# R.E.C.A.R

## Recursive Explore and Check Abstraction Refinement

Jean-Marie Lagniez, Daniel Le Berre,
Tiago de Lima and Valentin Montmirail

CRIL-CNRS UMR 8188, F62300 Lens, France

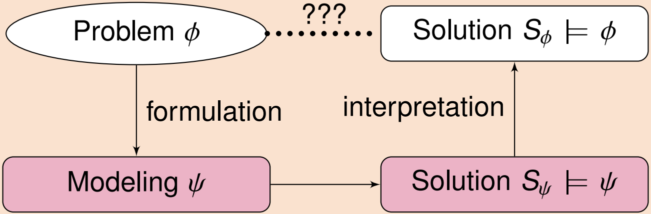
Séminaire CRIL - Lens - September 14th 2017

# Introduction: Abstraction



## Abstraction: Idea & Motivation

- Comes from: **Mathematical Modeling**



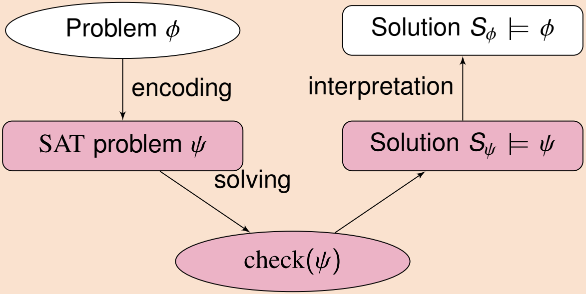

- Works for theoretical problems
- But what about practice?

# Introduction: Abstraction via SAT



## Modeling: Propositional Formula

- For many **NP** problems: Encoding into SAT




- Is it always a good idea?

## SAT solver

- Extremely efficient software
- Based on CDCL approach [SS99, MMZ+01]
- One of the current best is: Glucose [ES03a, AS09] ☺
- Able to solve efficiently problems with $\approx 10^8$ variables/clauses

# SAT solver



## SAT solver: Features

- Answer *SAT* and a model when the formula is satisfiable
- Answer *UNSAT*:
  - with a proof of unsatisfiability if asked [Gel02]
  - A unsatisfiable core if asked [ES03a]
- Can work in an incremental way [ES03b, ES03a, ALS13]
- Can work under assumptions [ES03a]

## Unsatisfiable core

Basically the “reason” why a formula is UNSAT (subset of clauses)

### SAT solver: One limitation

- What happen when the encoding of the problem is too big ?
- Could be solved ‘easily’ but will not because of memory...

### HCP via SAT: does not scale

- Ex. The Hamiltonian Cycle Problem (HCP)
- HCP: $\mathcal{O}(n^3)$ clauses [Pre03]
- Transitive relations for any three nodes
- HCP via SAT: hard to solve HCP of over 1000 nodes
- HCP solver ‘LKH’ scales up to 10,000 nodes

We need a SAT solver in a more complex procedure...

# SAT solver: how to solve HCP efficiently?



$V$ is a set of $n$ nodes, $A$ is a set of vertexes, and G = (V,A) is a digraph. $x_{ij} = 1 \leftrightarrow (i,j) \in A$ is used in a solution cycle.

$$\sum_{(i,j)\in A} x_{ij} = 1 \qquad \text{for each i = 1,...,n (out-degree)}$$

$$\sum_{(i,j)\in A} x_{ij} = 1 \qquad \text{for each j = 1,...,n (in-degree)}$$

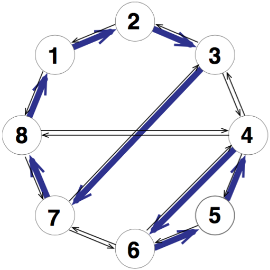
$$\sum_{(i,j)\in S} x_{ij} \leq |S| - 1 \qquad S \subset V, 2 \leq |S| \leq n-2 \text{ (connectivity)}$$

- in/out-degree constraints ensure that in/out-degrees are respectively exact one for each node in solution cycles
- connectivity constraint prohibits the formulation of sub-cycles

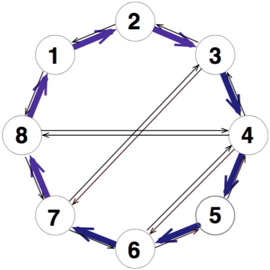
# SAT solver: how to solve HCP efficiently?



- With only in/out-degree constraints, we have cycles but they may not be connected (Case A)
- With all constraints, we can find a Hamiltonian cycle (Case B)



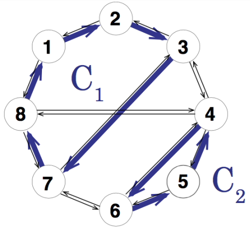

(Case A)
in/out-degree




(Case B)
in/out-degree + connectivity

## HCP via SAT: no need to generate connectivity constraints

- Refine overall constraints by adding blocking clauses generated from counter examples [SLR+14].
- We can get lucky and find a Hamiltonian Cycle quickly




**Blocking Clauses**

$C_1$ $\neg x_{12} \vee \neg x_{23} \vee \neg x_{37} \vee \neg x_{78} \vee \neg x_{81}$

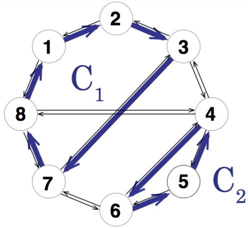
$C'_1$ $\neg x_{87} \vee \neg x_{73} \vee \neg x_{32} \vee \neg x_{21} \vee \neg x_{18}$

$C_2$ $\neg x_{46} \vee \neg x_{65} \vee \neg x_{54}$

$C'_2$ $\neg x_{45} \vee \neg x_{56} \vee \neg x_{64}$

## HCP via SAT: no need to generate connectivity constraints

- Refine overall constraints by adding blocking clauses generated from counter examples [SLR+14].
- We can get lucky and find a Hamiltonian Cycle quickly




**Blocking Clauses**

| | |
|---|---|
| $C_1$ | $\neg x_{12} \vee \neg x_{23} \vee \neg x_{37} \vee \neg x_{78} \vee \neg x_{81}$ |
| $C_1'$ | $\neg x_{87} \vee \neg x_{73} \vee \neg x_{32} \vee \neg x_{21} \vee \neg x_{18}$ |
| $C_2$ | $\neg x_{46} \vee \neg x_{65} \vee \neg x_{54}$ |
| $C_2'$ | $\neg x_{45} \vee \neg x_{56} \vee \neg x_{64}$ |

This idea of going step by step and refining each step is called:

**CEGAR**: CounterExample Guided Abstraction Refinement

# CounterExample Guided Abstraction Refinement

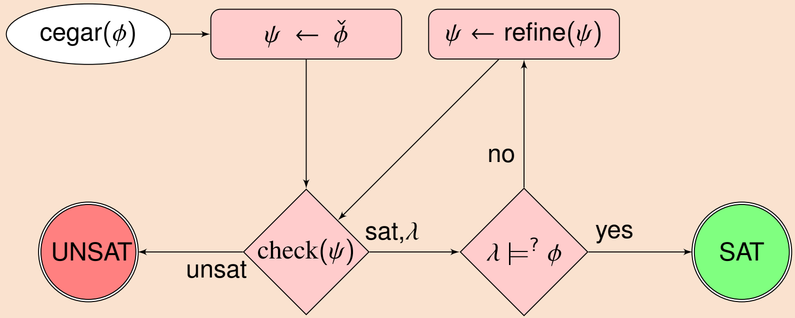


**CEGAR**: **C**ounter**E**xample **G**uided **A**bstraction **R**efinement

To solve a problem, we may need to consider only a small part of it [CGJ+03]

- To abstract problems: hoping it will be easier to solve
- Two variants of abstraction:
  - Under-abstraction: abstraction has **more** solutions
  - Over-abstraction: abstraction has **less** solutions
- CEGAR-over: CEGAR approach using over-abstractions
- CEGAR-under: CEGAR approach using under-abstractions

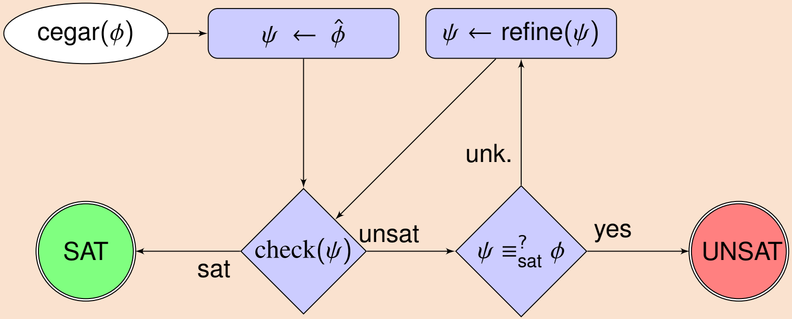
## CEGAR-under




### Example

SAT problem, by increasing step by step, the number of clauses

## CEGAR-over




## Example

Planification problem, by increasing step by step, the horizon

# CounterExample Guided Abstraction Refinement



## Advantages

- If problem mainly satisfiable: CEGAR-over
- If problem mainly unsatisfiable: CEGAR-under
- Everytime $\text{check}$ improves, CEGAR improves
- Many applications already use CEGAR

## Drawbacks

- Not efficient when 50/50 chances of being SAT/UNSAT
- Not efficient when we need many refinement steps

### **R**ecursive **E**xplore and **C**heck **A**bstraction **R**efinement

- Called *RECAR* [LLdLM17]
- Inspired by CEGAR [CGJ+03]
- Rely on 5 very important assumptions

### RECAR Assumptions

1. Function 'check' is sound, complete and terminates
2. $isSAT(\hat{\phi})$ implies $isSAT(\text{refine}(\hat{\phi}))$
3. $\exists.n \in \mathbb{N}$ s.t. $\text{refine}^n(\hat{\phi}) \equiv^{?}_{\text{sat}} \phi$.
4. $\text{isUNSAT}(\check{\phi})$ implies $\text{isUNSAT}(\phi)$
5. $\exists n \in \mathbb{N}$ s.t. $RC(under^n(\phi), under^{n+1}(\phi))$ is false.

# Recursive Explore and Check Abstraction Refinement

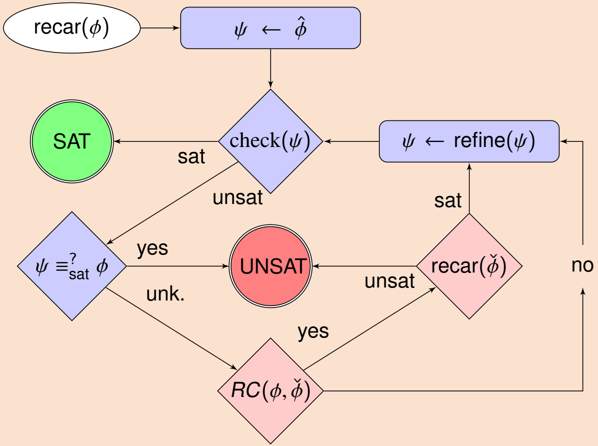


$\exists n \in \mathbb{N}$ s.t. $RC(under^{n}(\phi), under^{n+1}(\phi))$ is false.

## RC function

- 'true' if we can do a recursive call, 'false' otherwise
- It compares $under^{i}(\phi)$ and $under^{i+1}(\phi)$
- It checks if $under^{i+1}(\phi)$ will be "easier to solve" than $under^{i}(\phi)$

## RECAR



recar($\phi$)
$\psi \leftarrow \hat{\phi}$
check($\psi$)
sat
SAT
unsat
$\psi \leftarrow$ refine($\psi$)
sat
$\psi \equiv^?_{sat} \phi$
yes
UNSAT
unsat
recar($\check{\phi}$)
no
unk.
yes
$RC(\phi, \check{\phi})$

## RECAR

- 2 levels of abstractions
  - One at the Oracle level ($\mathrm{check}(\psi)$)
  - One at the Domain level (recursive call)
- Efficient even when 50/50 chance of being SAT/UNSAT
- Everytime $\mathrm{check}$ improves, RECAR improves
- The return of the recursive call can reduce the number of refinement
- Totally generic, can change SAT solver $\rightarrow$ FO solver?

### RECAR for Modal Logic K

- Modal Logic K is **PSPACE**-complete [Lad77, Hal95]
- What is Modal Logic K?
- How we over-approximate a formula $\phi$?
- How we under-approximate a formula $\phi$?
- Is it competitive against a CEGAR approach?
- Is it competitive against the state-of-the-art approaches?

# Preliminaries: Modal Logic



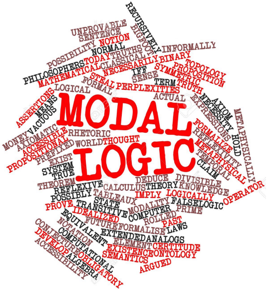
Modal Logic = Propositional Logic + $\Box$ and $\Diamond$

## Modal Logic

- $\Box\phi$ means $\phi$ is necessarily true
- $\Diamond\phi$ means $\phi$ is possibly true

$$\Diamond\phi \leftrightarrow \neg\Box\neg\phi$$

$$\Box\phi \leftrightarrow \neg\Diamond\neg\phi$$



# Preliminaries: Kripke Structure



- $\mathbb{P}$ finite non-empty set of propositional variables

**Kripke Structure [Kri59]**

$M = \langle W, R, V \rangle$ with:

- $W$, a non-empty set of possible worlds
- $R$, a binary relation on $W$
- $V$, a function that associate to each $p \in \mathbb{P}$, the set of possible worlds where $p$ is true

Pointed Kripke Structure: $\langle \mathcal{K}, w \rangle$

- $\mathcal{K}$: Kripke Structure
- $w$: a possible world in W

# Preliminaries: Satisfaction Relation



### Definition (Satisfaction Relation)

The relation $\models$ between Kripke Structures and formulae is recursively defined as follows:

$$
\begin{array}{lll}
\langle \mathcal{K}, w \rangle \models p & \text{iff} & w \in V(p) \\
\langle \mathcal{K}, w \rangle \models \neg\phi & \text{iff} & \langle \mathcal{K}, w \rangle \not\models \phi \\
\langle \mathcal{K}, w \rangle \models \phi_1 \wedge \phi_2 & \text{iff} & \langle \mathcal{K}, w \rangle \models \phi_1 \text{ and } \langle \mathcal{K}, w \rangle \models \phi_2 \\
\langle \mathcal{K}, w \rangle \models \phi_1 \vee \phi_2 & \text{iff} & \langle \mathcal{K}, w \rangle \models \phi_1 \text{ or } \langle \mathcal{K}, w \rangle \models \phi_2 \\
\langle \mathcal{K}, w \rangle \models \Box\phi & \text{iff} & (w, w') \in R \text{ implies } \langle \mathcal{K}, w' \rangle \models \phi \\
\langle \mathcal{K}, w \rangle \models \Diamond\phi & \text{iff} & (w, w') \in R \text{ and } \langle \mathcal{K}, w' \rangle \models \phi
\end{array}
$$

$\mathcal{K}$ that satisfied a formula $\phi$ will be called “Kripke model of $\phi$”

# Preliminaries: Example of a Kripke Structure



✓ $\phi_1 = \Box(\textcolor{cyan}{\bullet})$

✗ $\phi_2 = \Box\Diamond(\textcolor{orange}{\bullet})$

✓ $\phi_3 = \Diamond(\textcolor{red}{\bullet} \wedge \Diamond\neg\textcolor{green}{\bullet})$

✓ $\phi_4 = (\textcolor{cyan}{\bullet} \vee \textcolor{red}{\bullet} \vee \textcolor{orange}{\bullet})$

✗ $\phi_5 = \Diamond\Diamond(\textcolor{orange}{\bullet} \wedge \Box\neg\textcolor{orange}{\bullet})$

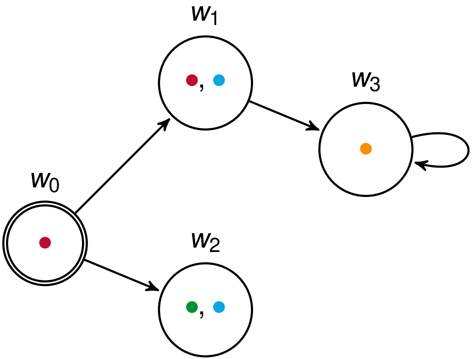



Figure: Example $\mathcal{K}$

## MoSaiC

- Open-Source Modal Logic K solver
- Uses Glucose as internal SAT solver
- Uses a RECAR approach

## MoSaiC

- Open-Source Modal Logic K solver
- Uses Glucose as internal SAT solver
- Uses a RECAR approach

## RECAR Assumptions: Reminder

✓1 Function 'check' is sound, complete and terminates

?2 $isSAT(\hat{\phi})$ implies $isSAT(\text{refine}(\hat{\phi}))$

?3 $\exists .n \in \mathbb{N}$ s.t. $\text{refine}^n(\hat{\phi}) \equiv^?_{\text{sat}} \phi$

4 $\text{isUNSAT}(\check{\phi})$ implies $\text{isUNSAT}(\phi)$

5 $\exists n \in \mathbb{N}$ s.t. $RC(under^n(\phi), under^{n+1}(\phi))$ is false

# MoSaiC: Over-Approximation



$\phi$ always in NNF and over($\phi$, $i$) in CNF thanks to Tseitin

$$\text{over}(\phi, n) = \text{over}'(\phi, 0, n)$$
$$\text{over}'(p_k, i, n) = p_{k,i} \quad \text{over}'(\neg p_k, i, n) = \neg p_{k,i}$$
$$\text{over}'(\Box\phi, i, n) = \bigwedge_{j=0}^{n} (r_{i,j} \rightarrow \text{over}'(\phi, j, n))$$
$$\text{over}'(\Diamond\phi, i, n) = \bigvee_{j=0}^{n} (r_{i,j} \wedge \text{over}'(\phi, j, n))$$

- $p_{k,i}$ means $p_k$ is true in the world $w_i$
- $r_{i,j}$ means that there is a relation between worlds $w_i$ and $w_j$

## RECAR Assumptions: Reminder

✓1 Function ‘check’ is sound, complete and terminates

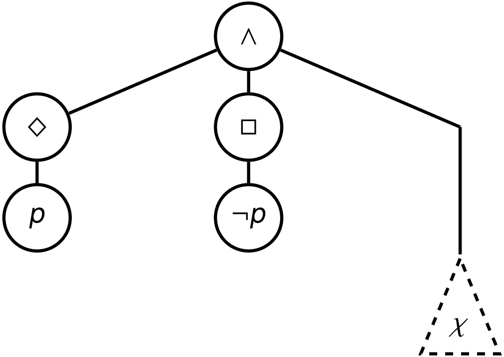
✓2 $isSAT(\hat{\phi})$ implies $isSAT(\text{refine}(\hat{\phi}))$

✓3 $\exists . n \in \mathbb{N}$ s.t. $\text{refine}^n(\hat{\phi}) \equiv^?_{\text{sat}} \phi$

?4 $\text{isUNSAT}(\check{\phi})$ implies $\text{isUNSAT}(\phi)$

?5 $\exists n \in \mathbb{N}$ s.t. $RC(under^n(\phi), under^{n+1}(\phi))$ is false

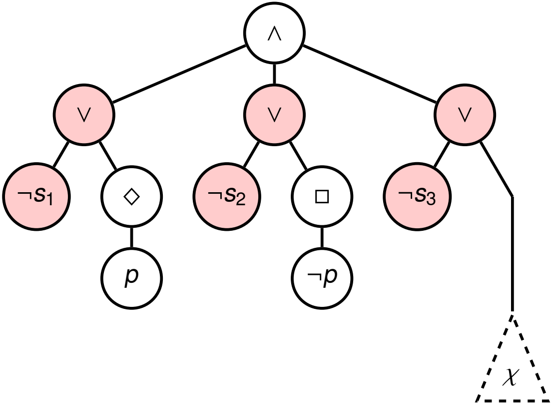
Let's take an example, with $\chi$ huge but satisfiable...




Worst case for CEGAR using our 'over' function

# MoSaiC: Under-Approximation



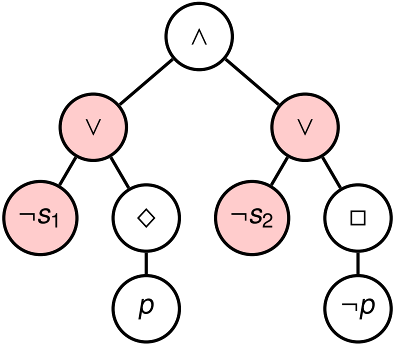



Modern SAT solvers returns 'the reason' why a formula with $n$ worlds is unsatisfiable ($core = \{s_1, s_2\}$)

# MoSaiC: Under-Approximation



We want to cut what is not part of the 'unsatisfiability' ($s_i \notin core$)




We just create $\check{\phi}$ smaller than $\phi$ and easier to solve.
The function *RC* from RECAR just says here: did we cut something ?

# MoSaiC: Under-Approximation



$$\text{under}(p, \mathit{core}) = p$$
$$\text{under}(\neg p, \mathit{core}) = \neg p$$
$$\text{under}(\Box\phi, \mathit{core}) = \Box(\text{under}(\phi, \mathit{core}))$$
$$\text{under}(\Diamond\phi, \mathit{core}) = \Diamond(\text{under}(\phi, \mathit{core}))$$
$$\text{under}((\phi \land \psi), \mathit{core}) = \text{under}(\phi, \mathit{core}) \land \text{under}(\psi, \mathit{core})$$
$$\text{under}((\psi \lor \chi), \mathit{core}) = \begin{cases} \text{under}(\chi, \mathit{core}) & \text{if } \psi = \neg s_i, s_i \in \mathit{core} \\ \top & \text{if } \psi = \neg s_i, s_i \notin \mathit{core} \\ (\text{under}(\psi, \mathit{core}) & \\ \lor\, \text{under}(\chi, \mathit{core})) & \text{otherwise} \end{cases}$$

- Unsatisfiable-cores: To create our under-approximations

### RECAR Assumptions: Reminder

✓1 Function 'check' is sound, complete and terminates

✓2 $isSAT(\hat{\phi})$ implies $isSAT(\text{refine}(\hat{\phi}))$

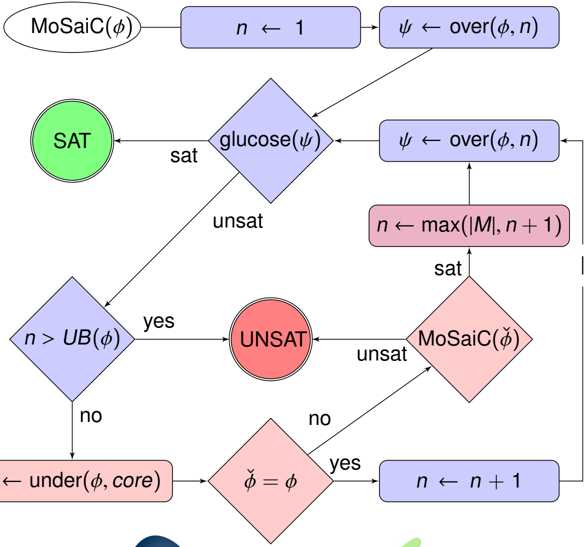
✓3 $\exists . n \in \mathbb{N}$ s.t. $\text{refine}^n(\hat{\phi}) \equiv^?_{\text{sat}} \phi$

✓4 $\text{isUNSAT}(\check{\phi})$ implies $\text{isUNSAT}(\phi)$

✓5 $\exists n \in \mathbb{N}$ s.t. $RC(under^n(\phi), under^{n+1}(\phi))$ is false

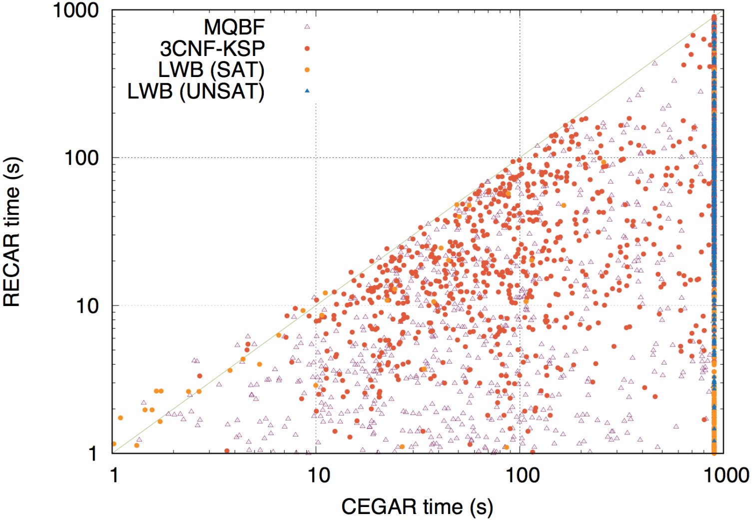
# MoSaiC: RECAR for Modal Logic K





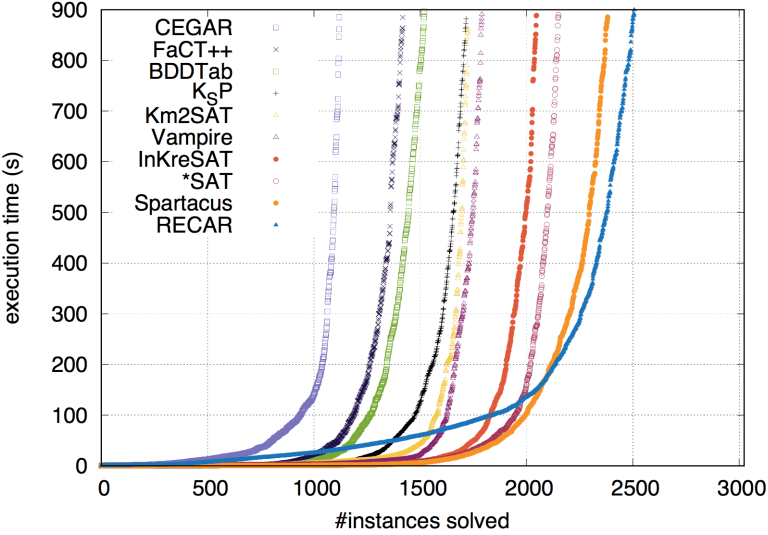
# MoSaiC: RECAR for Modal Logic K





# MoSaiC: RECAR for Modal Logic K



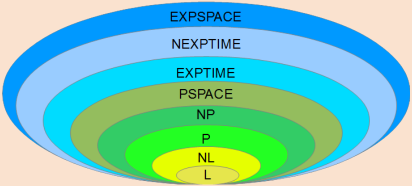


# Conclusion



## Abstractions according to complexity

- **PSPACE**: RECAR
- **NP**: CEGAR (over/under)




## What is next ?

- RECAR for QBF (**PSPACE**)?
- RECAR for other modal logic?

# Perspective: Other modal logics



## Sum-up of complexities in modal logics

| NP |
|---|
| K5 |
| K45 |
| KB45 |
| KD5 |
| KD45 |
| KT5 |

| PSPACE |
|---|
| K |
| KT |
| KT4 |
| KB |
| KD4 |
| KD |
| K4 |
| KDB |
| KBT |

# R.E.C.A.R

## Recursive Explore and Check Abstraction Refinement

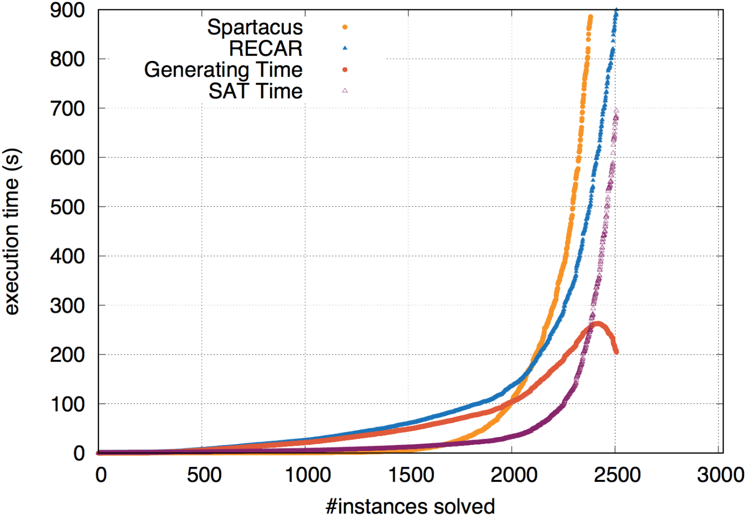
Jean-Marie Lagniez, Daniel Le Berre,
Tiago de Lima and Valentin Montmirail

CRIL-CNRS UMR 8188, F62300 Lens, France

Séminaire CRIL - Lens - September 14th 2017

# Explication on Cactus-Plot





# Bibliography I




- Gilles Audemard, Jean-Marie Lagniez, and Laurent Simon.
  Improving glucose for incremental SAT solving with assumptions: Application to MUS extraction.
  In *Proc. of SAT'13*, pages 309–317, 2013.
- Gilles Audemard and Laurent Simon.
  Predicting learnt clauses quality in modern SAT solvers.
  In *Proc. of IJCAI'09*, pages 399–404, 2009.
- Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith.
  Counterexample-guided abstraction refinement for symbolic model checking.
  *Journal of the ACM*, 50(5):752–794, 2003.

# Bibliography II




Niklas Eén and Niklas Sörensson.
An extensible sat-solver.
In *Proc. of SAT'03*, pages 502–518, 2003.

Niklas Eén and Niklas Sörensson.
Temporal induction by incremental SAT solving.
*Electr. Notes Theor. Comput. Sci.*, 89(4):543–560, 2003.

Allen Van Gelder.
Extracting (easily) checkable proofs from a satisfiability solver that employs both preorder and postorder resolution.
In *International Symposium on Artificial Intelligence and Mathematics*, 2002.

Joseph Y. Halpern.
The Effect of Bounding the Number of Primitive Propositions and the Depth of Nesting on the Complexity of Modal Logic.
*Artificial Intelligence*, 75(2):361–372, 1995.

Saul Kripke.
A completeness theorem in modal logic.
*J. Symb. Log.*, 24(1):1–14, 1959.

Richard E. Ladner.
The Computational Complexity of Provability in Systems of Modal Propositional Logic.
*SIAM J. Comput.*, 6(3):467–480, 1977.

- Jean-Marie Lagniez, Daniel Le Berre, Tiago de Lima, and Valentin Montmirail.
  A Recursive Shortcut for CEGAR: Application To The Modal Logic K Satisfiability Problem.
  In *Proc. of IJCAI'17*, 2017.
- Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik.
  Chaff: Engineering an efficient SAT solver.
  In *Proc. of DAC'01*, pages 530–535, 2001.
- Steven David Prestwich.
  SAT problems with chains of dependent variables.
  *Discrete Applied Mathematics*, 130(2):329–350, 2003.

# Bibliography V




- Takehide Soh, Daniel Le Berre, Stéphanie Roussel, Mutsunori Banbara, and Naoyuki Tamura.
  Incremental sat-based method with native boolean cardinality handling for the hamiltonian cycle problem.
  In *Proc of JELIA'14*, pages 684–693, 2014.
- João P. Marques Silva and Karem A. Sakallah.
  GRASP: A search algorithm for propositional satisfiability.
  *IEEE Transactions on Computers*, 48(5):506–521, 1999.