Exploiting Partial Knowledge for Efficient Model Analysis

Nuno Macedo Alcino Cunha Eduardo Pessoa
Universidade do Minho & INESC TEC, Portugal

ATVA 2017, Pune, India

Model Finding

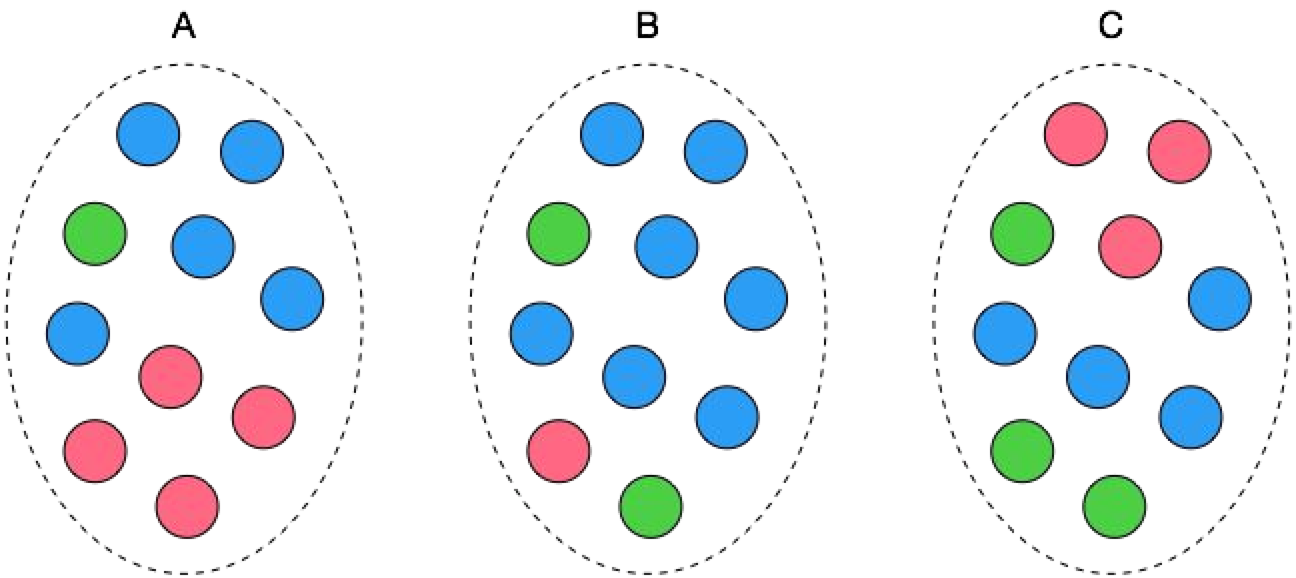
- *Model finders* automatically generate models within a bounded search space that satisfy a certain constraint
- Increasingly useful for model verification and validation on early software design stages
- Can be used directly by the end user
- But are typically at the backend of other frameworks with higher-level specification languages
- Effective due to the advancement of the underlying *solvers*

Kodkod

- Kodkod is *relational* model finder (Torlak & Jackson, 2007)
- In Kodkod a model finding problem is represented by:
  - A universe of atoms
  - A set of relations restricted by bounds
    - Atom tuples that **must** and **may** belong
  - A relational formula that must hold
- A solution is a binding for each relational variable
- Problems are solved by off-the-shelf SAT solvers
- Efficient iteration of instances/counter-examples through incremental SAT solving

Model Finding Scenario

All relations and constraints are solved in an “amalgamated” manner

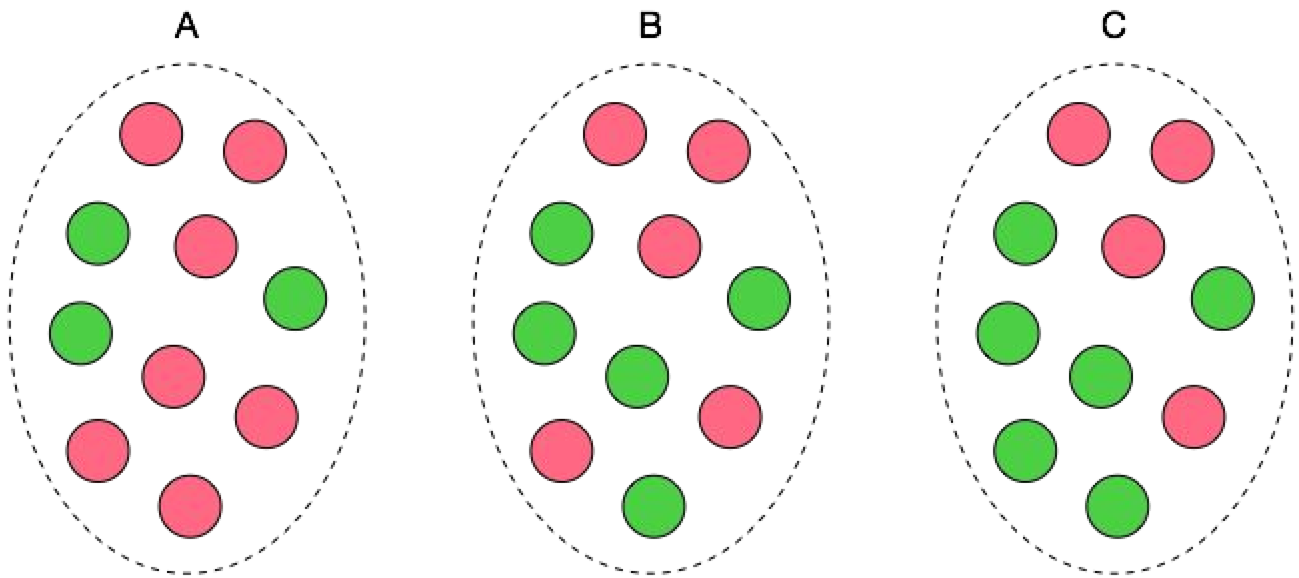


$$\phi = A \subseteq B \subseteq C \wedge \psi$$

Lower bounds define *partial instances*, upper bounds (usually) typing restrictions

Model Finding Scenario

Solving assigns a concrete set of elements to each relation

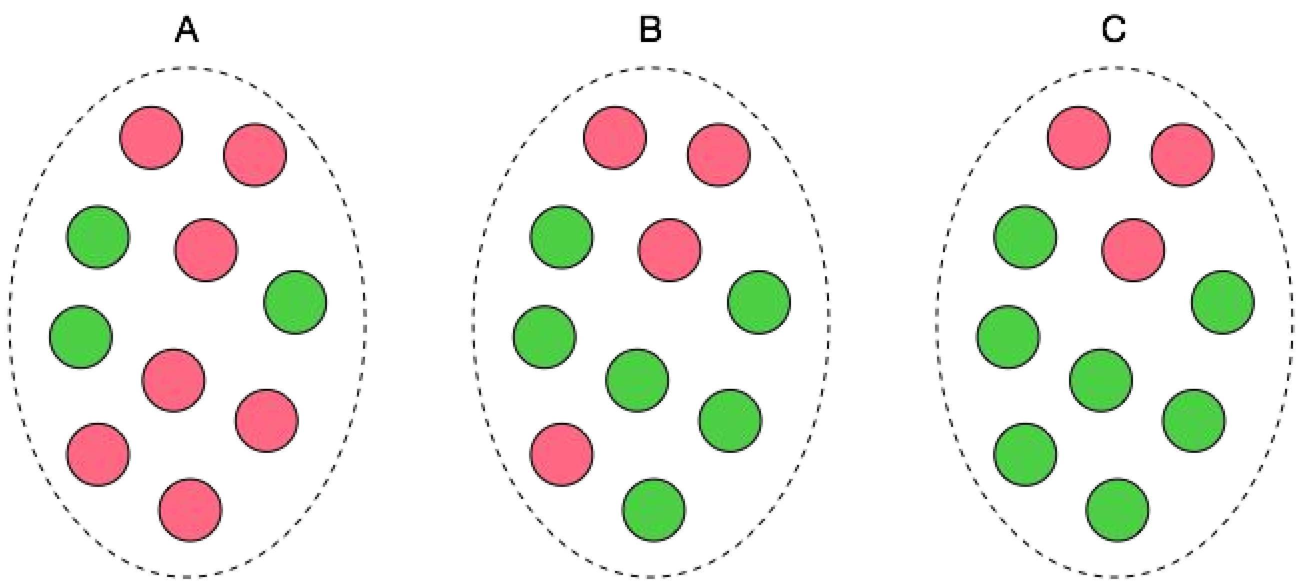


$$\phi = A \subseteq B \subseteq C \wedge \psi$$

Every (*non-symmetric*) solution can be inspected

Model Finding Scenario

Solving assigns a concrete set of elements to each relation



$$\phi = A \subseteq B \subseteq C \land \psi$$

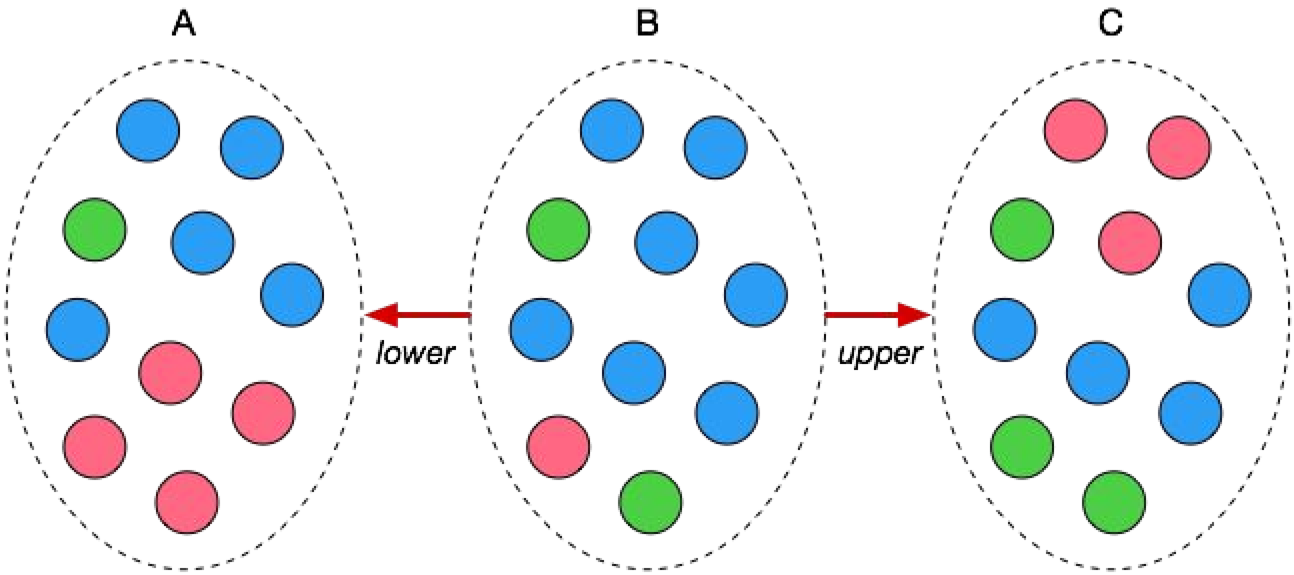
Every (*non-symmetric*) solution can be inspected

Exploiting Partial Knowledge

- Bounds allow users to express **partial knowledge** that could not be otherwise passed to the solvers
  - Can refer to concrete atoms
- These have proven to largely improve the performance of model finding, as they restrict the search space
- However, certain knowledge cannot be specified in bounds and must encoded in the constraint
  - Namely, *relationships* between the various relations
- This work explores this idea through the support for **symbolic bounds**

Model Finding Scenario Revisited

Symbolic bounds entail a *dependency* of *B* on *A* and *C*

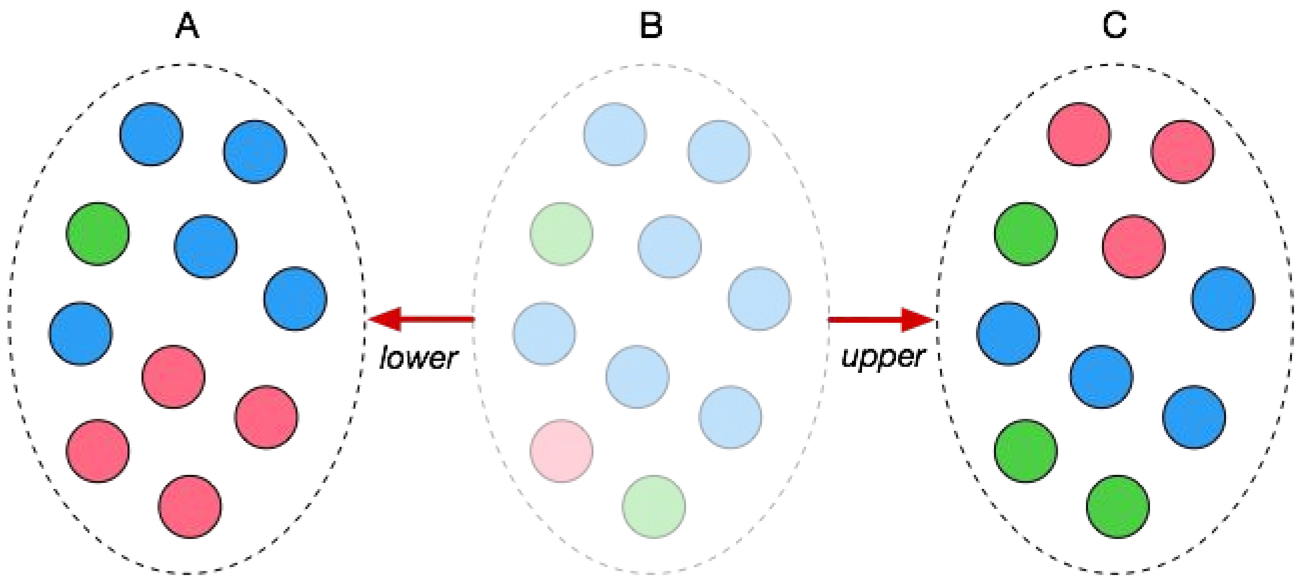


$$\phi = \psi$$

The constraint can be simplified

Model Finding Scenario Revisited

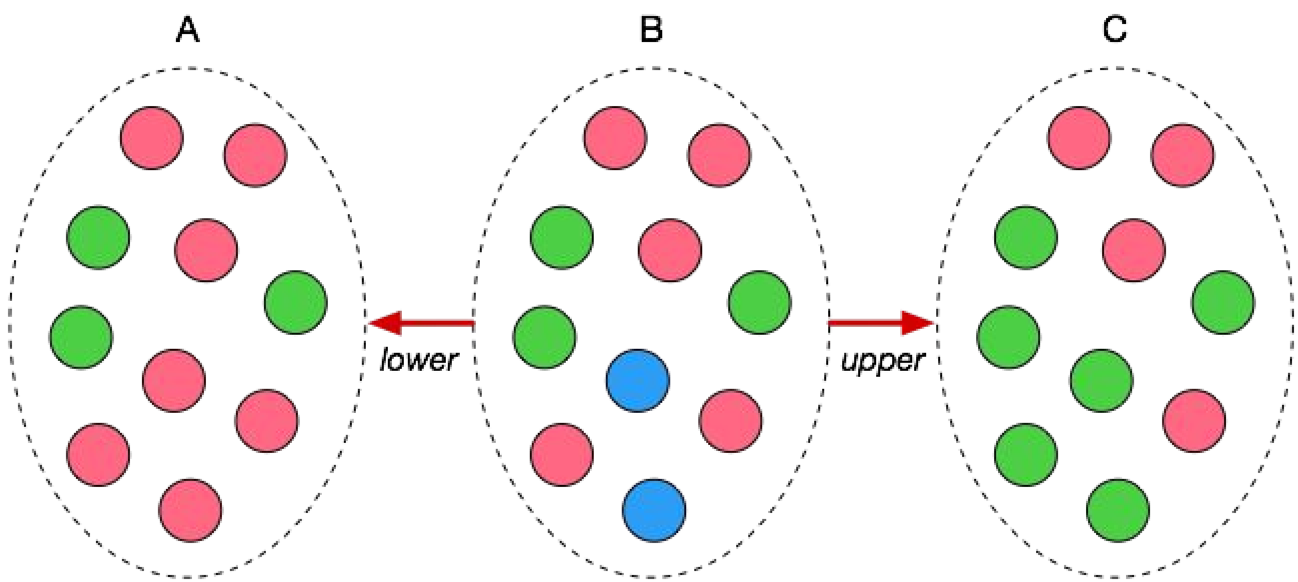
The problem can be *decomposed* based on the detected dependencies



$$\phi = \psi$$

Model Finding Scenario Revisited

Each solution of the *partial problem* restricts *B* by resolving the symbolic bounds

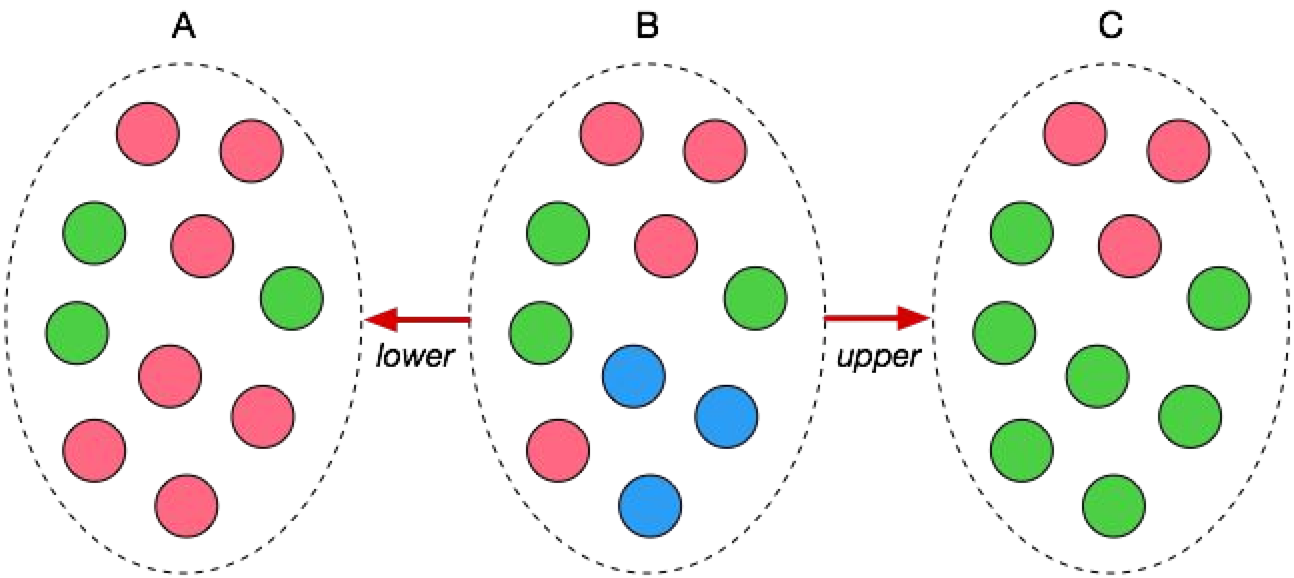


$$\phi = \psi$$

The search space for *B* is tighter

Model Finding Scenario Revisited

Each solution of the *partial problem* restricts *B* by resolving the symbolic bounds

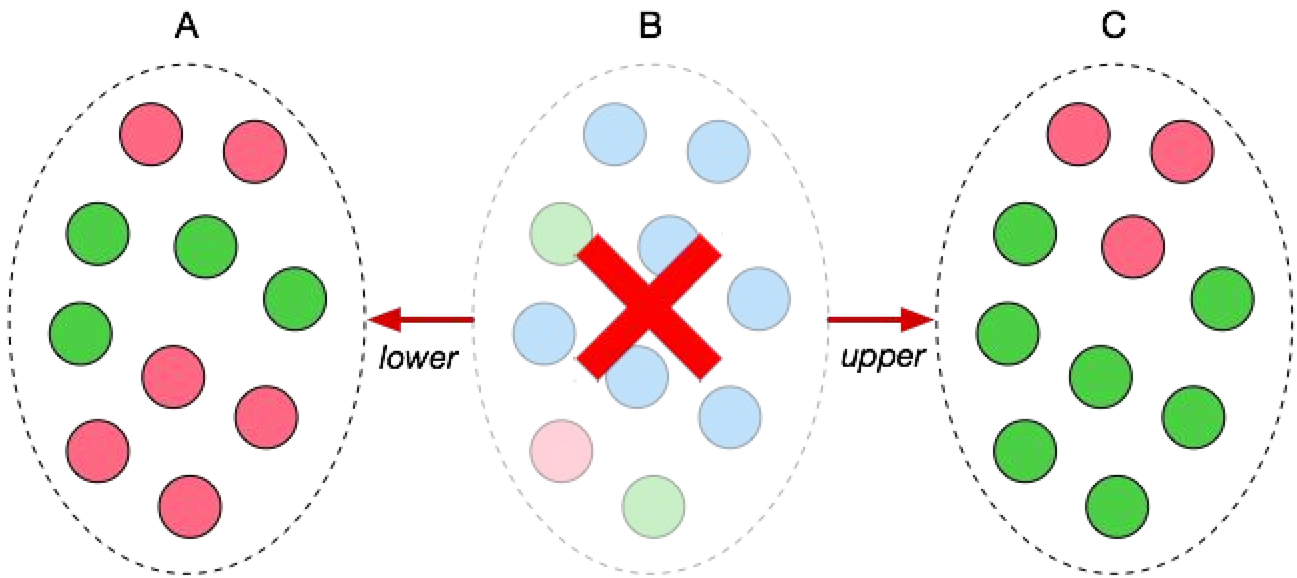


$$\phi = \psi$$

The search space for *B* is tighter

Model Finding Scenario Revisited

Not every partial solution leads to a complete solution



$$\phi = \psi$$

May potentially need to analyse every candidate

Decomposed Model Finding

Symbolic Bounds

- Regular bounds cannot encode such relationships: they refer only to concrete atom tuples
- We extend model finding to support symbolic bounds, arbitrary relational expressions that may refer to other relations
    - Can be evaluated efficiently
- If dependencies are constant, the bounds become tighter and the constraint may be removed from the problem
- Reduces the complexity of constraints
- More importantly, it can be exploited in a **decomposed solving** strategy

Symbolic Bounds

```
U = {R1,R2,K1,K2,G1,G2,T1,T2}

R = Time   : [{T1,T2},{T1,T2}]
    Key    : [{K1,K2},{K1,K2}]
    Room   : [{},{R1,R2}]
    Guest  : [{},{G1,G2}]
    keys   : [{},{(R1,K1),(R2,K1)
                ,(R1,K2),(R2,K2)}]
    guests : [{},{(R1,G1,T1),(R2,G1,T1),
              ...,(R1,G2,T2),(R2,G2,T2)}]
    g_keys : [{},{(G1,K1,T1),(G2,K1,T1),
              ...,(G1,K2,T2),(G2,K2,T2)}]
    ...

F = keys in Room -> Key &&
    guests in Room -> Guest -> Time &&
    g_keys in Guest -> Key -> Time &&
    all t:Time,r:Room | one r.r_keys.t &&
    all k:Key | one keys.k && ...
```

Explicit bounds

```
U = {R1,R2,K1,K2,G1,G2,T1,T2}

R = Room   : [{},{R1,R2}]
    Key    : [{K1,K2},{K1,K2}]
    Guest  : [{},{G1,G2}]
    keys   : [{},Room -> Key]
    Time   : [{T1,T2},{T1,T2}]
    guests : [{},Room -> Guest -> Time]
    g_keys : [{},Guest -> Key -> Time]
    ...

F = all t:Time,r:Room | one r.r_keys.t &&
    all k:Key | one keys.k && …
```

Symbolic bounds

Decomposed Solving

- Given a set of variables denoting the partial problem, slice the constraint into those formulas relevant for those variables
- Find a partial solution from this problem, and integrate it into the remainder problem bounds
  - Resolve the symbolic bounds of the remainder variables
  - Solve the integrated problem, whose solutions extend the partial solution
  - If UNSAT, find another partial solution and repeat the process
- If the partial problem is UNSAT, then the whole problem is UNSAT

Decomposed Solving

- Can be naturally encoded using Kodkod, as these *partial solutions* can be integrated in the bounds of the remaining variables
- Not every partial solution may lead to a complete solution, so potentially each one must be analyzed
    - Also relevant for solution iteration
- Efficient iteration of non-symmetric solutions renders the process feasible

Decomposed Solving

```
Room   : [{R1},{R1}]
Key    : [{K1,K2},{K1,K2}]
Guest  : [{G1},{G1}]
keys   : [{(R1,K1),(R1,K2)},
          {(R1,K1),(R1,K2)}]
```

Partial solution

```
Room   : [{},{R1,R2}]
Key    : [{K1,K2},{K1,K2}]
Guest  : [{},{G1,G2}]
keys   : [{},Room -> Key]
Time   : [{T1,T2},{T1,T2}]
guests : [{},Room -> Guest -> Time]
g_keys : [{},Guest -> Key -> Time]
...
```

Symbolic bounds

Decomposed Solving

```
Room   : [{R1},{R1}]
Key    : [{K1,K2},{K1,K2}]
Guest  : [{G1},{G1}]
keys   : [{(R1,K1),(R1,K2)},
          {(R1,K1),(R1,K2)}]
```

Partial solution

```
Room   : [{R1},{R1}]
Key    : [{K1,K2},{K1,K2}]
Guest  : [{G1},{G1}]
keys   : [{(R1,K1),(R1,K2)}
         ,{(R1,K1),(R1,K2)}]
Time   : [{T1,T2},{T1,T2}]
guests : [{},{(R1,G1,T1),(R1,G1,T2)}]
g_keys : [{},{(G1,K1,T1),(G1,K1,T1)
             ,(G1,K2,T2),(G1,K2,T2)}]
...
```

Integrated bounds

Implementation Details

- The general strategy exposed above is clearly prone to be **parallelized** on the analysis of the independent integrated problems
- The procedure is still sound and complete
  - The order of the solutions is more unpredictable, which is not necessarily negative
- UNSAT problems with a large number of partial solutions may predictably be much less efficient
- A **hybrid** approach was implemented, where the amalgamated problem runs in parallel for the worst-case scenario
- The implementation was carefully planned to preserve the soundness of the **symmetry** detection and predicate generation

Decomposition Criteria

- The variable decomposition can be provided by the user, who has domain knowledge
- Nonetheless, several automated decomposition criteria were tested
- We found out that a sensible criterion can be calculated from the **dependency graph** entailed by the symbolic bounds
- Roughly, relations with a *higher outdegree* in the dependency graph should be left for the second stage of solving
    - Intuitively, relations with more dependencies benefit more from prior solving

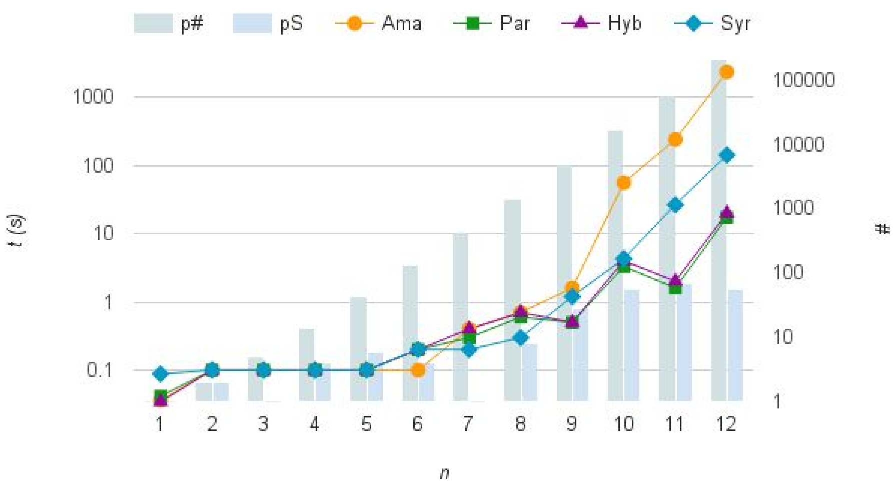
Evaluation

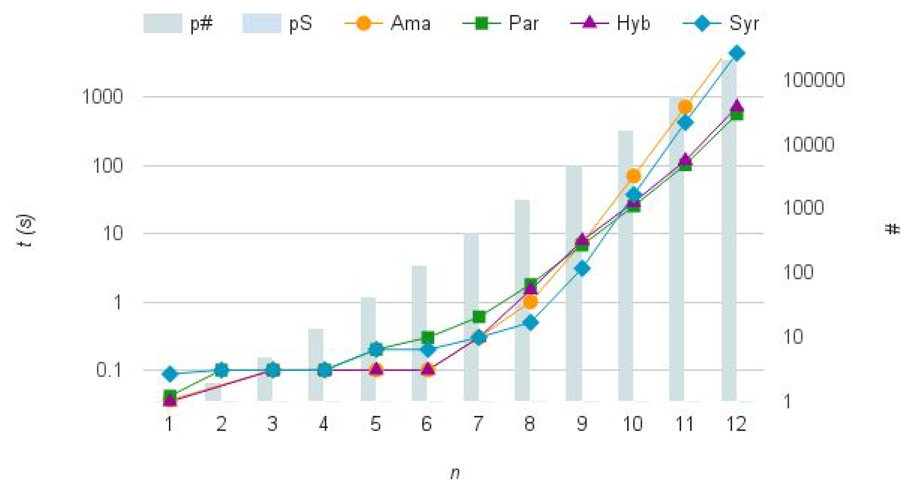
Evaluation Summary

- The decomposed strategy regularly outperforms sequential “amalgamated” model finding, and the hybrid technique balances out losses
  - For SAT, the decomposed strategy commonly outperforms the amalgamated procedure (up to 2 orders of magnitude)
  - For UNSAT, the hybrid approach balances out the losses (slowdowns never below 0.5)
- The strategy is “competitive” with state-of-the-art parallel SAT solvers
  - These are not incremental and thus cannot (efficiently) iterate solutions

Evaluation: Red-black tree

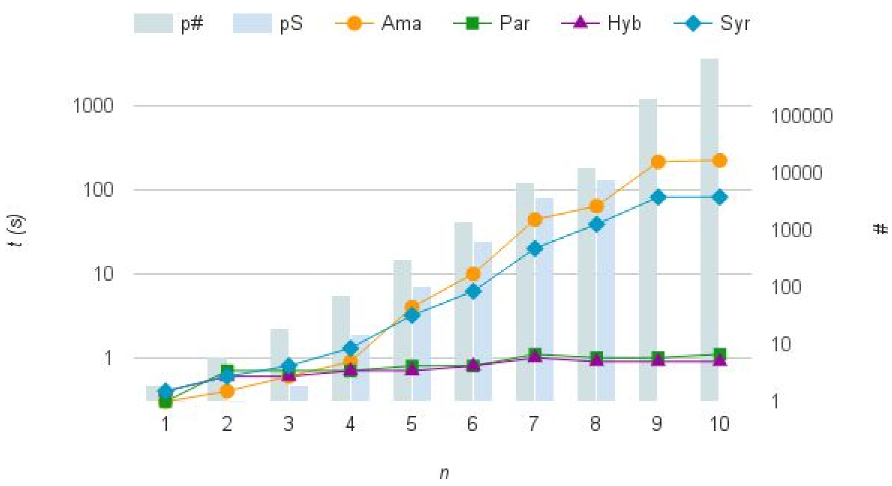
Red-black tree example, tree generation (SAT) and property checking (UNSAT)

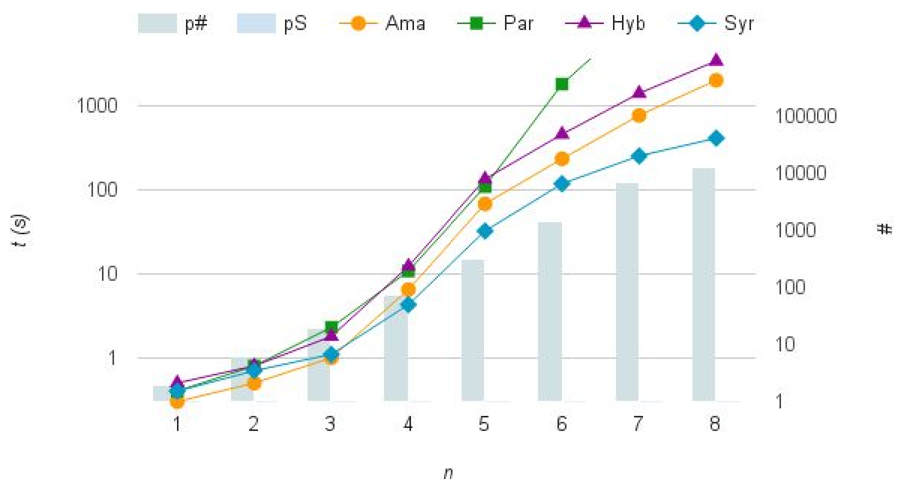




Evaluation: Hotel

Hotel room locking system example, with counter-examples (SAT) and corrected (UNSAT)





Conclusions

- Symbolic partial knowledge to enhance model finding
  - Decomposition criteria
  - Decomposed solving strategy
  - Tighter search space
- Fully automated process, preserves symmetry breaking and solution iteration
- Parallel implementation, efficient for SAT problems and balanced for UNSAT

- Working on extracting symbolic knowledge from higher-level specification languages