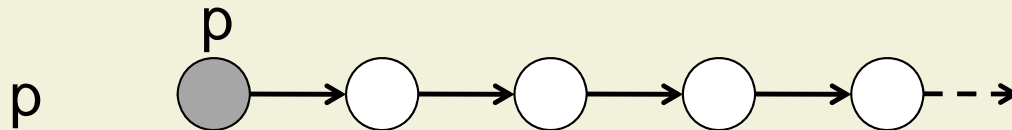
LTL: Basic Operators

- Syntax

$$\phi = p \mid \neg\phi \mid \phi \wedge \phi \mid \phi \mathbf{U} \phi \mid \bigcirc\phi$$

- where p is a proposition in $AP \neq \emptyset$

- Intuitive meaning of temporal operators

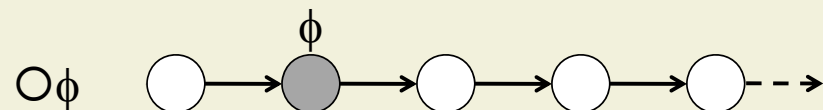
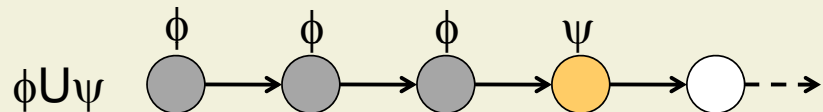
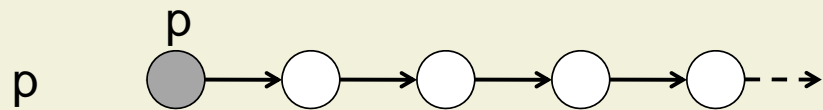


LTL: Semantics

- $t \models \phi$ expresses that trace $t \in \mathcal{P}(AP)^\omega$ satisfies LTL formula ϕ

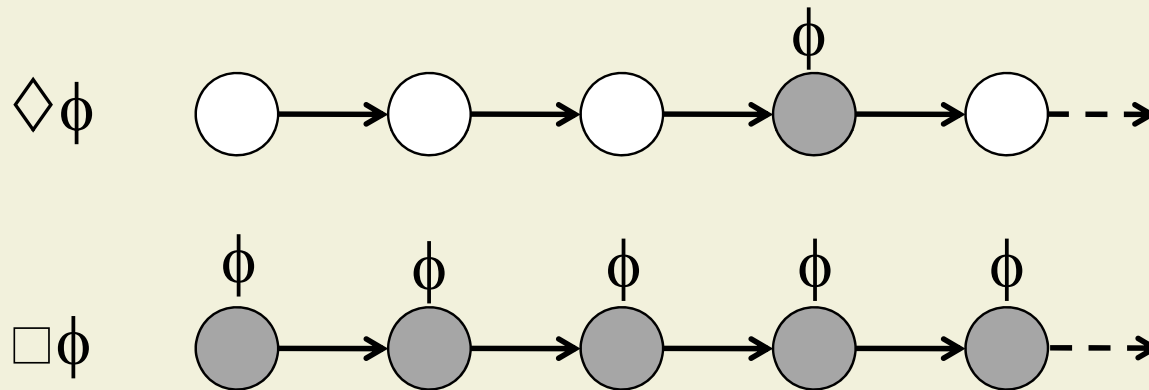
$t \models p$	iff	$p \in t_0$
$t \models \neg\phi$	iff	not $t \models \phi$
$t \models \phi \wedge \psi$	iff	$t \models \phi$ and $t \models \psi$
$t \models \phi \cup \psi$	iff	there is a $k \geq 0$ with $t_{\geq k} \models \psi$ and $t_{\geq j} \models \phi$ for $0 \leq j < k$
$t \models \bigcirc\phi$	iff	$t_{\geq 1} \models \phi$

- where $t_{\geq i}$ is the suffix of t starting at t_i



Derived Operators

- *true*, *false*, \vee , \Rightarrow as usual
- Eventually: $\Diamond \phi \equiv \text{true} \cup \phi$
- Generally: $\Box \phi \equiv \neg \Diamond \neg \phi$
- Intuitive meaning



Specification Patterns

- Strong invariant
 - $\Box p$: p always holds
 - A file is always open or closed: $\Box(open \vee closed)$
 - Safety property
- Monotone invariant
 - $\Box(p \Rightarrow \Box p)$: once p , always p
 - Once information is public, it can never become secret again (but it may always stay secret): $\Box(public \Rightarrow \Box public)$
 - Safety property
- Establishing an invariant
 - $\Diamond \Box p$: eventually p will always hold
 - System initialization starts daemon process: $\Diamond \Box daemonRunning$
 - Liveness property

Specification Patterns (cont'd)

- Responsiveness

- $\Box(p \Rightarrow \Diamond q)$: everytime p holds, q will eventually hold
- All opened files must be closed eventually: $\Box(open \Rightarrow \Diamond closed)$
- Liveness property

- Fairness

- $\Box \Diamond p$: p holds infinitely often
- Producer does not wait infinitely long before entering the critical section:
 $\Box \Diamond crit$
- Liveness property

Needham-Schroeder Protocol

- If Alice and Bob have completed their protocol runs then Alice should believe her partner to be Bob if and only if Bob believes to talk to Alice

$$\Box(statusA = 1 \wedge statusB = 1 \Rightarrow (partnerA = agentB \Leftrightarrow partnerB = agentA))$$

- If Alice completed her protocol run with Bob, the intruder should not have learned Alice's nonce

$$\Box(statusA = 1 \wedge partnerA = agentB \Rightarrow knows_nonceA = 0)$$

- If Bob completed his protocol run with Alice, the intruder should not have learned Bob's nonce

$$\Box(statusB = 1 \wedge partnerB = agentA \Rightarrow knows_nonceB = 0)$$

5. Linear Temporal Logic

5.1 Linear-Time Properties

5.2 Linear Temporal Logic

5.3 LTL Model Checking

LTl Model Checking Problem

Given a finite transition system TS and an LTL formula ϕ ,
decide whether $t \models \phi$ for all $t \in \mathcal{T}(TS)$

- We need to check inclusion of traces
 - LTL formula ϕ describes set of traces $P(\phi)$
 - We need to determine whether $\mathcal{T}(TS) \subseteq P(\phi)$

A Simpler Problem: Regular Safety Properties

- A safety property is regular if its **bad prefixes** are described by a **regular language**
- Every invariant over AP is a regular safety property
 - Invariant $\Box p$
 - Bad prefixes start with $q^* r$ where $p \in q$ and $p \notin r$
- Regular safety property that is not an invariant
 - Traffic light: red is immediately preceded by yellow
 $\Box(\neg \text{yellow} \Rightarrow \bigcirc \neg \text{red})$
 - Bad prefixes start with $(\text{green} \mid \text{yellow yellow}^*(\text{red} \mid \text{green}))^* \text{red}$
- Non-regular safety property
 - Vending machine: at least as many coins inserted as drinks dispensed
 - Cannot be expressed in LTL with $AP = \{\text{pay}, \text{drink}\}$ (“LTL cannot count”)
 - Bad prefixes: regular languages cannot count

Checking Regular Safety Properties

- Safety properties are violated in finite time
 - Look at all finite prefixes $\mathcal{T}_{fin}(TS)$ of the traces $\mathcal{T}(TS)$ of a transition system TS
 - Check whether $\mathcal{T}_{fin}(TS)$ contains a bad prefix

Checking Regular Safety Properties

- Safety properties are violated in finite time
 - Look at all finite prefixes $\mathcal{T}_{fin}(TS)$ of the traces $\mathcal{T}(TS)$ of a transition system TS
 - Check whether $\mathcal{T}_{fin}(TS)$ contains a bad prefix
- Approach

Checking Regular Safety Properties

- Safety properties are violated in finite time
 - Look at all finite prefixes $\mathcal{T}_{fin}(TS)$ of the traces $\mathcal{T}(TS)$ of a transition system TS
 - Check whether $\mathcal{T}_{fin}(TS)$ contains a bad prefix
- Approach
 1. Describe finite prefixes $\mathcal{T}_{fin}(TS)$ by **finite automaton** \mathcal{FA}_{TS}

Checking Regular Safety Properties

- Safety properties are violated in finite time
 - Look at all finite prefixes $\mathcal{T}_{fin}(TS)$ of the traces $\mathcal{T}(TS)$ of a transition system TS
 - Check whether $\mathcal{T}_{fin}(TS)$ contains a bad prefix
- Approach
 1. Describe finite prefixes $\mathcal{T}_{fin}(TS)$ by **finite automaton** \mathcal{FA}_{TS}
 2. Describe bad prefixes of regular safety property P by **finite automaton** $\mathcal{FA}_{\bar{P}}$

Checking Regular Safety Properties

- Safety properties are violated in finite time
 - Look at all finite prefixes $\mathcal{T}_{fin}(TS)$ of the traces $\mathcal{T}(TS)$ of a transition system TS
 - Check whether $\mathcal{T}_{fin}(TS)$ contains a bad prefix
- Approach
 1. Describe finite prefixes $\mathcal{T}_{fin}(TS)$ by **finite automaton** \mathcal{FA}_{TS}
 2. Describe bad prefixes of regular safety property P by **finite automaton** $\mathcal{FA}_{\bar{P}}$
 3. Construct finite automaton for **intersection** of \mathcal{FA}_{TS} and $\mathcal{FA}_{\bar{P}}$

Checking Regular Safety Properties

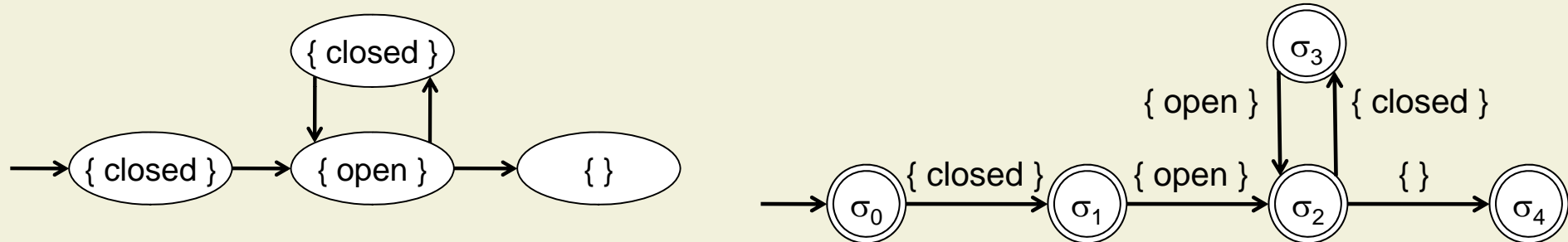
- Safety properties are violated in finite time
 - Look at all finite prefixes $\mathcal{T}_{fin}(TS)$ of the traces $\mathcal{T}(TS)$ of a transition system TS
 - Check whether $\mathcal{T}_{fin}(TS)$ contains a bad prefix
- Approach
 1. Describe finite prefixes $\mathcal{T}_{fin}(TS)$ by **finite automaton** \mathcal{FA}_{TS}
 2. Describe bad prefixes of regular safety property P by **finite automaton** $\mathcal{FA}_{\bar{P}}$
 3. Construct finite automaton for **intersection** of \mathcal{FA}_{TS} and $\mathcal{FA}_{\bar{P}}$
 4. Check whether intersection is **empty**
 - If intersection is non-empty, property P is violated
 - Each word in the intersection is a counterexample

Reminder: Finite Automata

- A finite automaton (FA) is a tuple $(Q, \Sigma, Q, \delta, q_0, F)$
 - Q : a finite set of states
 - Σ : a finite alphabet
 - δ : a transition relation, $\delta \subseteq Q \times \Sigma \times Q$
 - q_0 : an initial state
 - $F \subseteq Q$: a set of accepting states

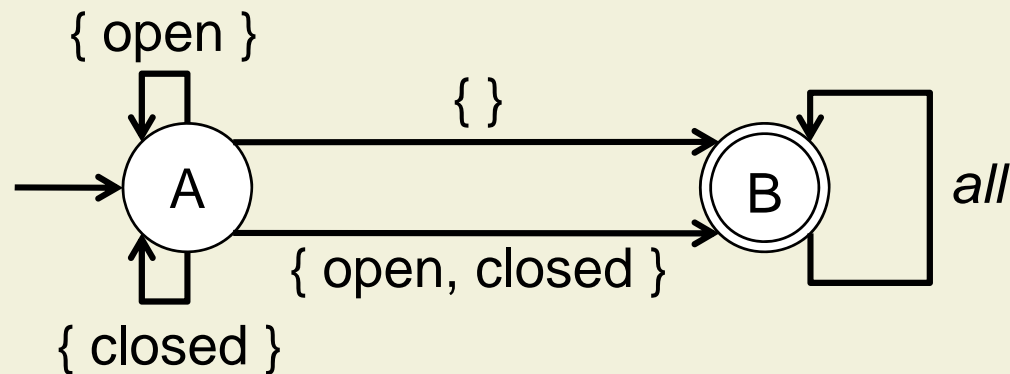
Step 1: Finite Automaton for Finite Prefixes

- Given a transition system $TS = (\Gamma, \sigma_I, \rightarrow)$, propositions AP , and labeling function L
- The automaton $\mathcal{FA}_{TS} = (Q, \Sigma, \delta, q_0, F)$ accepts $\mathcal{T}_{fin}(TS)$
 - $Q = \Gamma \cup \{\sigma_0\}$, where $\sigma_0 \notin \Gamma$
 - $\Sigma = \mathcal{P}(AP)$
 - $\delta = \{(\sigma, p, \sigma') \mid \sigma \rightarrow \sigma' \text{ and } p \in L(\sigma')\} \cup \{(\sigma_0, p, \sigma_I) \mid p \in L(\sigma_I)\}$
 - $q_0 = \sigma_0$
 - $F = Q$
- Example: `o:=o+1; while * do o:=o-1; o:=o+1 end; o:=o+1`



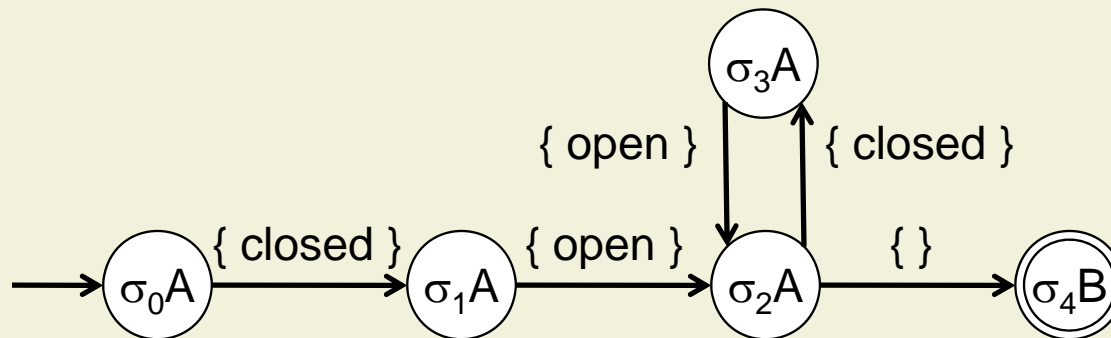
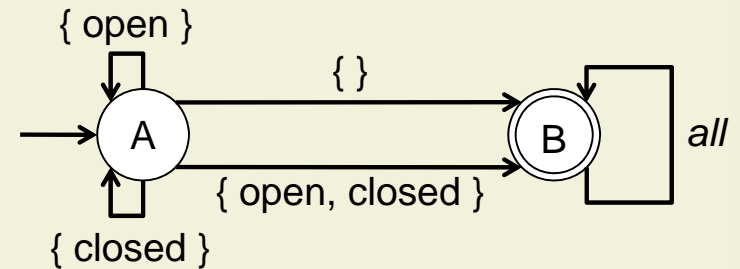
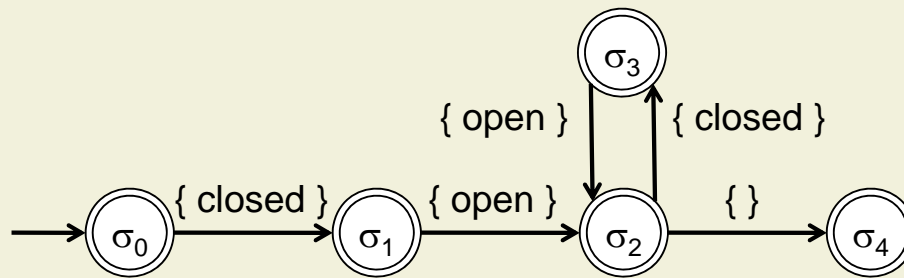
Step 2: Finite Automaton for Bad Prefixes

- By definition, bad prefixes are described by a regular language
- Apply standard construction to obtain FA $\mathcal{FA}_{\bar{p}}$ from regular expression
- Example: $\square(open \vee closed)$
 - Bad prefixes start with $(\{open\} \mid \{closed\})^* (\{\} \mid \{open, closed\})$



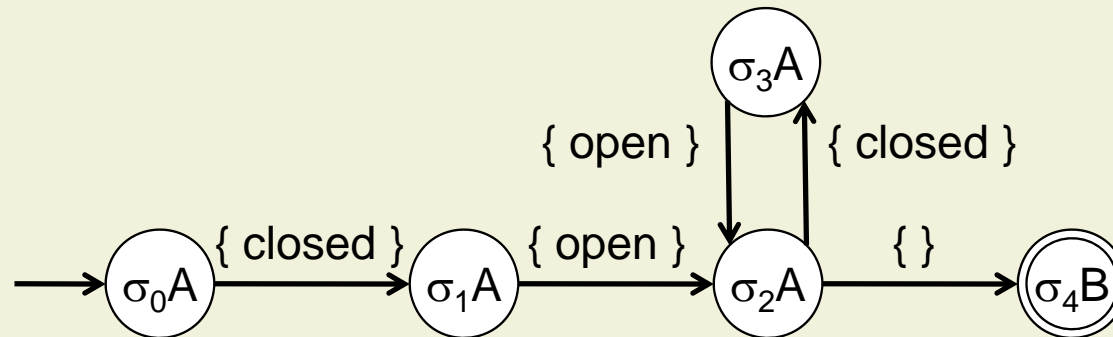
Step 3: Finite Automaton for Intersection

- Construct FA $\mathcal{FA}_{TS \cap \bar{P}}$ that accepts the intersection of the languages accepted by \mathcal{FA}_{TS} and $\mathcal{FA}_{\bar{P}}$
- Apply standard construction for product of two FA
- Example



Step 4: Check Emptiness

- If $\mathcal{FA}_{TS \cap \bar{P}}$ accepts a word w then
 - $w \in \mathcal{T}_{fin}(TS)$ because it is accepted by \mathcal{FA}_{TS} and
 - w is a bad prefix because it is accepted by $\mathcal{FA}_{\bar{P}}$
 - Therefore, P is not satisfied, and w is a counterexample
- Apply standard algorithm to check emptiness of FA
- Example



- Accepts $\{closed\}\{open\}(\{closed\} \mid \{open\})^* \{\}$
- Smallest counterexample: $\{closed\}\{open\}\{\}$
- Counterexample can be mapped back to transition system

Büchi Automata

- Büchi automata are similar to finite automata, but accept **infinite words**
- A Büchi automaton (BA) is a tuple $(Q, \Sigma, Q, \delta, q_0, F)$
 - Q : a finite set of states
 - Σ : a finite alphabet
 - δ : a transition relation, $\delta \subseteq Q \times \Sigma \times Q$
 - q_0 : an initial state
 - $F \subseteq Q$: a set of accepting states
- A run of a BA accepts its input if it **passes infinitely often through an accepting state**
- Büchi automata enjoy many of the properties of finite automata
 - We can construct the product of two BA
 - Emptiness is decidable

LTL Model Checking: Approach

1. Describe traces $\mathcal{T}(TS)$ by Büchi automaton \mathcal{BA}_{TS}
 - Construction is analogous to \mathcal{FA}_{TS}

LTL Model Checking: Approach

1. Describe traces $\mathcal{T}(TS)$ by Büchi automaton \mathcal{BA}_{TS}
 - Construction is analogous to \mathcal{FA}_{TS}
2. For an LTL formula ϕ , construct Büchi automaton $\mathcal{BA}_{\neg\phi}$ that accepts the traces characterized by $\neg\phi$ (bad traces)
 - We omit the details here

LTL Model Checking: Approach

1. Describe traces $\mathcal{T}(TS)$ by Büchi automaton \mathcal{BA}_{TS}
 - Construction is analogous to \mathcal{FA}_{TS}
2. For an LTL formula ϕ , construct Büchi automaton $\mathcal{BA}_{\neg\phi}$ that accepts the traces characterized by $\neg\phi$ (bad traces)
 - We omit the details here
3. Construct BA for intersection of \mathcal{BA}_{TS} and $\mathcal{BA}_{\neg\phi}$

LTL Model Checking: Approach

1. Describe traces $\mathcal{T}(TS)$ by Büchi automaton \mathcal{BA}_{TS}
 - Construction is analogous to \mathcal{FA}_{TS}
2. For an LTL formula ϕ , construct Büchi automaton $\mathcal{BA}_{\neg\phi}$ that accepts the traces characterized by $\neg\phi$ (bad traces)
 - We omit the details here
3. Construct BA for intersection of \mathcal{BA}_{TS} and $\mathcal{BA}_{\neg\phi}$
4. Check whether intersection is empty
 - If intersection is non-empty, property ϕ is violated
 - Each word in the intersection is a counterexample

Complexity Results

For a finite transition system TS and an LTL formula ϕ ,
the model checking problem $TS \models \phi$ is solvable in
 $\mathcal{O}(|TS| \times 2^{|\phi|})$

- $|TS|$ is the size of the transition system (which grows exponentially in the number of variables, processes, and channels)
- $|\phi|$ is the size of ϕ ; exponential complexity comes from the construction of $\mathcal{BA}_{\neg\phi}$

Advanced Model Checking Techniques

- On-the-fly model checking
 - Often violation of a property can be detected without checking all possible states or traces (for instance, $\Box p$)
 - Generate transition system and check property step-by-step
 - Implemented in Spin
- Partial order reduction
 - Remove redundancy from different interleavings of concurrent executions
 - Code segments that operate only on local state are not affected by interleaving
 - Implemented in Spin

Advanced Model Checking Techniques (cont'd)

- Bounded model checking
 - Check only prefixes of traces up to a certain length
 - Closer to testing than verification
 - Very effective in practice
- Symbolic model checking
 - Uses sets of states rather than individual states
 - Sets of states are represented through boolean functions
 - Very efficient data structure: binary decision diagram (BDDs)
 - Typically used to check branching-time properties
 - Can deal with larger models

Spec# Demo

Recommendations

- Course *System Development in Event-B* by J.-R. Abrial and T. S. Hoang-Do
 - Correctness-by-construction
 - Offered Fall 2009
- *Software Engineering Seminar* by P. Müller
 - Many topics of advanced formal methods
 - Offered Fall 2009
- Course *Concepts of Object-Oriented Programming* by P. Müller
 - **Informal** discussion of semantics of OO-languages; language design
 - Offered Fall 2009
- Planned course *Principles of Programming Languages* by P. Müller
 - Will cover semantics, type systems, static analysis, and verification of object-oriented programs
 - Planned for Fall 2010

Conclusions

- Variety of approaches
 - Best method depends on application area
- Tool support is essential
 - Proofs are tedious and error-prone
 - Some tools have reached maturity for industrial applications
- In this course, you have just seen the foundations
 - Advanced course planned to start next year
 - Interesting student projects available, for instance, on Spec#

Exam Info

- Exam will cover part II of the course (Formal Methods)
- Date
 - Monday, June 8th, 2009, 10:00 - 12:00
- Place
 - NO C 60 for last names starting with letter A - L
 - HG E 3 for last names starting with letter M - Z
- Closed-book
 - But rules of NS, SOS, and Hoare-logic will be provided if needed