SS 2017
Software Verification
Omega-regularity

Prof. Dr. Holger Schlingloff $^{1,2}$
Dr. Esteban Pavese $^1$

(1) Institut für Informatik der Humboldt Universität
(2) Fraunhofer Institut für offene Kommunikationssysteme FOKUS
Recap

- What is the „model checking problem“?
- What is the complexity of model checking for prop. logic?
- How can model checking be reduced to satisfiability checking?
- What does safety and liveness mean (informally)?
- Is $G^*p$ a safety or liveness property?
- What does recurrence and persistence mean?
- What does separation of safety and liveness mean?
Kripke Models define Safety Properties

- Each LTL formula defines the set of natural models in which it is valid
- Each Kripke model defines a set of maximal traces
- Can this be compared?

- Lemma: A finite Kripke model defines a safety property
  - thus, there are LTL-definable languages not definable by a Kripke model
- We have seen that finite Kripke models can be described in LTL with additional propositions (using only $G^*$)
  - Without additional variables it may not be possible to distinguish between states which have the same label
  - Of course, infinite Kripke models can define all possible languages
Regular and $\omega$-regular properties

- LTL with Until is expressively complete wrt. first-order logic
- However, there are interesting second-order properties
  - e.g. “p holds in every even state” ($\{s_0, s_2, s_4, \ldots\}$)
    - $\Box\Box p$
    - $p \land \Box(\neg p \rightarrow \Box p)$
    - $p \land \Box(\neg p \rightarrow X \neg p) \land \Box(\neg \neg p \rightarrow Xp)$
  - easy to model with a (nondeterministic) Kripke structure
  - easy to model with additional propositions
    - $q \land \Box(\neg q \rightarrow Xq) \land \Box(\neg q \rightarrow Xq) \land \Box(q \rightarrow p)$
- Solution: Incorporate grammar operators or second-order quantifiers into the logic (ETL or QPTL)
  - $\Box^{2n} p$
  - $\exists q \ (q \land \Box(q \rightarrow X \neg q) \land \Box(\neg q \rightarrow Xq) \land \Box(q \rightarrow p)$
Regular and ω-regular Languages

- Given signature $\mathbf{P} = \{p_1, \ldots, p_n\}$; each node in a Kripke model is labelled with a subset of $\mathbf{P}$
- Alphabet $\mathbf{A} = 2^\mathbf{P}$ allows to view models as automata / language generators

- $\text{Regexp}_\mathbf{A} ::= a \mid \lambda \mid (\text{Regexp}_\mathbf{A} ; \text{Regexp}_\mathbf{A}) \mid (\text{Regexp}_\mathbf{A} + \text{Regexp}_\mathbf{A}) \mid \text{Regexp}_\mathbf{A}^*$

- $\omega\text{-Regexp}_\mathbf{A} ::= a \mid \lambda \mid (\omega\text{-Regexp}_\mathbf{A} ; \omega\text{-Regexp}_\mathbf{A}) \mid (\omega\text{-Regexp}_\mathbf{A} + \omega\text{-Regexp}_\mathbf{A}) \mid \omega\text{-Regexp}_\mathbf{A}^* \mid \omega\text{-Regexp}_\mathbf{A}^\omega$
Büchi Automata

An $\omega$-automaton or fair transition system over the alphabet $\Sigma = 2^P$ is defined like a usual (nondeterministic) automaton with an additional recurrence set (“fairness constraint”); it is a tuple $(S, \Delta, S_0, S_{acc}, S_{rec})$, where

- $S$ is a set of states,
- $\Delta \subseteq S \times \Sigma \times S$ is the transition relation,
- $S_0 \subseteq S$ is the set of initial states,
- $S_{acc} \subseteq S$ is the set of accepting states (for finite words), and
- $S_{rec} \subseteq S$ is the set of recurring states (for infinite words).

- each finite run must end in an accepting state
- in each infinite run, at least one recurring state must be selected infinitely often
Figure 3: A Büchi automaton accepting \((\neg p1)^\omega + (T^+; p2)^\omega\)
Operations on Büchi-automata

- Union
- Intersection
- Complement?

- Büchi-automata can not necessarily be determinized
  - Example: “finitely many a”
Muller Automata

- Characterize exactly which set of states is recurring: In a Muller $S_{\text{rec}}$ is a set of sets of accepting states
  - $S_{\text{rec}} \subseteq 2^S$
  - run $\sigma = (s_0, s_1, s_2, ...)$ is accepting if $\{s | s \text{ occurs infinitely often in } \sigma\} \in S_{\text{rec}}$
- Easy complementation procedure
- Easy determinisation procedure
- Exponential blowup