Property-Preserving Updates

Ivan Lanese
University of Bologna/INRIA

Joint work with Davide Bresolin
University of Bologna

Map of the talk

- Research questions
- Our setting
- Solutions and complexity
- Conclusion

Map of the talk

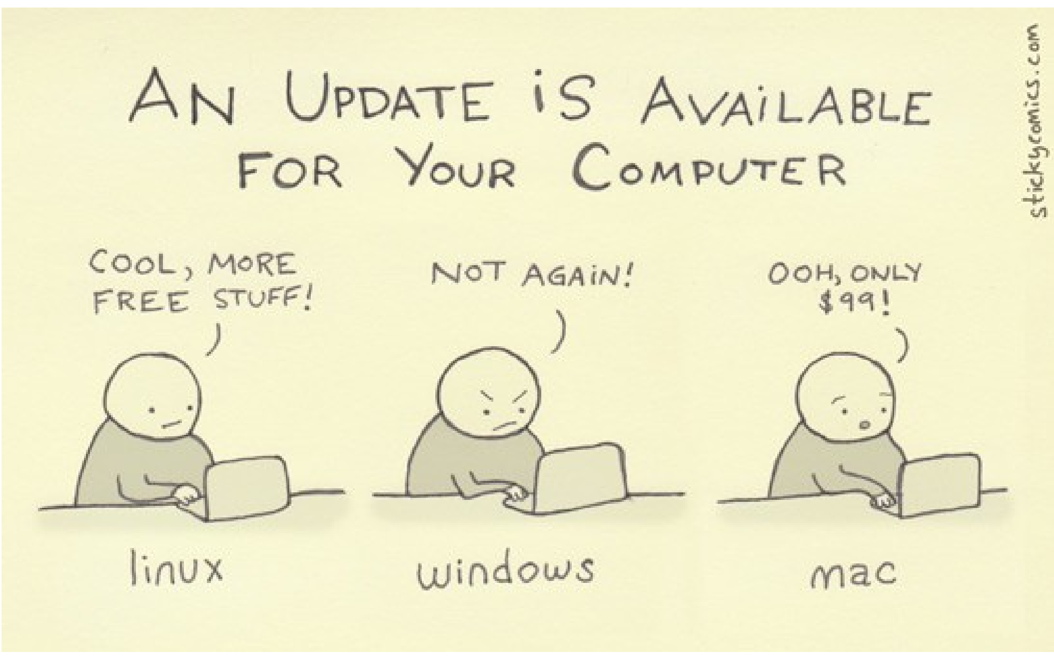
- Research questions
- Our setting
- Solutions and complexity
- Conclusion

Why updates?

- Applications need to be updated, statically or dynamically
- Many possible uses
  - Deal with changing business rules or environment conditions
  - Bug fixes
  - Specialize the application to user preferences

Which updates?

- We consider a simple but general update mechanism
- The application is composed by a context C, and a component to be updated A
- The component A is replaced by B
- We go from C[A] to C[B]



Property preservation

- Many approaches to check application correctness
  - Model checking, testing, abstract interpretation
- If the application is updated, we do not want to redo the whole checking from scratch
- This entails the following research question

If C[A] satisfies a given property φ,
what should one require on B
to ensure that also C[B] satisfies φ

Some natural generalizations

- One may require that φ is preserved while replacing A with B in any context C
- One may require that replacing A with B in context C preserves all the properties of C[A]
- One may require that replacing A with B in any context C preserves all the properties of C[A]

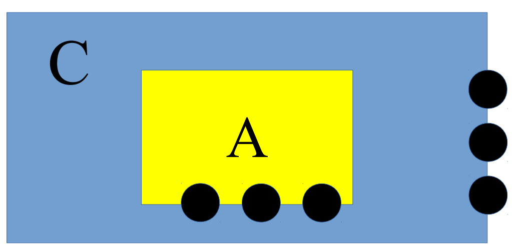
- The answer to these questions depend
  - on the models for context and components
  - on the synchronization mechanisms
  - on the logic for expressing properties

Map of the talk

- Research questions
- Our setting
- Solutions and complexity
- Conclusion

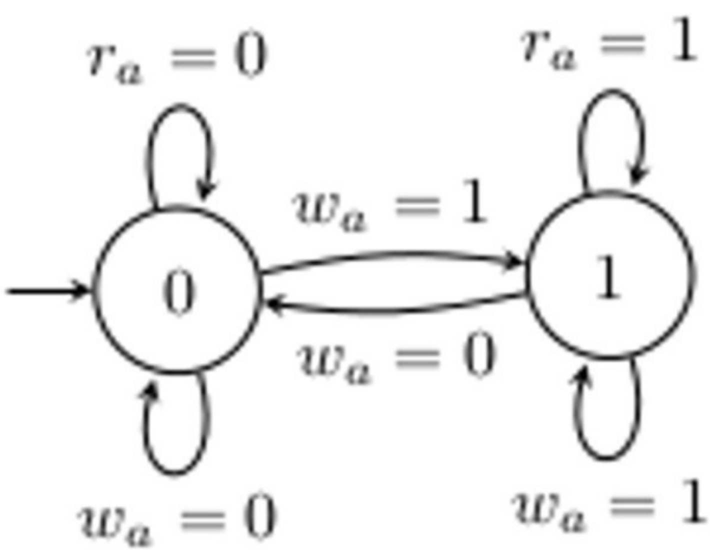
Components and contexts

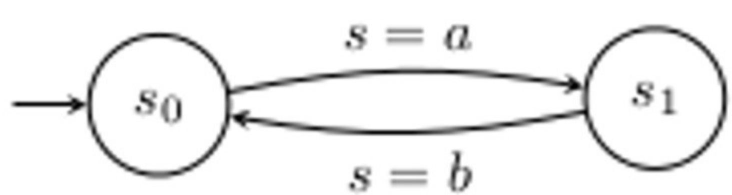
- We model contexts and components as constraint automata
  - $\mathcal{A} = < Q, N, q_0, \rightarrow >$
  - Automata with internal and external interface
  - Each interface is a subset of the set of nodes N
  - At each step, values are communicated on nodes
  - Labels are functions from N to data $\cup$ $\{\perp\}$
- We consider embeddings
  - The component communicates only with the context
- We consider both synchronous and asynchronous synchronization



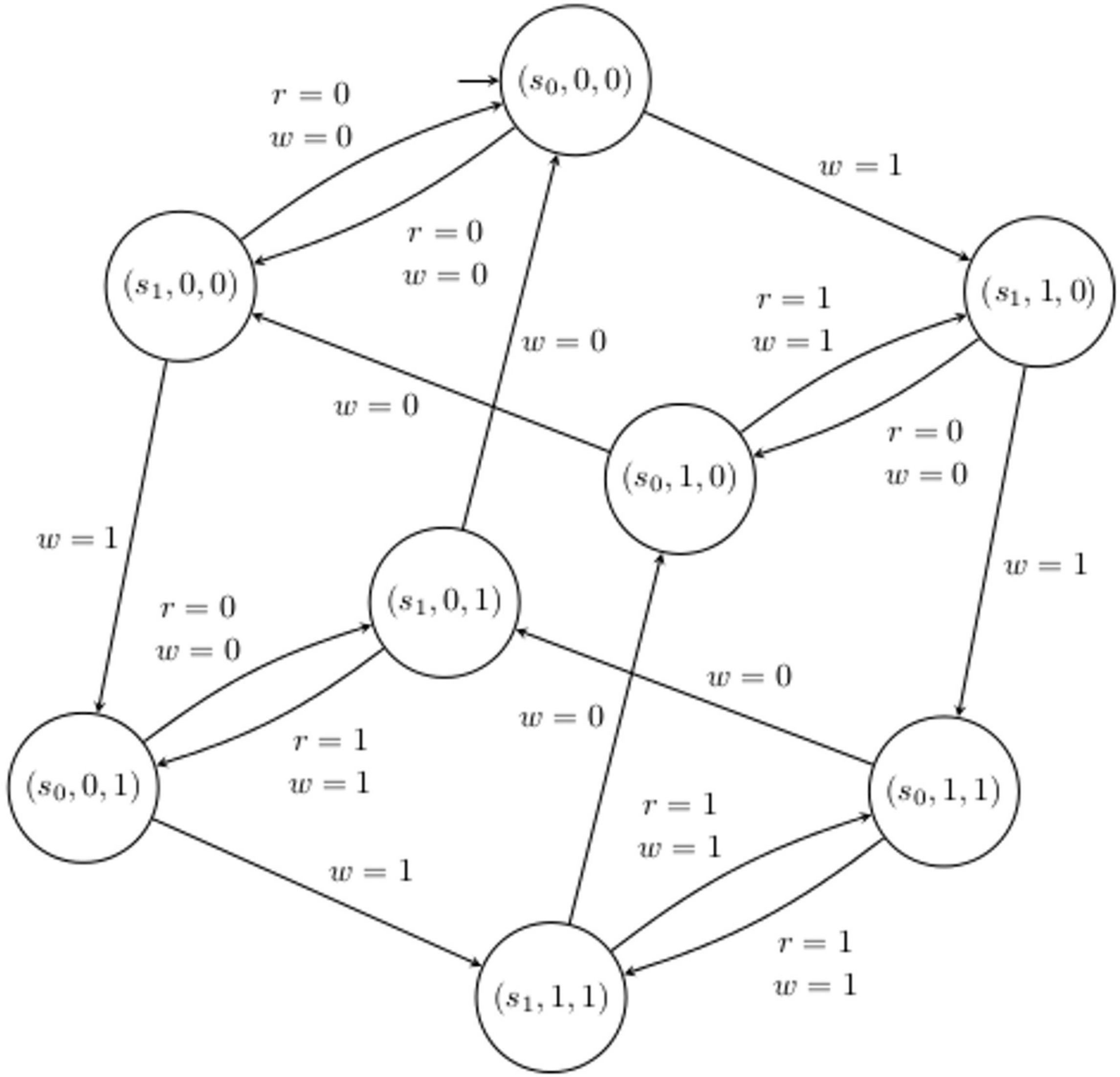
Running example

- We consider a system composed by two 1 bit registers, A and B
- Registers can be read and written
- A scheduler decides at each step which register can be accessed from outside





The whole system



Formulas

- We consider formulas in the safety fragment of μ-calculus
- $\varphi ::= tt \mid ff \mid X \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid [\alpha]\varphi \mid \nu X.\varphi$
- We have I/O constraints inside the modality
  - I/O constraints describe which data pass on nodes
  - $\alpha ::= tt \mid n = d \mid n = \perp \mid n \neq d \mid n \neq \perp \mid \alpha,\alpha$
- We interpret it on both finite and infinite runs of the automata
- $\varphi = [w = 1]\, ff \vee [tt][tt][r = 0]\, ff$
- Either I don't write 1 at first step, or I don't read 0 at third step

Map of the talk

- Research questions
- Our setting
- Solutions and complexity
- Conclusion

One property, one context (1)

- We want to find a most general B such that if C[A] satisfies a given property φ, then also C[B] satisfies φ
- This amounts to solve the following language equation:

$$L(C[B]) \subseteq L(\Phi)$$

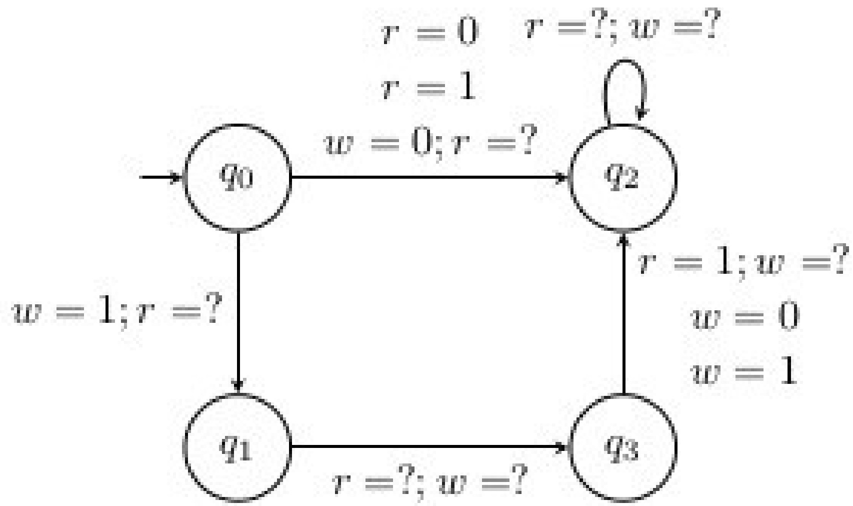
  where Φ is an automaton equivalent to φ
- This has been solved in the literature, and the solution is $B = C[\overline{\Phi}]$

One property, one context (2)

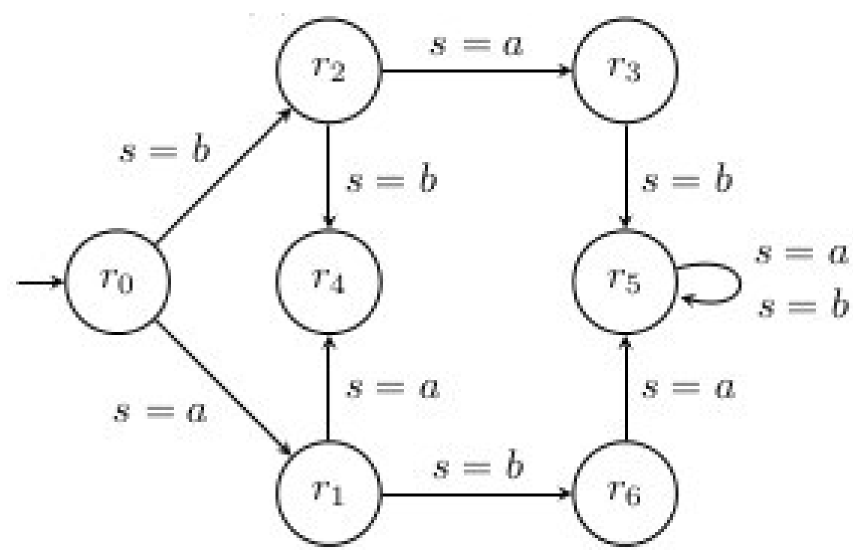
- $B=C[\overline{\Phi}]$ can be computed by adding final states to the automata
- Since we are interested in prefix-closed solutions, one can remove final states from the solution
- The problem is in 2-EXPTIME, since it requires a double complementation
- The problem is EXPSPACE-hard
  - Proved by reducing a suitable three-player game to it
  - The component and the formula play against the context
- The same approach can be used to ensure that a given property that does not hold in C[A] holds in C[B]

One property, one context: running example

- We consider φ = [w = 1] ff ∨ [tt][tt][r = 0] ff



- The resulting scheduler is:

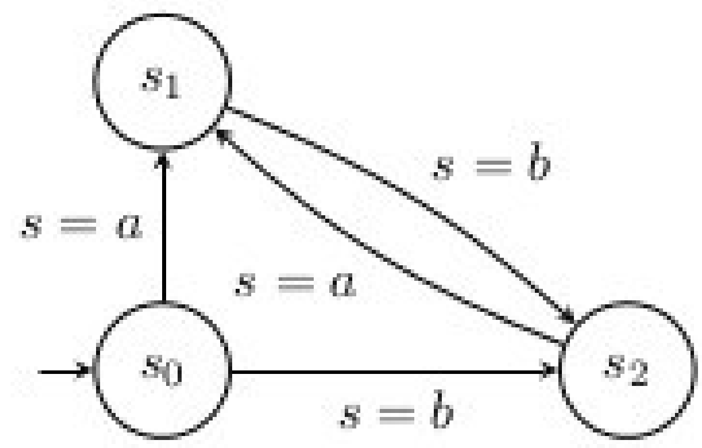


All properties, one context

- We want to find a most general B such that C[B] satisfies all the properties satisfied by C[A]
- This amounts to solve the following language equation:

  $$L(C[B]) \subseteq L(C[A])$$

- This has been solved in the literature, and the solution is $B=C[\overline{C[A]}]$
- The problem is in 2-EXPTIME, since it requires a double complementation



One property, all contexts

- We want to find a most general B such that for each context C if C[A] satisfies a given property φ, then also C[B] satisfies φ
- For asynchronous embedding, unless the formula is true or false, we need:

$$L(B) \subseteq L(A)$$

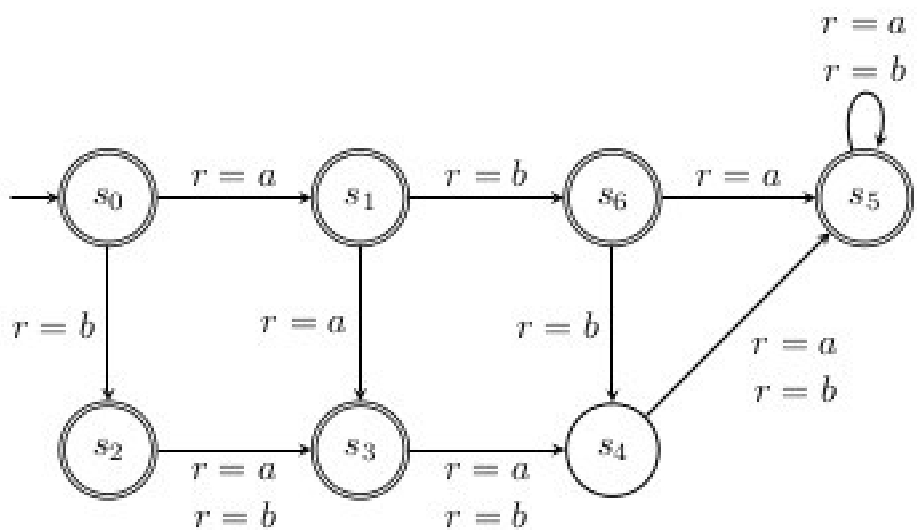
- For synchronous embedding we need:

$$L(B) \subseteq L(A) \cup \overline{R(\varphi)}$$

where $R(\varphi)$ contains all the words of lengths n such that there exists zc of length n such that z satisfies φ and zc does not satisfy φ

One property, all contexts: running example

- We consider φ = [w = 1] ff ∨ [tt][tt][r = 0] ff
- The resulting scheduler is:



- We can remove the non-final state to restrict to prefix-closed solutions

All properties, all contextx

- We want to find the most general B such that, for each context C, C[B] satisfies all the properties satisfied by C[A]
- This amounts to solve the following language equation:

$$L(B) \subseteq L(A)$$

Map of the talk

- Research questions
- Our setting
- Solutions and complexity
- Conclusion

Summary

- We studied under which conditions updates preserve a given property
- We generalized to all properties and/or all contexts

Future work

- Consider the same problem in different settings
  - Other kinds of automata
  - Other kinds of properties
- What happens when multiple updates are considered?

Finally

Thanks!

Questions?