Specification Languages for Stutter-Invariant Regular Properties

Christian Dax

ETH Zurich

Joint work with

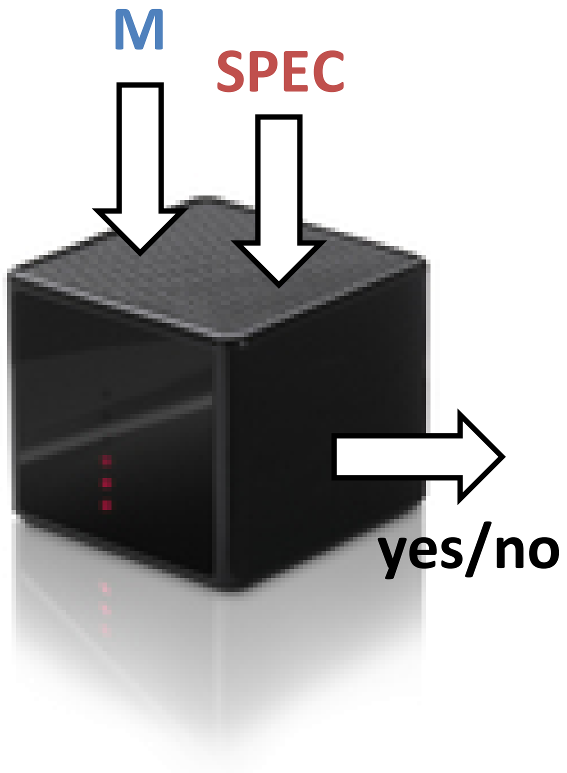
Felix Klaedtke and Stefan Leue

ATVA 2009

Motivation: Finite-State Model Checking

- Given:
  1. Transition system M
  2. Specification SPEC

- Question: M satisfies SPEC?

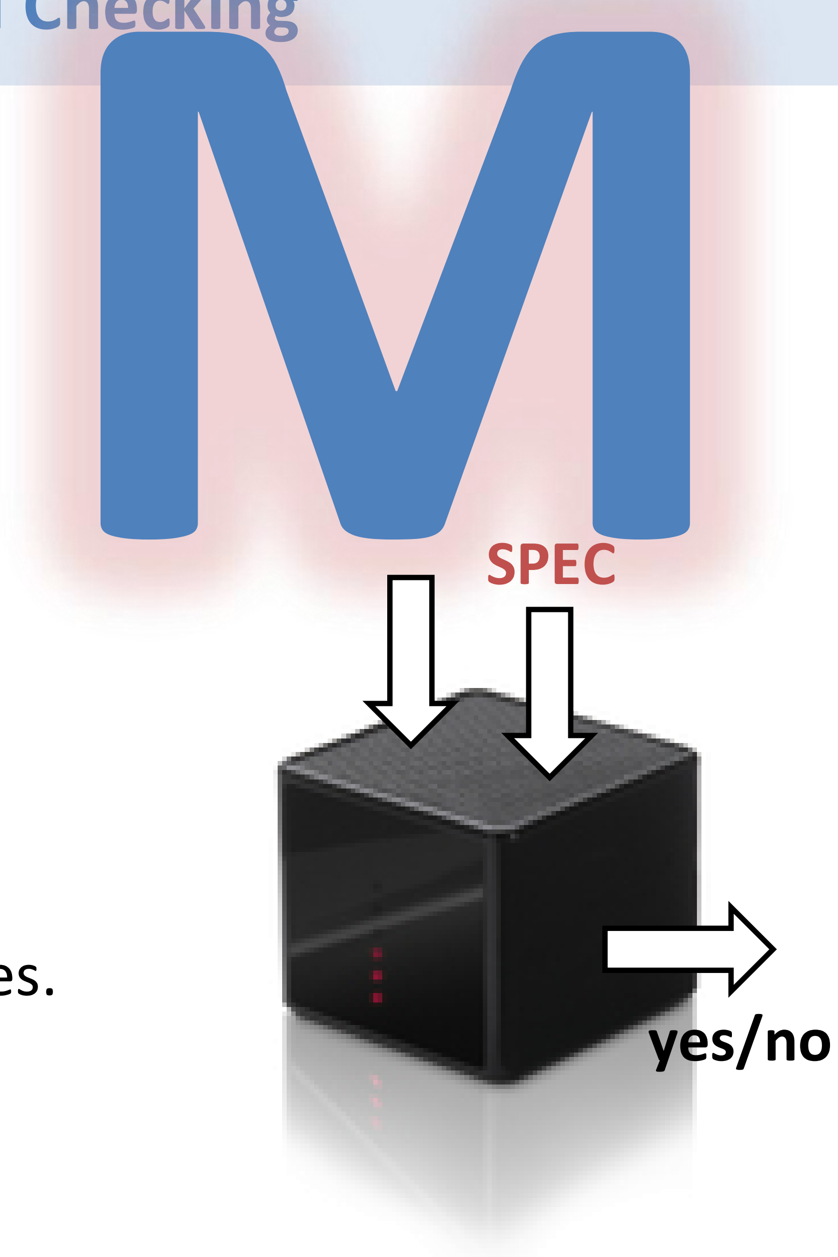


Motivation: Finite-State Model Checking

- Given:
  1. Transition system M
  2. Specification SPEC

- Question: M satisfies SPEC?

- **State-space explosion** of M when modeling e.g. concurrent processes.



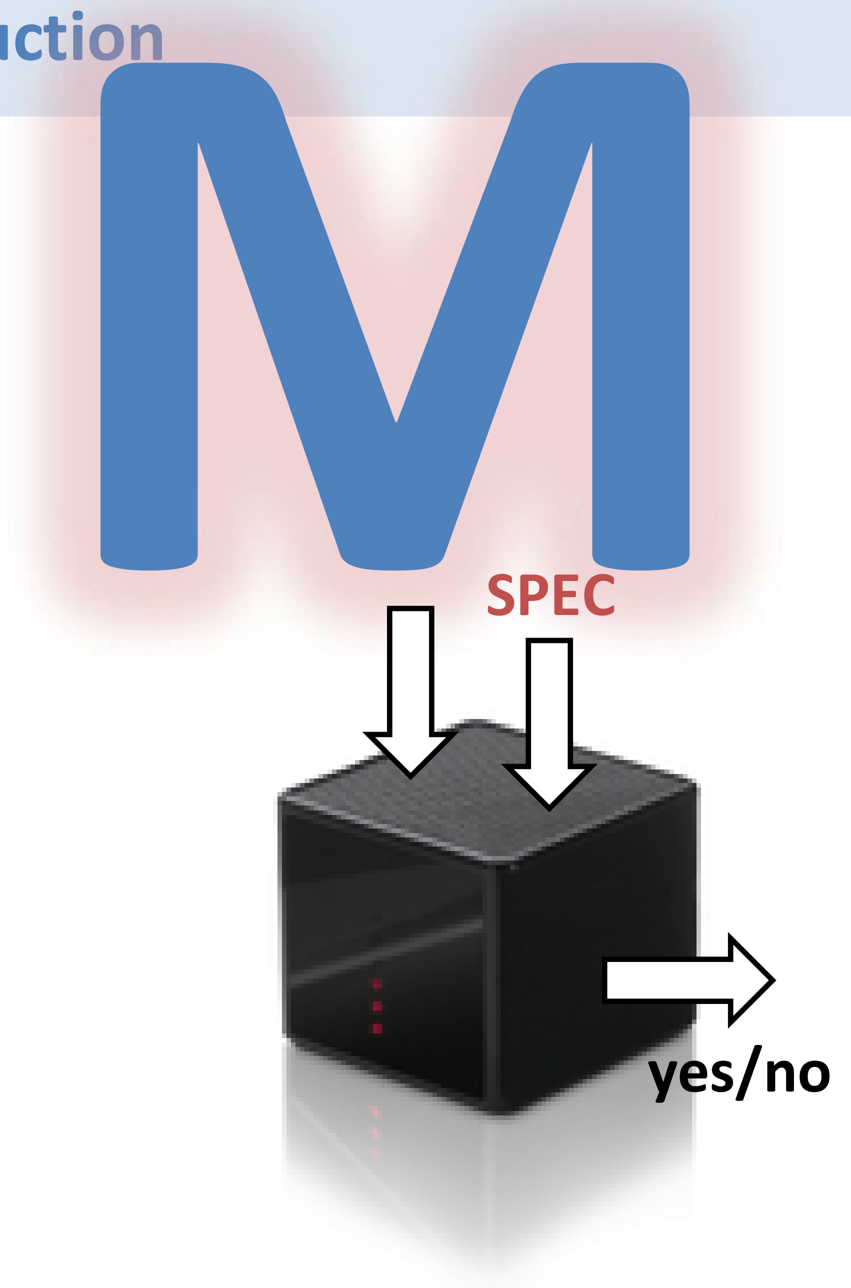
Motivation: Partial-Order Reduction

- One technique to cope with huge Ms is **partial-order reduction:**

  Model checker **explores only pruned version** of M.

- Is the answer still correct?

- Yes, if SPEC is **stutter-invariant**.



Definition of Stutter Invariant SPECs

- A SPEC is **stutter invariant** if

  for each trace

  aaaaaaacccbbbbbbbbaaaaaa in SPEC

  SPEC also contains all traces described by

  a+ c+ b+ a+

Stutter-Invariance Check is Hard

- We want to use partial-order reduction…

  BUT

  is SPEC stutter-invariant?

- Bad news: check is **PSPACE-complete** even for LTL

- Important question: **How to avoid such an expensive check?**

Related Work

How to avoid such an expensive check?

Related Work 1: [Holzmann/Kupferman ’96]

1. Translation of SPEC to automaton.

2. Make automaton stutter invariant by **adding missing traces**.

   for each trace

   aaaaaaacccbbbbbbbbaaaaaa in SPEC

   add all traces described by

   a+ c+ b+ a+

Problems: [Holzmann/Kupferman ’96]

1. Adding traces blows up the SPEC and decelerates model checking.

2. Incompatible with symbolic model checking.

3. If SPEC is not stutter-invariant, we might get counter-examples that are false positives.

Related Work 2: [Peled/Wilke '97]

- **Syntactic characterization** of stutter-invariant SPECs.

- LTL without the next operator is stutter-invariant and all stutter-invariant LTL properties are expressible.

  $$\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \, U \, \varphi,$$

  where p is a proposition.

- **Expressive power too weak:**
  not all stutter-inv. regular properties expressible.

Related Work 3: [Rabinovich ’98], [Etessami ‘99]

- **Syntactic characterization** of stutter-invariant **regular** SPECs using quantifiers over propositions.

- Problems:
  1. Semantically restricted quantification makes expressing properties difficult.
  2. Rabinovich: syntactically closed under negation **but** complexity of model checking is nonelementary.

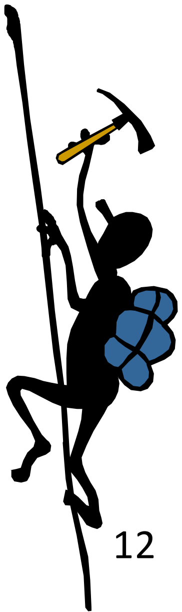
     Etessami: complexity of model checking is in PSPACE **but** syntactically not closed under negation.

What do we want?

1. Syntactic characterization with **simple** syntax and semantics.
   $\Rightarrow$ Practical for software engineers

2. **Feasible** for (symbolic) model checking.
   $\Rightarrow$ Model checking should be in PSPACE.

3. Expressing **all stutter-invariant regular** properties.



Our Suggestion

Our Specification Languages

1. **siRE** for finite-trace languages.
   (variation of regular expressions)

2. **siPSL** for infinite-trace languages.
   (variation of PSL, which combines regular expressions + LTL)

Defining siRE: First Attempt

- Restrict regular expressions by **adding plus to letters**

  $$r ::= \epsilon \mid b+ \mid r ; r \mid r \cup r \mid r^*$$

  where b is a Boolean combination of propositions.

- Example: {p, q} {p} {q} {q} {p, q} in language of **(p ∨ q)+**

- Problem: L(**p+; q+**) over {p, q} not stutter-invariant!
- L(**p+; q+**) contains "**aa**" but not "**a**", for **a** := {p, q}

Glimpse at siRE

- Syntax:

$$r ::= \epsilon \mid b+ \mid \mathbf{b^*;r} \mid r \cup r \mid \mathbf{r:r} \mid \mathbf{r\oplus}$$

where b is a Boolean combination of propositions.

- **Fusion :=** concatenation with one **overlapping letter**

L(**r1** : **r2**) := { w**b**v | w**b** in L(**r1**) and **b**v in L(**r2**) }

- Example: {p} {p, q} {p} {**p**} {p} {q} {q} {p, q} {q}

in L(**r1**) in L(**r2**)

Glimpse at siRE

- Syntax:

  $$r ::= \epsilon \mid b+ \mid \mathbf{b^*; r} \mid r \cup r \mid \mathbf{r : r} \mid \mathbf{r\oplus}$$

  where b is a Boolean combination of propositions.

- L(**r**⊕) := L(r+) but **with fusion** instead of concatenation.

- Example: {p} {p, q} {**p**} {p} {p} {q} {q} {**p, q**} {q} {p, q} {p}

  in L(**r**) in L(**r**) in L(**r**)

Glimpse at siRE: Examples

- **p at the first position and q at the last position**

  language of **p+ : true+ : q+** contains

  {p} {p, q} {p} {**p**} {p} {q} {q} {p, q} {**q**} {q}

- **Alternately {p} and {q}**

  language of **((p ∧ ¬q) ∨ (q ∧ ¬p))+** contains

  {p} {p} {q} {q} {p} {p} {p} {q} {q} {p}

Glimpse at siPSL

- Syntax:

  $$\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \, U \, \varphi \mid <r> \varphi$$

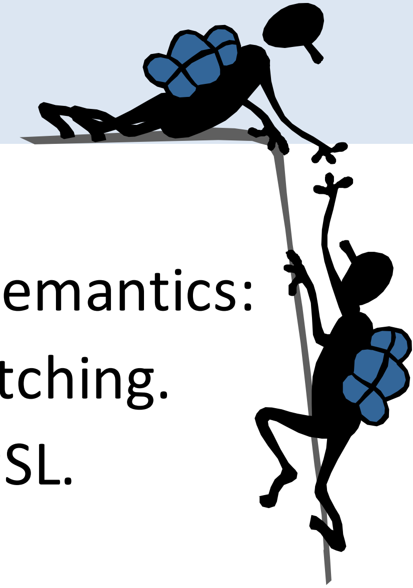
  where p is a proposition and r is a siRE.

- $L(<r> \varphi) := \{ w\mathbf{b}v \mid w\mathbf{b} \text{ in } L(r) \text{ and } \mathbf{b}v \text{ in } L(\varphi) \}$
  with **overlapping letter** b

- Example: : {p} {p, q} {p} {**p**} {p} {q} {q} {p, q} {q} …

  in L(r)   in L(φ)

What We Get for Free

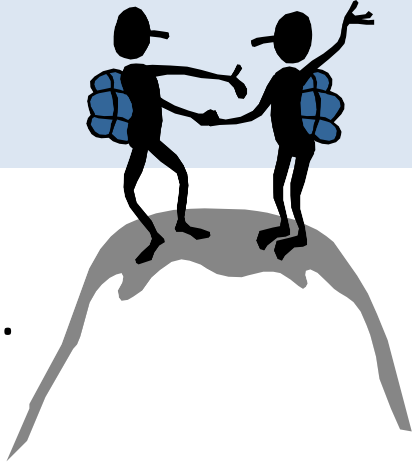


1. Syntactic characterization has **simple** syntax and semantics:
    1. Regular expressions widely used in pattern matching.
    2. siPSL similar to industry-driven IEEE standard PSL.
    3. siPSL syntactically closed under negation.

2. **Feasible** for (symbolic) model checking.
   For siPSL, **in PSPACE** (reuse algorithms)

3. What about expressiveness?

Expressiveness of siRE/siPSL



3. Expressing **all stutter-invariant regular** properties.

Theorem: every siRE/siPSL formula is stutter-invariant.

Theorem: every stutter-invariant regular property is expressible.

- Translation from regular expressions to siREs
- Intuition: translation adds missing traces
- Translation preserves property if stutter-inv.
- Exponential blow-up

Conclusion

- Contributions:
  1. We present **easy-to-use** specification languages
  2. We prove that they express **stutter-invariant** properties.
  3. We prove that they cover all stutter-invariant **regular** properties.

- Future work:
  - Optimized implementation to use siPSL with the SPIN model checker.
  - Open question: polynomial translation from RE to siRE?