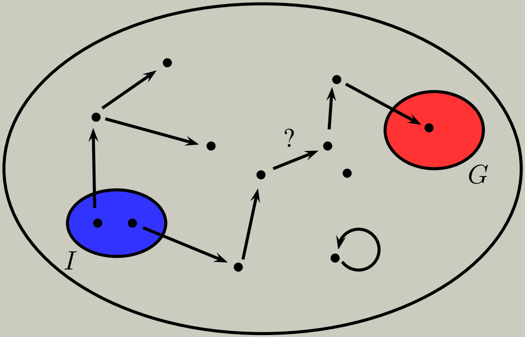
Semantic guidance for unbounded symbolic reachability

Martin Suda

Max Planck Institute für Informatik

VTSA 2012

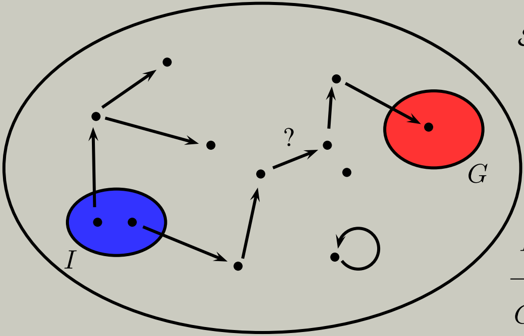
Transition system



Reachability

Does there exist a finite path from an I-state to a G-state?

Symbolically represented transition system



$$\mathcal{S} = (\Sigma, \varphi, \tau, \psi)$$

Σ ... prop. signature
φ ... fla over Σ
τ ... fla over $\Sigma \cup \Sigma'$
ψ ... fla over Σ

$$I = \{s \mid s \models \varphi\}$$
$$\rightarrow = \{(s, s') \mid (s, s') \models \tau\}$$
$$G = \{s \mid s \models \psi\}$$

Reachability

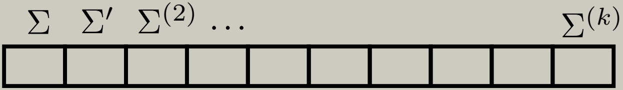
Does there exist a finite path from an I-state to a G-state?

Fixed length reachability via SAT

- Does there exist a path from an I-state to a G-state of length k?
- We can use a SAT-solver to answer such question:

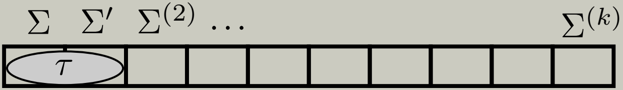
Fixed length reachability via SAT

- Does there exist a path from an I-state to a G-state of length k?
- We can use a SAT-solver to answer such question:



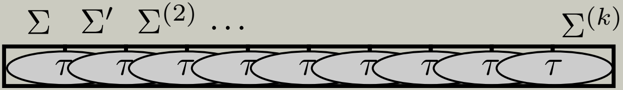
Fixed length reachability via SAT

- Does there exist a path from an *I*-state to a *G*-state of length *k*?
- We can use a SAT-solver to answer such question:



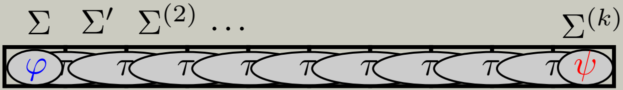
Fixed length reachability via SAT

- Does there exist a path from an I-state to a G-state of length k?
- We can use a SAT-solver to answer such question:



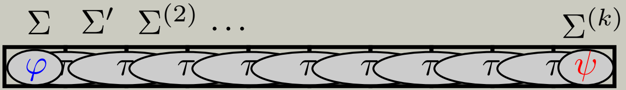
Fixed length reachability via SAT

- Does there exist a path from an *I*-state to a *G*-state of length *k*?
- We can use a SAT-solver to answer such question:



Fixed length reachability via SAT

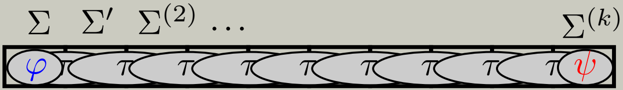
- Does there exist a path from an I-state to a G-state of length k?
- We can use a SAT-solver to answer such question:



- Now just run the solver: A push button technology!

Fixed length reachability via SAT

- Does there exist a path from an I-state to a G-state of length k?
- We can use a SAT-solver to answer such question:



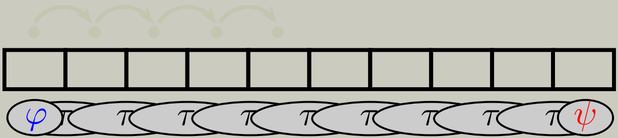
- Now just run the solver: A push button technology!

Bounded model checking

- Iterate the above for increasing values of $k = 0, 1, 2, \ldots$
- If one of them is SAT, we have an answer!
- But how do we know when to terminate in the other case?

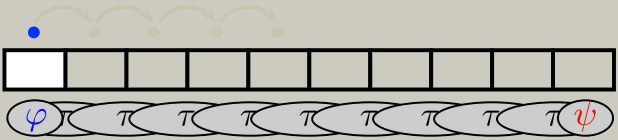
Opening the blackbox

- We need more control over what's happening inside the solver
- Let's control the way the model is constructed:



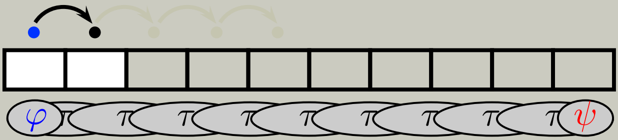
Opening the blackbox

- We need more control over what's happening inside the solver
- Let's control the way the model is constructed:



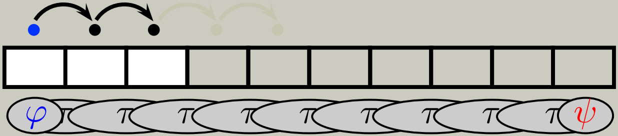
Opening the blackbox

- We need more control over what's happening inside the solver
- Let's control the way the model is constructed:



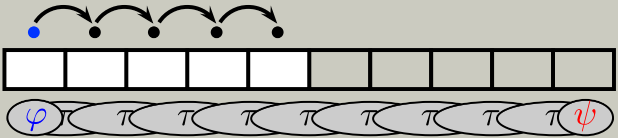
Opening the blackbox

- We need more control over what's happening inside the solver
- Let's control the way the model is constructed:



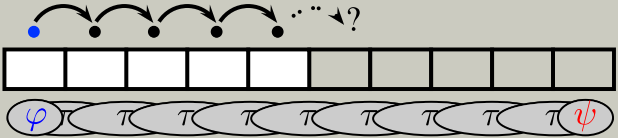
Opening the blackbox

- We need more control over what's happening inside the solver
- Let's control the way the model is constructed:



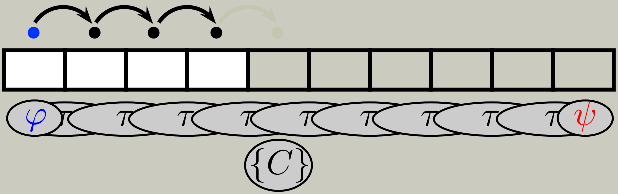
Opening the blackbox

- We need more control over what's happening inside the solver
- Let's control the way the model is constructed:



Opening the blackbox

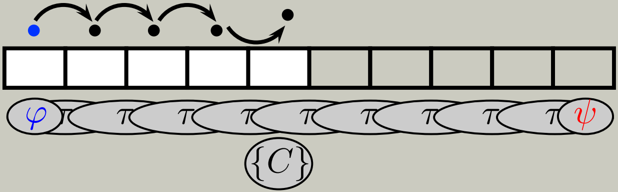
- We need more control over what's happening inside the solver
- Let's control the way the model is constructed:



- If the model cannot be extended, a *conflict clause* is derived,

Opening the blackbox

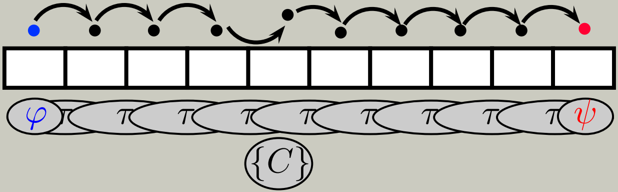
- We need more control over what's happening inside the solver
- Let's control the way the model is constructed:



- If the model cannot be extended, a *conflict clause* is derived,
- which forces the search to take a different path.

Opening the blackbox

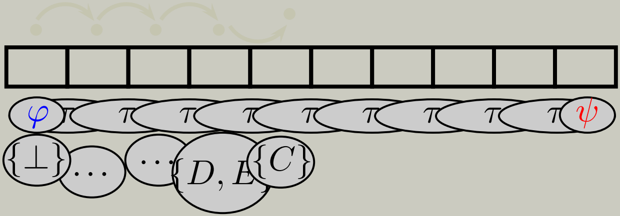
- We need more control over what's happening inside the solver
- Let's control the way the model is constructed:



- If the model cannot be extended, a *conflict clause* is derived,
- which forces the search to take a different path.
- As with BMC we either finish with the full model,

Opening the blackbox

- We need more control over what's happening inside the solver
- Let's control the way the model is constructed:



- If the model cannot be extended, a *conflict clause* is derived,
- which forces the search to take a different path.
- As with BMC we either finish with the full model,
- or discover inconsistency in a form of the empty clause $\perp$.

Dependency

We say that a conflict clause C *depends* on another clause D if D was used as an assumption in the proof of C.

Dependency

We say that a conflict clause C *depends* on another clause D if D was used as an assumption in the proof of C.

Dependency in action

Typically, the empty clause depends both on φ and ψ in our runs, otherwise we can directly terminate with UNSAT:

Dependency

We say that a conflict clause C *depends* on another clause D if D was used as an assumption in the proof of C.

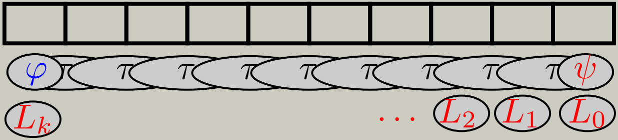
Dependency in action

Typically, the empty clause depends both on φ and ψ in our runs, otherwise we can directly terminate with UNSAT:

- Empty clause depending only on φ:
  there is no path of length k starting in a φ-state.
- Empty clause depending only on ψ:
  there is no path of length k ending in a ψ-state.
- Empty clause depending on neither:
  there is no path of lenght k.

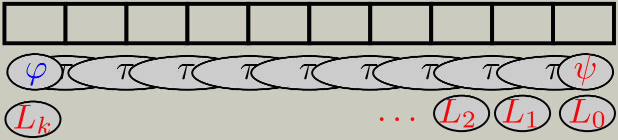
Defining layers

Let L_i be the set of clauses that *depend* on ψ and were inserted j steps before the goal formula ψ.



Defining layers

Let L_i be the set of clauses that *depend* on ψ and were inserted j steps before the goal formula ψ.



Properties of layers

- $(L_i)' \wedge \tau \models L_{i+1}$ (The way they get derived.)
- $L_i \wedge \varphi \models \bot$ (That's how it ended when $k = i$.)
- Once $L_i = L_j$ for $i \neq j$, the whole instance is UNSAT. (Cut and paste argmument over the proof.)

Summary of the method

- SAT-solver builds a model path for left to right
- Failure to proceed is recorded as a clause at that position
- Repeating pattern of such clauses entails overall UNSAT

Summary of the method

- SAT-solver builds a model path for left to right
- Failure to proceed is recorded as a clause at that position
- Repeating pattern of such clauses entails overall UNSAT

Related work

- BMC [Biere, Cimatti, Clarke, Zhu 1999]
- *k*-induction [Sheeran, Singh, Stålmarck 2000]
- Interpolation [McMillan 2003]
- IC3/PDR [Bradley 2011]

Summary of the method

- SAT-solver builds a model path for left to right
- Failure to proceed is recorded as a clause at that position
- Repeating pattern of such clauses entails overall UNSAT

Related work

- BMC [Biere, Cimatti, Clarke, Zhu 1999]
- *k*-induction [Sheeran, Singh, Stålmarck 2000]
- Interpolation [McMillan 2003]
- IC3/PDR [Bradley 2011]

Thank you for attention

Comments? Questions? Suggestions?