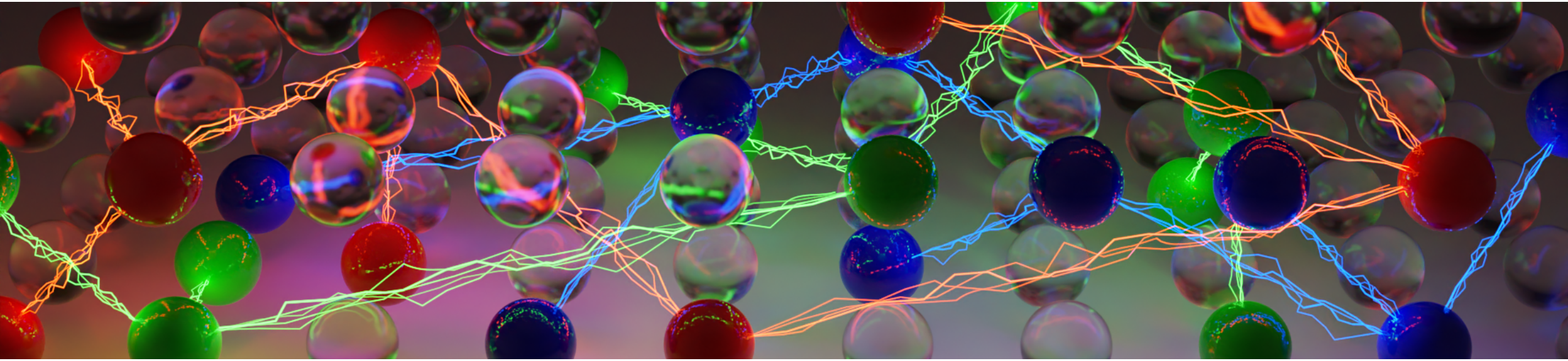
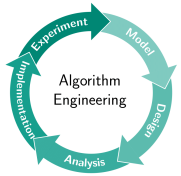


Practical SAT Solving



Lecture 10 – Planning & Model Checking

Markus Iser, Dominik Schreiber | July 7, 2025

SAtRes
Scalable Automated Reasoning

Overview

Recap Lecture 9

- Common state-of-the-art proof formats
- Pragmatics of proof production and checking
- Producing proofs with parallel + distributed solvers
- Beyond proof files: on-the-fly checking

Today: Applications I

- Why consider applications?
- Automated planning: Foundations and SAT-based methods
- From planning to checking: Bounded Model Checking basics

Why consider the Application View? (1/2)

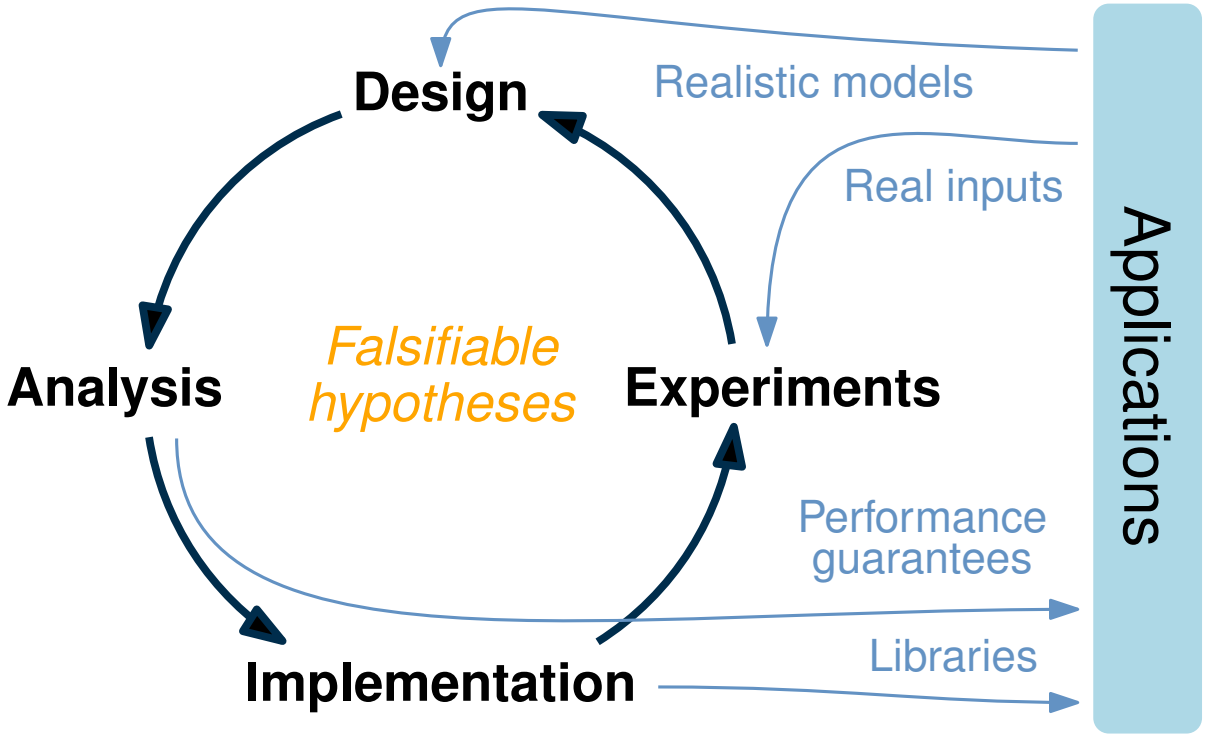
Result, instance family		1st author	Speedups of MallobSat over Kissat-MAB_HyWalk
SAT	Hypertree decomposition	Schidler	3, 5, 5, 5, 5, 7, 8, 9, 12, 13
SAT	Hamilton circle	Heule	4, 4, 7, 11, 17, 20, 21, 22, 24, 31, 33, 36, 42
SAT	Tree decomposition	Ehlers	5, 7, 87
UNSAT	Cellular automatons	Chowdhury	5, 8, 8, 9, 9, 10, 22, 22, 66
			⋮
UNSAT	Relativized pidgeon hole	Elffers	277, 542, 638
UNSAT	Bioinformatics	Bonet	292, 717
UNSAT	Balanced random	Spence	321, 388
SAT	Sum of three cubes	Riveros	384, 509, 1018, 3345
SAT	Circuit multiplication	Shunyang	17, 31, 32, 62, 105, 119, 213, 254, 393, 741, 746, 1401, 2650
UNSAT	Perfect matchings	Reeves	119, 413, 12 439, 108 593, 217 099

3072 cores (128 machines) of SuperMUC-NG · 400 problems from Int. SAT Competition 2021

Only instances with seq. time $\geq$ 60 s · Only families with $\geq$ 2 instances

Why consider the Application View? (2/2)

- Perform sound **algorithm engineering** with realistic applications in the loop
- Understand application-specific solver behavior, needs, shortcomings
- Advance the positive feedback loop between solvers and applications



SAT Competition Benchmarks – “Real” Applications?

5355 problems,
138 families + 72 unknown

Font size $\propto$ # problems

via `wordclouds.com`,
GBD
(`benchmark-database.de`)

Automated Planning: Introduction

Classical Automated Planning in a Nutshell

Given a set of world features, an according initial world state, and a set of conditional rules to modify the world state ("actions"), find a way to successively transform the initial world state until it satisfies some goal condition.

Properties of (classical) automated planning:

- Deterministic, full-knowledge, closed-world [12] ("*everything not explicitly true is assumed false*")
- Extremely generic "state transition system" model
  – essentially shortest path search but on a prohibitively large graph that must be generated on the go
- **PSPACE**-complete [3] – solutions may need to be exponentially long
- Applications of classical planning models mostly limited to highly idealized/simplified settings
  – But: Ideal "fruitfly problem" for considering interactions of applications with SAT solving
- Tons of extensions, enhancements, variants

Automated Planning: Definitions

Planning Problem (semi formal, simplified PDDL-style)

A **planning problem** is a tuple $P := (s_I, A, g)$, where s_I and g are sets of consistent propositional world state features (each true or false) and A is a set of actions. Each action has the shape $a = (pre_a, eff_a)$, where pre_a and eff_a are also sets of consistent propositional world state features. The **objective** is to find a plan $\Pi = \langle a_1, \ldots, a_n \rangle$ such that each pre_{a_k} is satisfied by $s_k := (s_I$ updated by $eff_{a_1}, \ldots, eff_{a_{k-1}})$ and g is satisfied by the final arising state s_{n+1}.

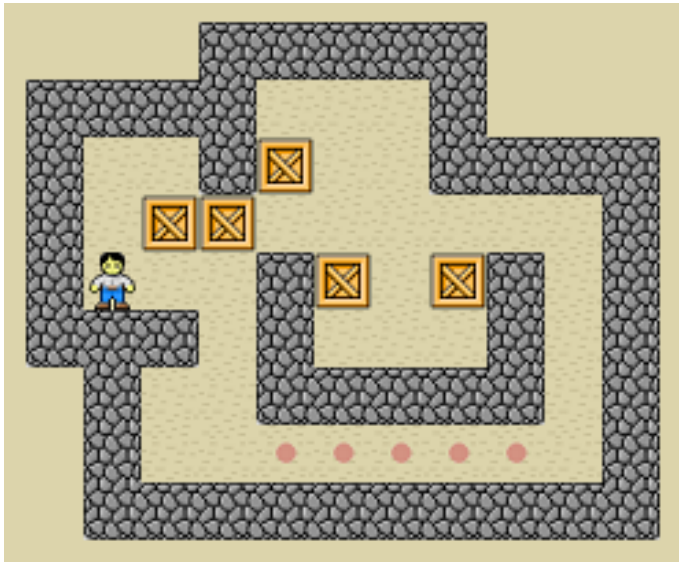
Automated Planning: Definitions

Planning Problem (semi formal, simplified PDDL-style)

A **planning problem** is a tuple $P := (s_I, A, g)$, where s_I and g are sets of consistent propositional world state features (each true or false) and A is a set of actions. Each action has the shape $a = (pre_a, eff_a)$, where pre_a and eff_a are also sets of consistent propositional world state features. The **objective** is to find a plan $\Pi = \langle a_1, \ldots, a_n \rangle$ such that each pre_{a_k} is satisfied by $s_k :=$ (s_I updated by $eff_{a_1}, \ldots, eff_{a_{k-1}}$) and g is satisfied by the final arising state s_{n+1}.

Example: Sokoban [5]

- **World state features:** Positions of walls, boxes, goals, player
- **Actions:** Walk and push. E.g., `push-right(x, y)`:
  - Preconditions: `player-at(x,y)`, `box-at(x+1,y)`, `¬wall-at(x+2,y)`, `¬box-at(x+2,y)`
  - Effects: `player-at(x+1,y)`, `¬box-at(x+1,y)`, `box-at(x+2,y)`
- **Goal:** `box-at(`x^*`,`y^*`)` for each marked location (x^*, y^*)
- **Plan:** Series of move/push actions leading to a goal state



Automated Planning

How do we solve planning problems?

- Most common: Forward state space search [8] – explore state space while heuristically closing in on a goal state
- More exotic: Backward search, plan-space search (not discussed here)
- Old and robust (since 1992 [10]): SAT-based planning

Automated Planning

How do we solve planning problems?

- Most common: Forward state space search [8] – explore state space while heuristically closing in on a goal state
- More exotic: Backward search, plan-space search (not discussed here)
- Old and robust (since 1992 [10]): SAT-based planning

(How) Can we encode planning problems into SAT?

Automated Planning

How do we solve planning problems?

- Most common: Forward state space search [8] – explore state space while heuristically closing in on a goal state
- More exotic: Backward search, plan-space search (not discussed here)
- Old and robust (since 1992 [10]): SAT-based planning

(How) Can we encode planning problems into SAT?

- We cannot translate planning to a single polynomially sized SAT encoding (NP vs. PSPACE?)
  – Unclear how many steps to encode!
- We can translate planning to a compact SAT encoding for a fixed number of K steps
  $\Rightarrow$ Incrementally increase K until plan is found!
- Main aspects of design space: Used SAT encoding, Scheduling strategies for values of K

Automated Planning: A SAT Encoding [9] (1/2)

1. The initial state has to hold at time 0: (Assume s_I explicitly defines *all relevant/reachable propositions*)

$$\forall \ell \in s_I : \quad \ell^{(0)}$$

2. Apply at most one action at each time: (Optionally also add "*At least one* action")

$$\forall t \in \{0, \ldots, K\} \;\; \forall a, a' \in A : \quad \neg a^{(t)} \vee \neg a'^{(t)}$$

3. Applying an action at time t implies its preconditions at time t:

$$\forall t \in \{0, \ldots, K\} \;\; \forall a \in A \;\; \forall \ell \in pre_a : \quad a^{(t)} \rightarrow \ell^{(t)}$$

4. Applying an action at time t implies its effects at time $t+1$:

$$\forall t \in \{0, \ldots, K\} \;\; \forall a \in A \;\; \forall \ell \in eff_a : \quad a^{(t)} \rightarrow \ell^{(t+1)}$$

5. Each goal has to hold at time K:

$$\forall \ell \in g : \quad \ell^{(K)}$$

Automated Planning: A SAT Encoding (2/2)

We are done, right?

Automated Planning: A SAT Encoding (2/2)

We are done, right?
$\Rightarrow$ Sokoban example: SAT Solver finds solution at $K = 1$, where boxes magically materialize at all goal spots.
What did we forget?

Automated Planning: A SAT Encoding (2/2)

We are done, right?
$\Rightarrow$ Sokoban example: SAT Solver finds solution at $K = 1$, where boxes magically materialize at all goal spots.
What did we forget?

6. **Frame axioms:** A state feature only changes if an action supports this change:

$$\forall t \in \{0, \ldots, K-1\} \;\; \forall \ell \in s_I \cup \bar{s}_I : \quad \bar{\ell}^{(t)} \wedge \ell^{(t+1)} \rightarrow \bigvee_{a \in A \mid \ell \in \mathit{eff}_a} a^{(t)}$$

Proposition

Using clause rules 1–6, encoding a planning problem P to SAT for some fixed $K \geq 0$ yields a satisfiable CNF formula if and only if there exists a plan Π for P with at most K steps.
Such a Π can be decoded from a model by reading the satisfying assignment to the variables $a^{(t)}$.

SAT Planning

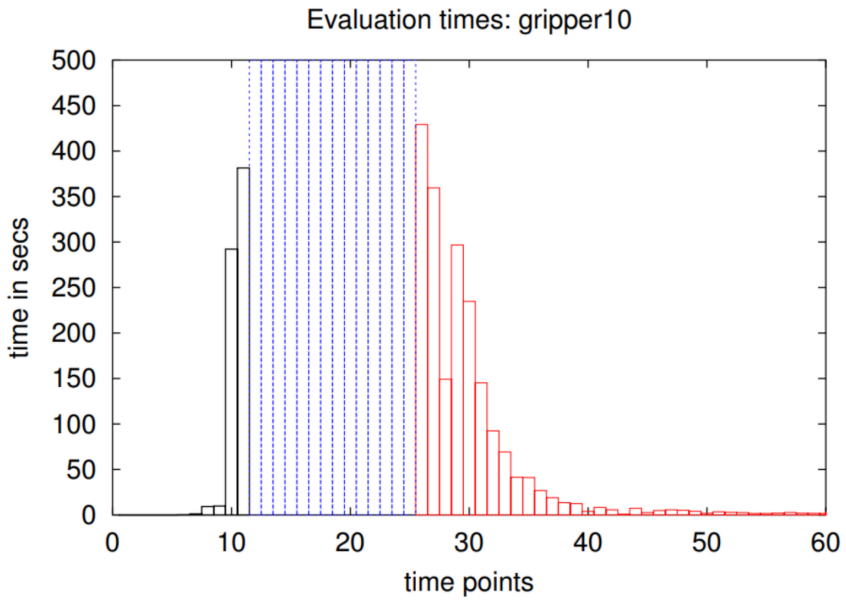
Improvements to SAT-based planning

- More efficient encodings, in particular for 2. (“at most one action”)
- Relaxed semantics: Execute several actions at once (e.g., [1])
- Exploit incremental SAT solving! [7]
  - Encode 1. initially
  - Encode 2.,3.,4.,6. at every new increment
  - Enforce 5. (“goal reached”) as temporary assumptions at each SAT call

SAT Planning

Improvements to SAT-based planning

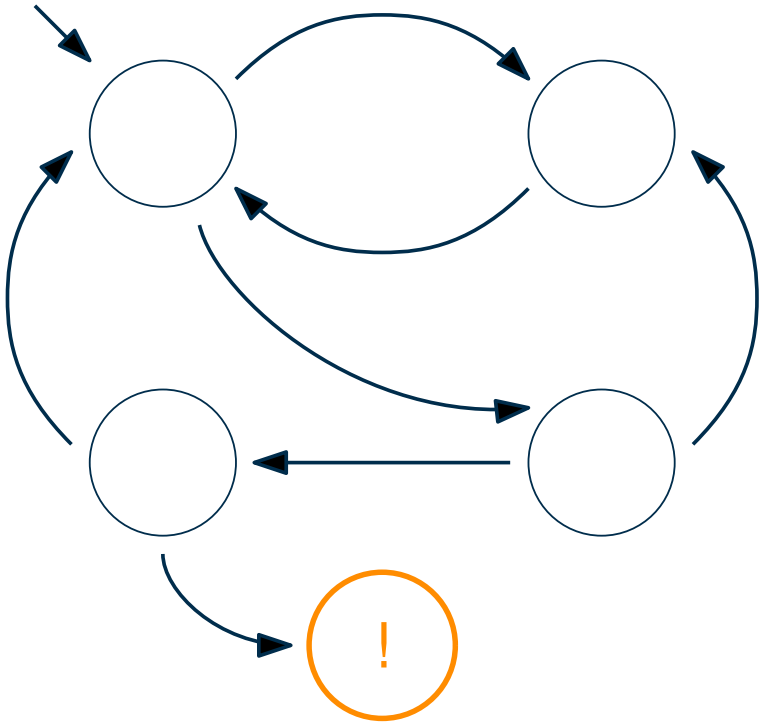
- More efficient encodings, in particular for 2. (“at most one action”)
- Relaxed semantics: Execute several actions at once (e.g., [1])
- Exploit incremental SAT solving! [7]
  - Encode 1. initially
  - Encode 2.,3.,4.,6. at every new increment
  - Enforce 5. (“goal reached”) as temporary assumptions at each SAT call
- Find more effective sequences of values for K to test (“makespan scheduling”) [13]
  - Try to bypass unsurmountable wall of exceedingly difficult K (esp. UNSAT)
  - Increase K linearly, exponentially, . . .
  - Test several K in parallel



From Planning to Verification?

Let's use our SAT-based planning to check the correctness of a system!

- World state features ≡ State features of our system
- Actions ≡ Valid transitions between states
- Goals ≡

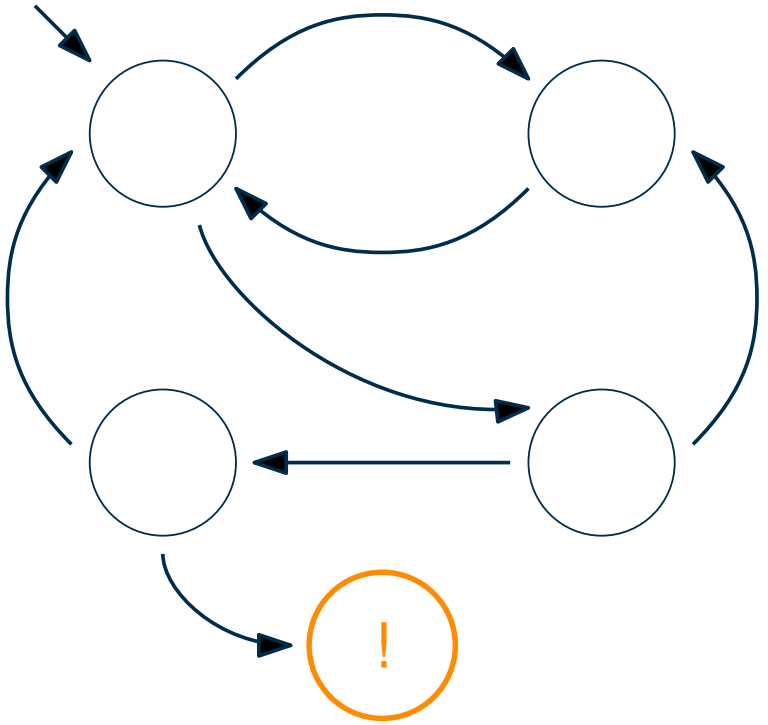


A snack machine or an electronic component or a C program or . . .

From Planning to Verification?

Let's use our SAT-based planning to check the correctness of a system!

- World state features ≡ State features of our system
- Actions ≡ Valid transitions between states
- Goals ≡ incorrect state, violating some constraint
- Plan ≡

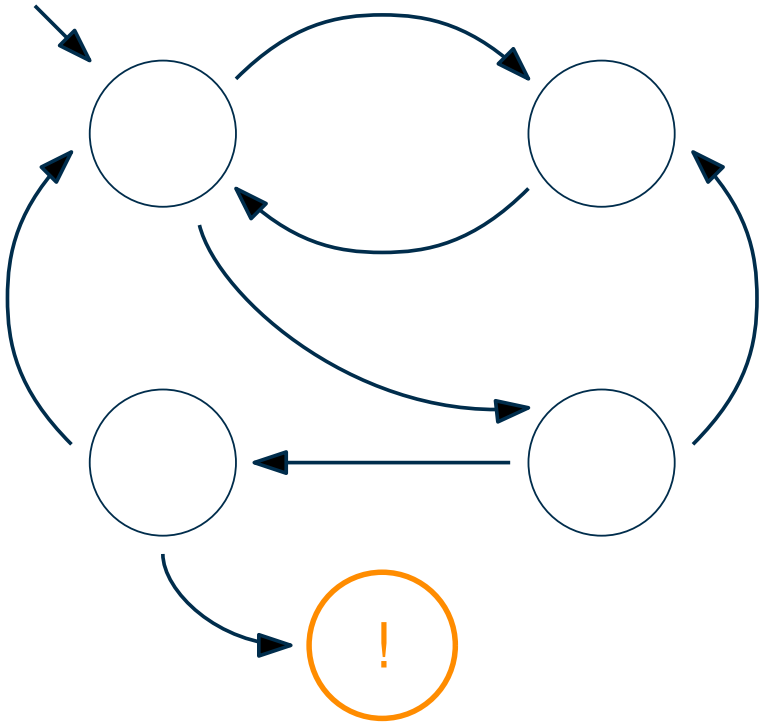


A snack machine or an electronic component or a C program or . . .

From Planning to Verification?

Let's use our SAT-based planning to check the correctness of a system!

- World state features ≡ State features of our system
- Actions ≡ Valid transitions between states
- Goals ≡ incorrect state, violating some constraint
- Plan ≡ reachable incorrectness
- Unsatisfiability ≡ system is always correct?

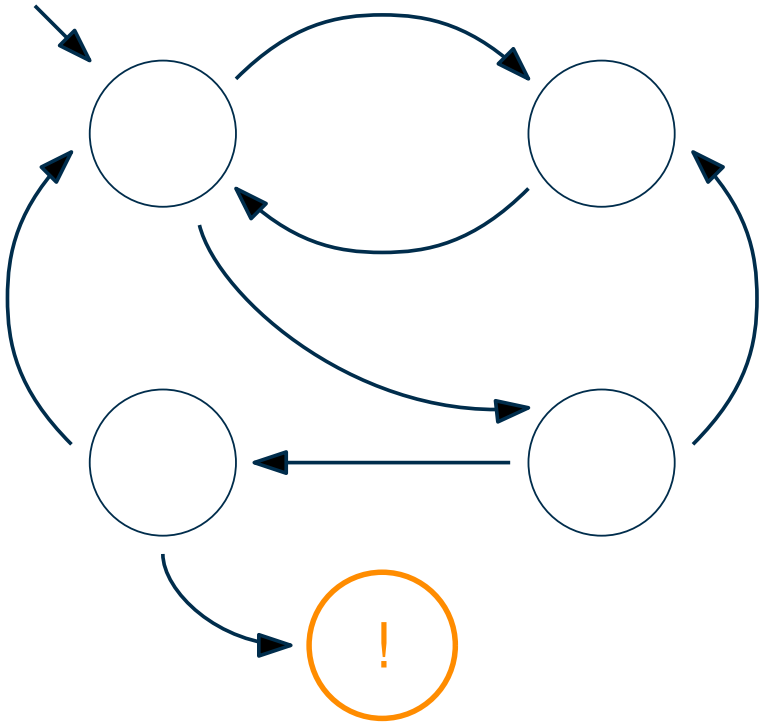


A snack machine or an electronic component or a C program or . . .

From Planning to Verification?

Let's use our SAT-based planning to check the correctness of a system!

- World state features ≡ State features of our system
- Actions ≡ Valid transitions between states
- Goals ≡ incorrect state, violating some constraint
- Plan ≡ reachable incorrectness
- Unsatisfiability at k steps ≡ system is correct within k steps



A snack machine or an electronic component or a C program or . . .

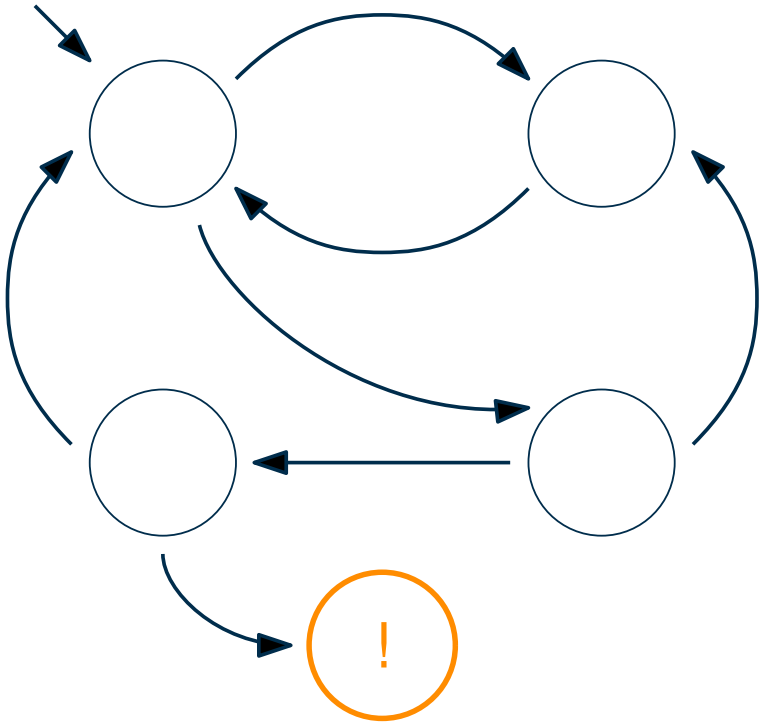
From Planning to Verification?

Let's use our SAT-based planning to check the correctness of a system!

- World state features $\equiv$ State features of our system
- Actions $\equiv$ Valid transitions between states
- Goals $\equiv$ incorrect state, violating some constraint
- Plan $\equiv$ reachable incorrectness
- Unsatisfiability at k steps $\equiv$ system is correct within k steps

Bounded Model Checking

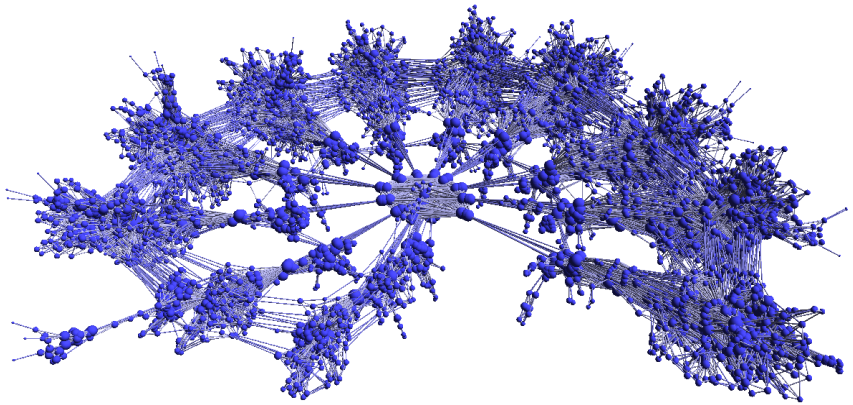
- Encode and check transition system for $k = 1, 2, \ldots$
- Satisfying assignment $\equiv$ counter example!
- Crucial tool for hardware and software verification [14]



A snack machine or an electronic component or a C program or ...

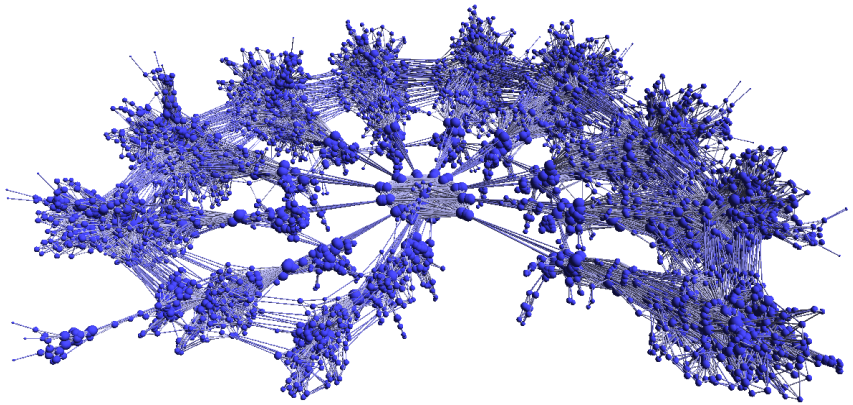
Bounded Model Checking

- Invented by Clarke & Biere in ∼2000 [4], mostly replacing BDD-based model checking
- State transition system based on temporal logic (LTL, CTL, . . .); solving via SAT or SMT
- Applications: Computer-aided design (CAD), software verification, invariant checking, bug detection, . . . [14]



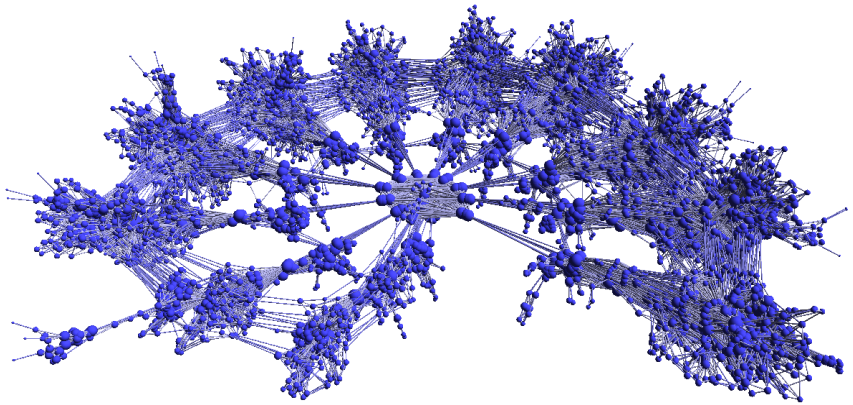
Bounded Model Checking

- Invented by Clarke & Biere in ∼2000 [4], mostly replacing BDD-based model checking
- State transition system based on temporal logic (LTL, CTL, . . .); solving via SAT or SMT
- Applications: Computer-aided design (CAD), software verification, invariant checking, bug detection, . . . [14]
- One of the most essential real-world applications of SAT
  - Pushed industrial interest in SAT solvers in 2000s
  - Actively influenced solver design and algorithms
  - Some of the largest, structurally most distinct benchmarks



Bounded Model Checking

- Invented by Clarke & Biere in ∼2000 [4], mostly replacing BDD-based model checking
- State transition system based on temporal logic (LTL, CTL, . . .); solving via SAT or SMT
- Applications: Computer-aided design (CAD), software verification, invariant checking, bug detection, . . . [14]
- One of the most essential real-world applications of SAT
  - Pushed industrial interest in SAT solvers in 2000s
  - Actively influenced solver design and algorithms
  - Some of the largest, structurally most distinct benchmarks
- Examples for BMC @ KIT:
  - Low-Level Bounded Model Checker (LLBMC) [6] (C program verification)
  - Verification of Java contracts [2] (see right)
  - Cryptography [11]



```
/*@ requires 0 <= x1;
  @ ensures \result == x1 * x2;
  @ assignable \nothing;
  @*/
public int mult(int x1, int x2) {
  int res = 0;
  /*@ loop_invariant 0 <= i && i <= x1 && res == i * x2;
    @ decreases x1 - i;
    @ assignable \nothing;
    @*/
  for (int i = 0; i < x1; ++i) res += x2;
  return res;
}
```

Recap

Applications I – Planning & Checking

- Why consider applications?
- Automated planning: Foundations and SAT-based methods
- From planning to checking: Bounded Model Checking basics

Next Up: Applications II – Highlights

- Electronic design
- Cryptanalysis
- Path finding and scheduling
- ...

References I

[1] Toma Balyo. „Relaxing the relaxed exist-step parallel planning semantics“. In: *2013 IEEE 25th International Conference on Tools with Artificial Intelligence*. IEEE. 2013, S. 865–871.

[2] Bernhard Beckert u. a. „Modular verification of JML contracts using bounded model checking“. In: *Int. Symposium on Leveraging Applications of Formal Methods (ISoLA)*. Springer. 2020, S. 60–80.

[3] Tom Bylander. „The computational complexity of propositional STRIPS planning“. In: *Artificial Intelligence* 69.1-2 (1994), S. 165–204. DOI: 10.1016/0004-3702(94)90081-7.

[4] Edmund Clarke u. a. „Bounded model checking using satisfiability solving“. In: *Formal methods in system design* 19 (2001), S. 7–34. DOI: 10.1023/A:1011276507260.

[5] Joseph Culberson. „Sokoban is PSPACE-complete“. In: *Technical report, Department of Computing Science, University of Alberta* (1997).

[6] Stephan Falke, Florian Merz und Carsten Sinz. „The bounded model checker LLBMC“. In: *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE. 2013, S. 706–709.

[7] Stephan Gocht und Tomas Balyo. „Accelerating SAT Based Planning with Incremental SAT Solving“. In: *Proc. ICAPS*. 2017, S. 135–139. DOI: 10.1609/icaps.v27i1.13798.

[8] Malte Helmert. „The Fast Downward planning system“. In: *Journal of Artificial Intelligence Research* 26 (2006), S. 191–246.

[9] Henry Kautz und Bart Selman. „Pushing the envelope: Planning, propositional logic, and stochastic search“. In: *AAAI Conference on Artificial Intelligence*. 1996, S. 1194–1201.

[10] Henry A. Kautz und Bart Selman. „Planning as Satisfiability“. In: *Proc. ECAI*. Bd. 92. Citeseer. 1992, S. 359–363.

References II

[11] Alexander Koch, Michael Schrempp und Michael Kirsten. „Card-based cryptography meets formal verification“. In: *New Generation Computing* 39.1 (2021), S. 115–158. DOI: 10.1007/s00354-020-00120-0.

[12] Raymond Reiter. „On closed world data bases“. In: *Readings in artificial intelligence*. Elsevier, 1981, S. 119–140. DOI: 10.1016/b978-0-934613-03-3.50014-3.

[13] Jussi Rintanen. „Evaluation strategies for planning as satisfiability“. In: *Proc. ECAI*. Bd. 16. 2004, S. 682.

[14] Yakir Vizel, Georg Weissenbacher und Sharad Malik. „Boolean Satisfiability Solvers and Their Applications in Model Checking“. In: *IEEE*. Bd. 103. 11. 2015, S. 2021–2035. DOI: 10.1109/JPROC.2015.2455034.