October 3, 2023
Shonan Village Center, Japan

IPASIR-UP: User Propagators for CDCL

Based on joint work with:
Aina Niemetz, Mathias Preiner, Markus Kirchweger,
Stefan Szeider, Armin Biere

Katalin Fazekas
TU Wien, Vienna, Austria

Outline

IPASIR-UP: User Propagators for CDCL

Inprocessing SAT Solvers

Open Problems with Proofs & Solutions

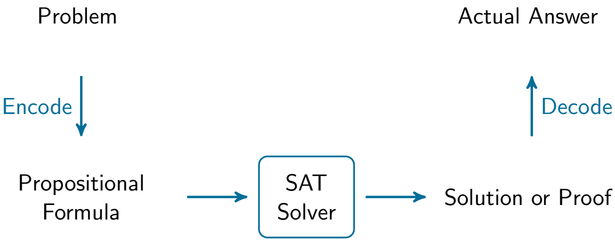
Outline

IPASIR-UP: User Propagators for CDCL

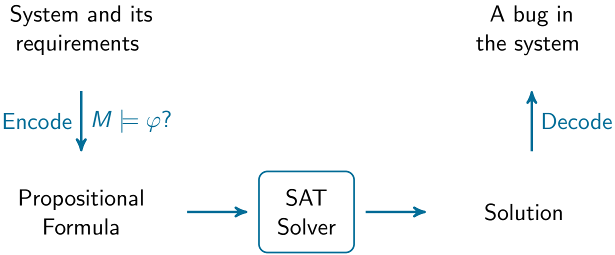
Inprocessing SAT Solvers

Open Problems with Proofs & Solutions

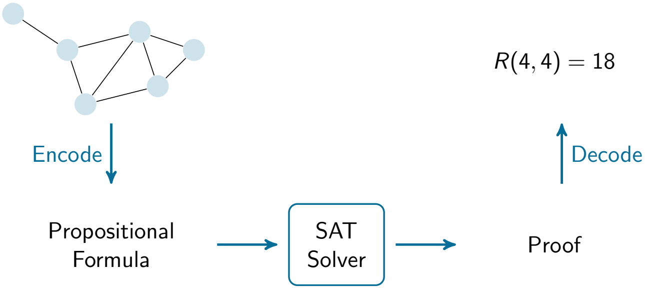
Usual Use of SAT Solvers



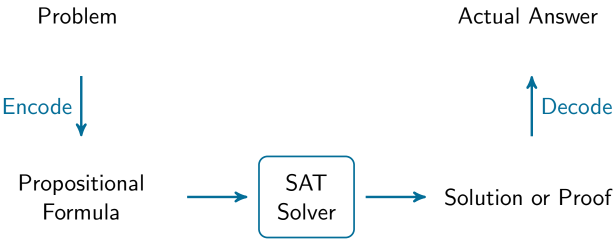
Usual Use of SAT Solvers



Usual Use of SAT Solvers

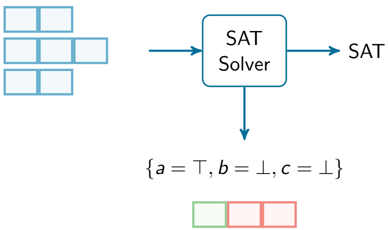


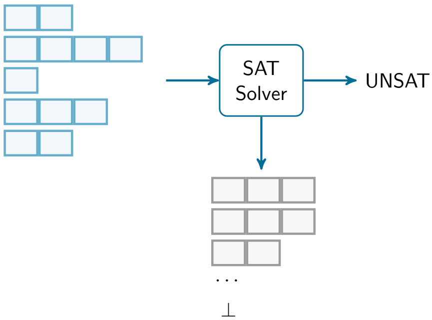
Usual Use of SAT Solvers



+ Efficient Tools & Verifiable results

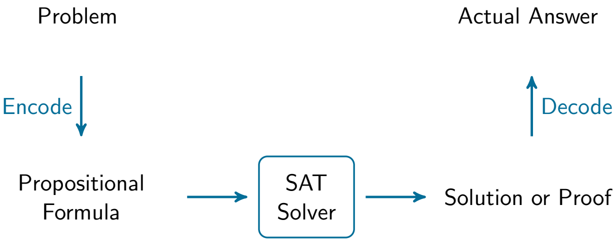
Verifiable Results – Proofs & Solutions of SAT Solvers





- Solution $\sim$ Trail of the solver when all variables are assigned
- Proof $\sim$ Record of all added (and deleted) clauses
- Both are built while the solver decides satisfiability

Usual Use of SAT Solvers

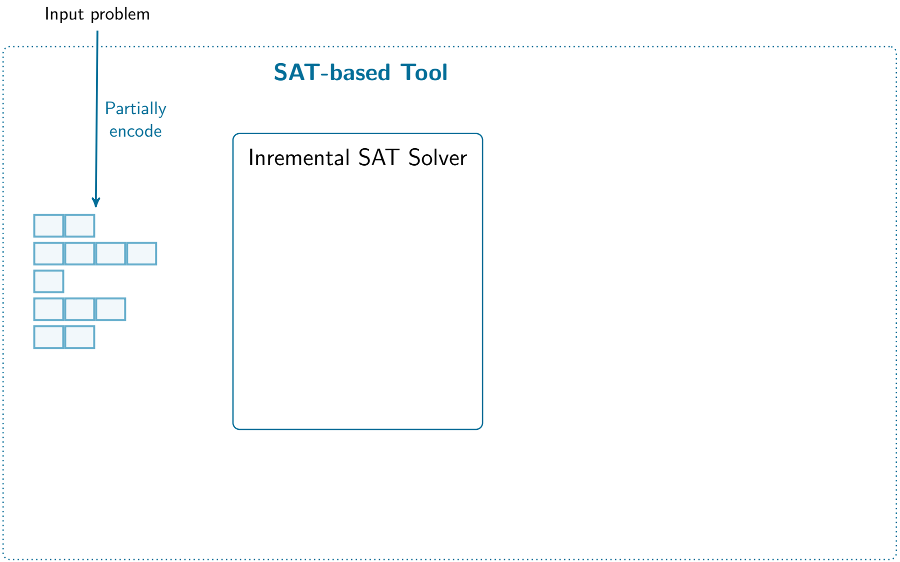


+ Efficient Tools & Verifiable results

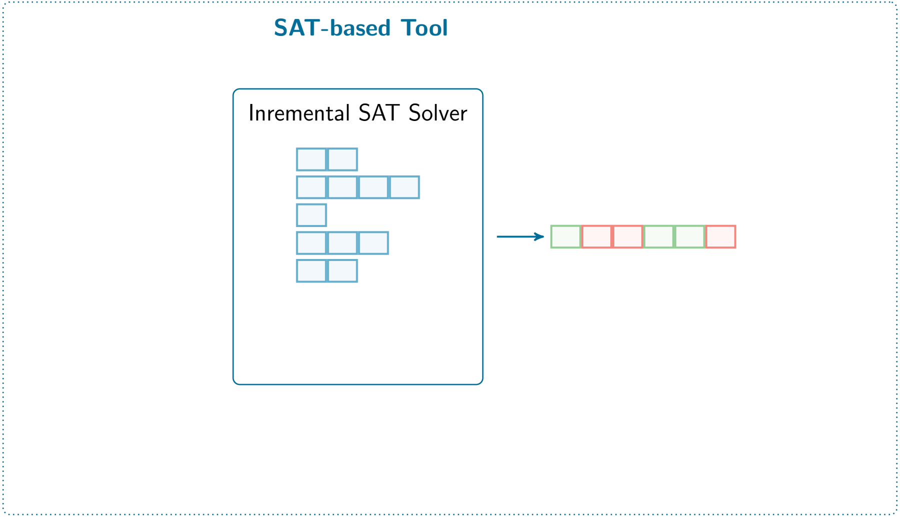
But . . .

- Complete encoding can be extremely large (or impossible)
- Not everything is relevant to find a refutation
- Not everything is best solved as SAT

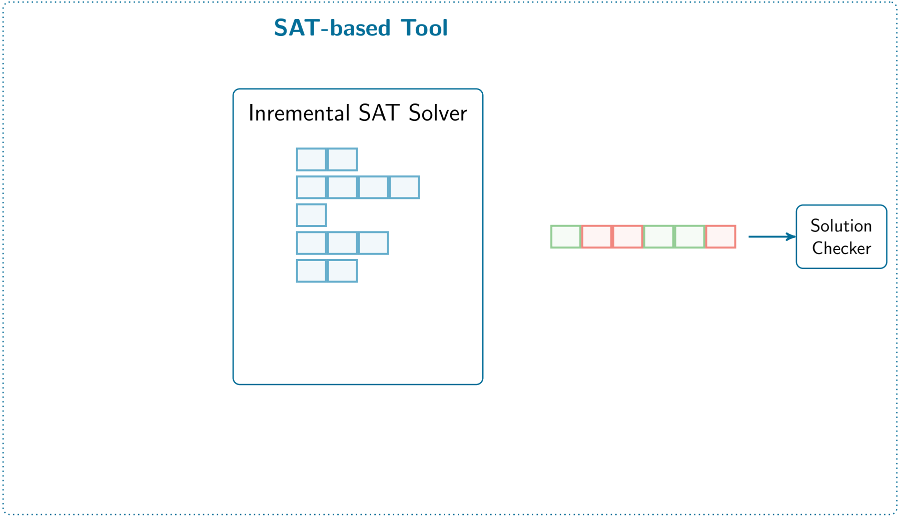
Usual Use of Incremental SAT Solvers



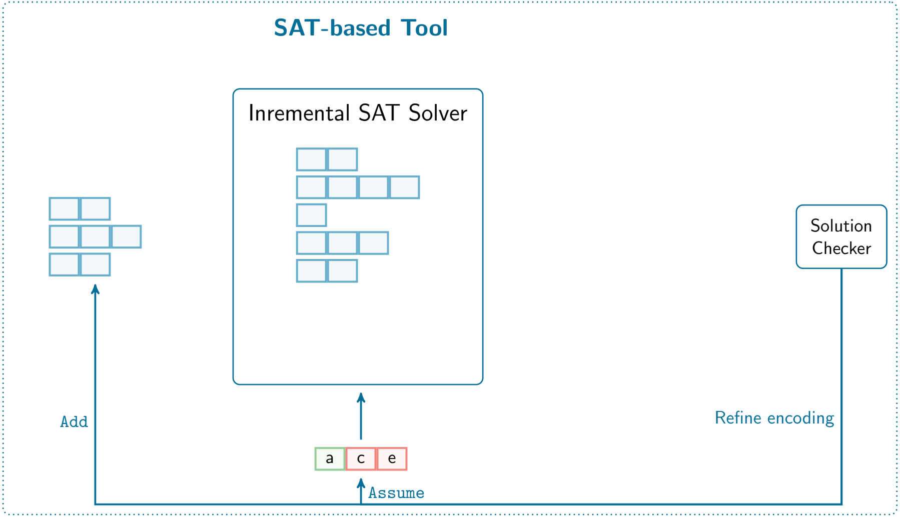
Usual Use of Incremental SAT Solvers



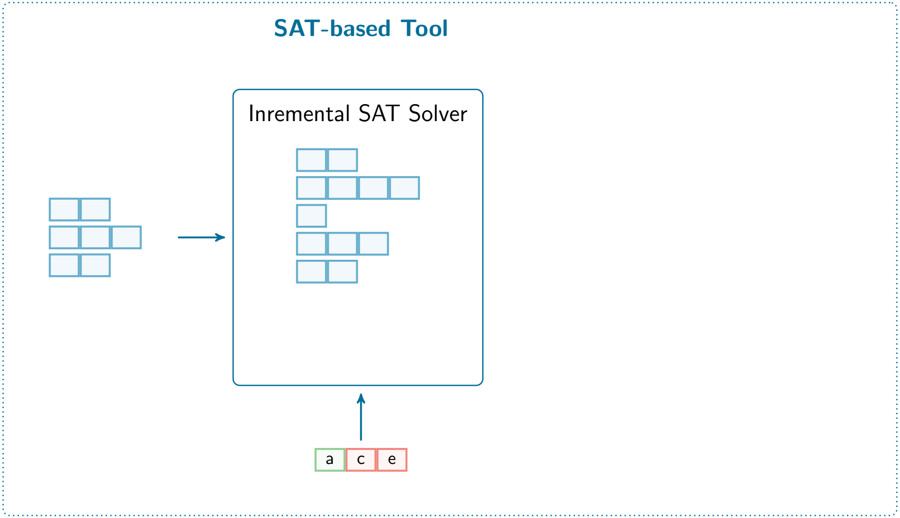
Usual Use of Incremental SAT Solvers



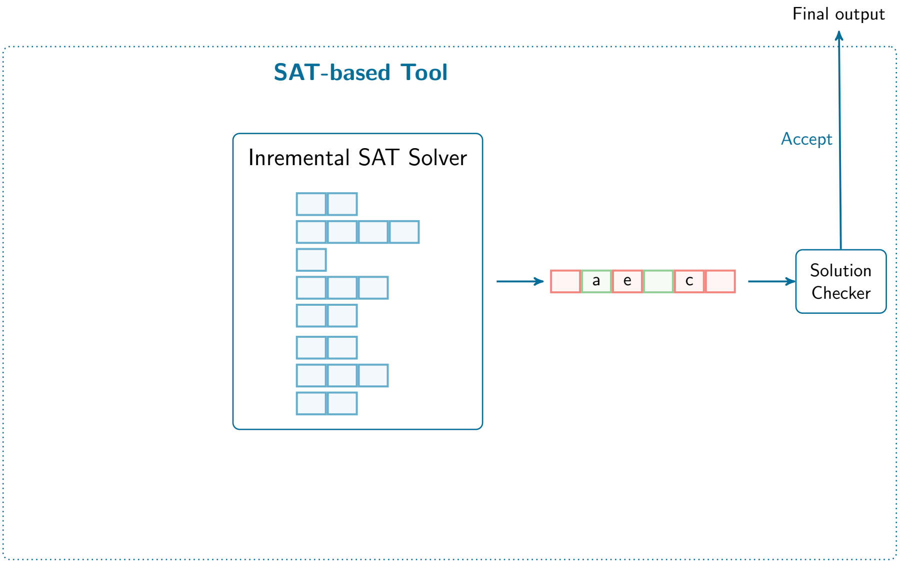
Usual Use of Incremental SAT Solvers



Usual Use of Incremental SAT Solvers



Usual Use of Incremental SAT Solvers



Usual Use of Incremental SAT Solvers

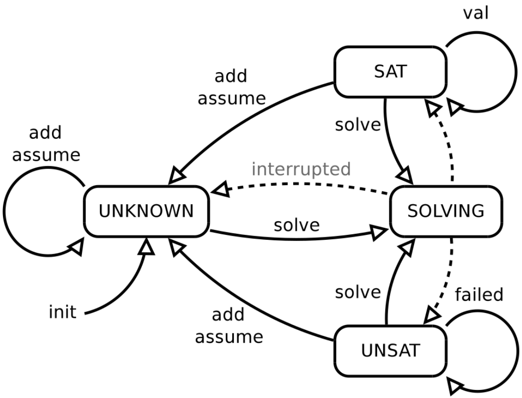
- Bounded Model Checking, Planning, MaxSAT, lazy SMT, . . .

- Reuse exact same solver instance

+ Smaller initial encoding
+ Can reuse previous reasoning steps instead of repeating them
  - Keep learned clauses
  - Keep gathered information (e.g. phases, scores)
  - Keep applied formula simplifications
+ Assumptions provide some influence over the search
+ IPASIR interface makes SAT Solvers interchangeable

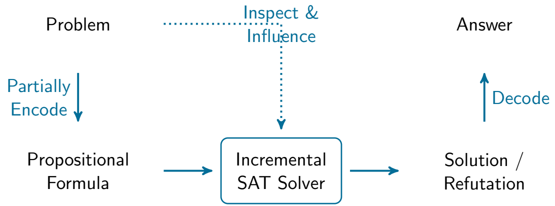
IPASIR – Interface of Incremental SAT Solving

- Standardized interface, used also at annual competition [BalyoBiereIserSinz-JAI'16]
- IPASIR: *"Re-entrant Incremental Satisfiability Application Program Interface"*
- Supports interactions *between* solve calls



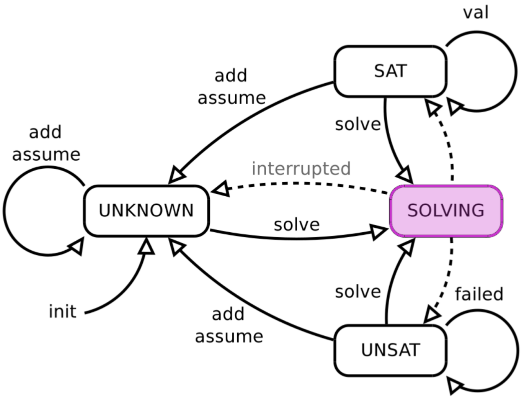
Usual Use of User Propagators

- Incremental SAT is not always enough: CDCL(CAS), Combinatorial problems, SMT, maxSAT, . . .
  - Interaction is possible only once the solving is finished



- Requires workarounds and modifications in the SAT solver
  - Non-replaceable SAT solver → missed advancements
  - New application needs new modifications
  - Error prone, potential drop in performance

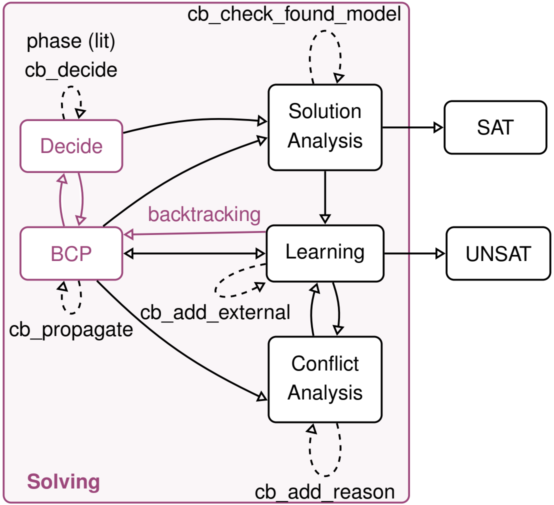
IPASIR-UP: Standardize Propagator Interface for CDCL



- Support interactions **during** the solve () calls

IPASIR-UP: IPASIR with User Propagators

- Inspect search
  - Notify all changes to the trail
- Influence search
  1. Add propagations (without adding reason clauses)
  2. Dictate decisions & phases
  3. Add new clauses (anytime!)
  4. Overrule found solutions
  5. Explain relevant propagations



Example C++ implementation

```
class ExternalPropagator {
public:
  virtual ~ExternalPropagator () { }

  virtual void notify_assignment (int lit, bool is_fixed) {}
  virtual void notify_new_decision_level () {}
  virtual void notify_backtrack (size_t new_level) {}

  virtual int cb_decide () { return 0; }
  virtual int cb_propagate () { return 0; }
  virtual int cb_add_reason_clause_lit (int propagated_lit) {
      return 0;
  }
  virtual bool cb_check_found_model (const std::vector<int> & model) {
      return true;
  }

  virtual bool cb_has_external_clause () { return false; }
  virtual int cb_add_external_clause_lit () { return 0; }
};
```

Related Work

- clingo [GebserKaminskiKaufmannOstrowskiSchaubWanko'16]
  - A state-of-the-art ASP solver
  - Supports *theory propagators*
- Interactive SAT
  - Programmatic SAT: Lynx [GaneshO'DonnellSoosDevadasRinardSolar-Lezama'12]
  - IntelSAT [Nadel'22]
- CP solvers [GentMiguelMoore'10]
  - Lazy explanation, lazy clause generation
- SAT and Theory solvers of SMT solvers [NieuwenhuisOliverasTinelli'06]
  - SAT worker interface [CimattiGriggioSchaafsmaSebastiani'13]
  - User propagators of z3 [BjørnerEisenhoferKovács'22]

IPASIR-UP Experiments

- Extended CaDiCaL with IPASIR-UP
  - A state-of-the-art incremental, inprocessing, proof producing SAT solver
  - ~800 lines of additional code (plus another ~700 for testing)

IPASIR-UP Experiments

- Extended CaDiCaL with IPASIR-UP
  - A state-of-the-art incremental, inprocessing, proof producing SAT solver
  - ~800 lines of additional code (plus another ~700 for testing)
- Evaluated on two representative use cases
  - Combinatorial problem solving: SAT modulo Symmetries (SMS)
    - See talk of Stefan Szeider
  - Satisfiability modulo Theories: cvc5
    - See talk of Mathias Preiner

IPASIR-UP Experiments

- Extended CaDiCaL with IPASIR-UP
  - A state-of-the-art incremental, inprocessing, proof producing SAT solver
  - ~800 lines of additional code (plus another ~700 for testing)
- Evaluated on two representative use cases
  - Combinatorial problem solving: SAT modulo Symmetries (SMS)
    - See talk of Stefan Szeider
  - Satisfiability modulo Theories: cvc5
    - See talk of Mathias Preiner
- Generic interface to inspect and influence CDCL search
  - Simple & Flexible → relatively easy to implement
  - Sufficient to simplify several use cases

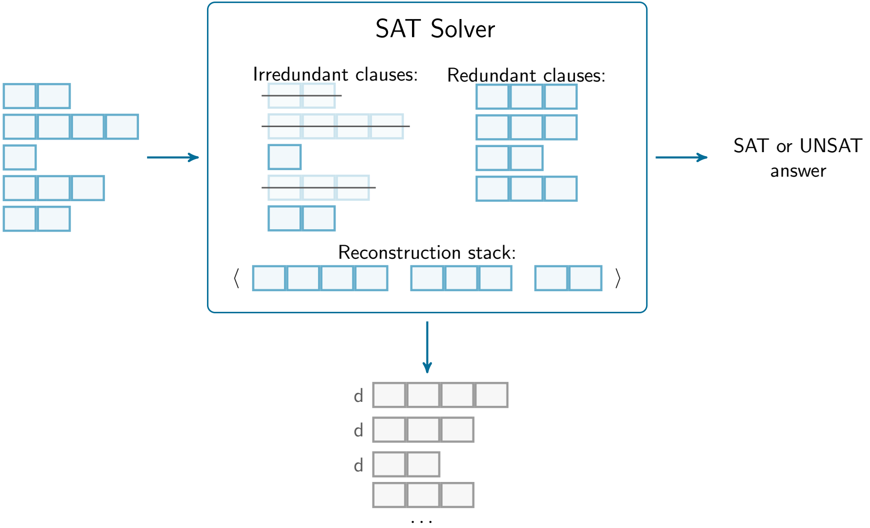
Outline

IPASIR-UP: User Propagators for CDCL

Inprocessing SAT Solvers

Open Problems with Proofs & Solutions

Inprocessing SAT Solvers [JärvisaloHeuleBiere-IJCAR'12]



Inprocessing Rules [JärvisaloHeuleBiere-IJCAR'12]

- Satisfiability preserving clause addition or removal
- Inprocessing as sequence of abstract states: $\varphi\ [\,\rho\,]\ \sigma$

  φ: Irredundant clauses ρ: Redundant clauses σ: Reconstruction stack

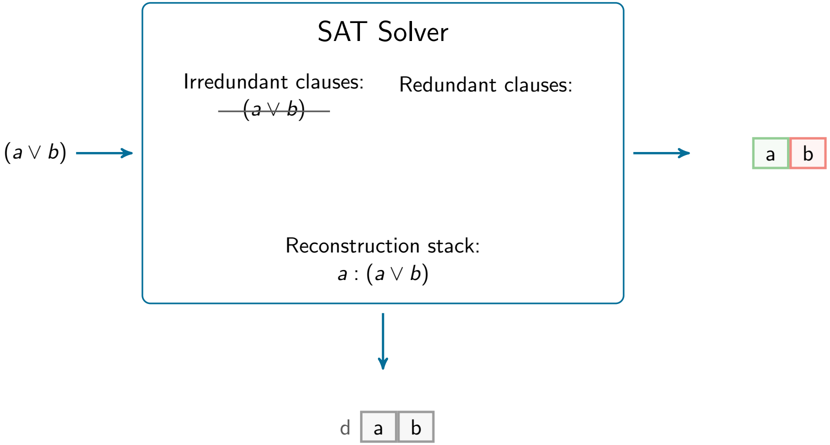
$$\frac{\varphi\,[\,\rho\,]\,\sigma}{\varphi\,[\,\rho \wedge C\,]\,\sigma}\ \boxed{\sharp} \qquad \frac{\varphi\,[\,\rho \wedge C\,]\,\sigma}{\varphi\,[\,\rho\,]\,\sigma} \qquad \frac{\varphi\,[\,\rho \wedge C\,]\,\sigma}{\varphi \wedge C\,[\,\rho\,]\,\sigma} \qquad \frac{\varphi \wedge C\,[\,\rho\,]\,\sigma}{\varphi\,[\,\rho \wedge C\,]\,\sigma \cdot (l : C)}\ \boxed{\flat}$$

LEARN FORGET STRENGTHEN WEAKEN

where $\boxed{\sharp}$ is $\varphi \wedge \rho \equiv_{sat} \varphi \wedge \rho \wedge C$ and $\boxed{\flat}$ is $\varphi \wedge C \equiv_{sat}^{\ell} \varphi$

Formulas φ and $\varphi \wedge \rho$ are both satisfiability equivalent to the original input formula.

Solution Reconstruction [JärvisaloHeuleBiere-IJCAR'12]



- Inprocessing is satisfiability but not model preserving
- Solution reconstruction is needed to get model of original formula

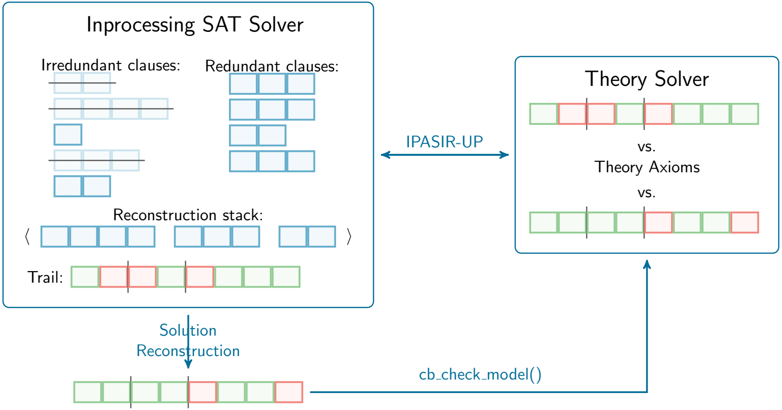
Outline

IPASIR-UP: User Propagators for CDCL

Inprocessing SAT Solvers

Open Problems with Proofs & Solutions

Problem 1: IPASIR-UP & Solution Reconstruction

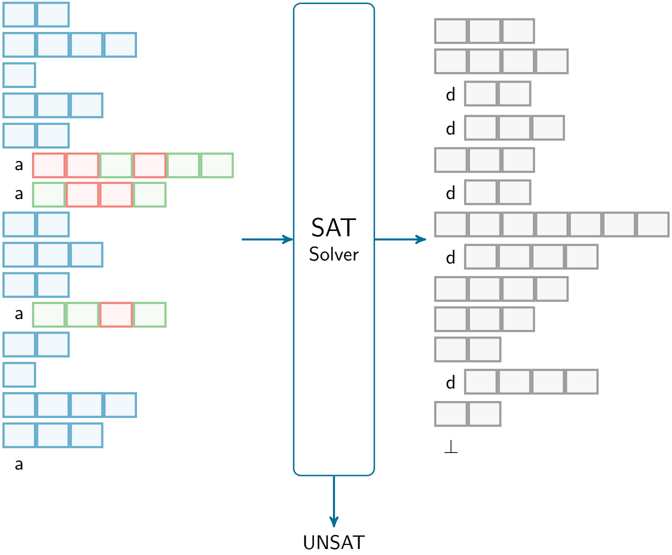


- Theory solver works to always keep the trail of SAT solver theory consistent
- In the final solution some values are flipped → theory consistency is unknown
- Non-incremental theory queries

Problem 1 – Solution Ideas

1. Forbid inprocessing of theory literals (freezing)
   - + Very simple implementation (current solution)
   - – Only very limited inprocessing is allowed
2. Forbid notifying assignments of witness literals
   - + No flipped assignments in solution reconstruction
   - – Many theory literals gets assigned only in the complete model → lazy
3. Apply solution reconstruction on the partial solution
   - – Not correct (in theory) [FleuryLammich-CADE'23]
   - – Does not guarantee that last solution reconstruction will not flip values
4. Allow only "theory-consistent" elimination steps
   - □ Theory aware inprocessing → SMT inprocessing [BjørnerFazekas-CADE'23]
   - + Solution reconstruction maintains theory consistency
   - – How to do that?

Problem 2: Incremental Queries & Their Proofs

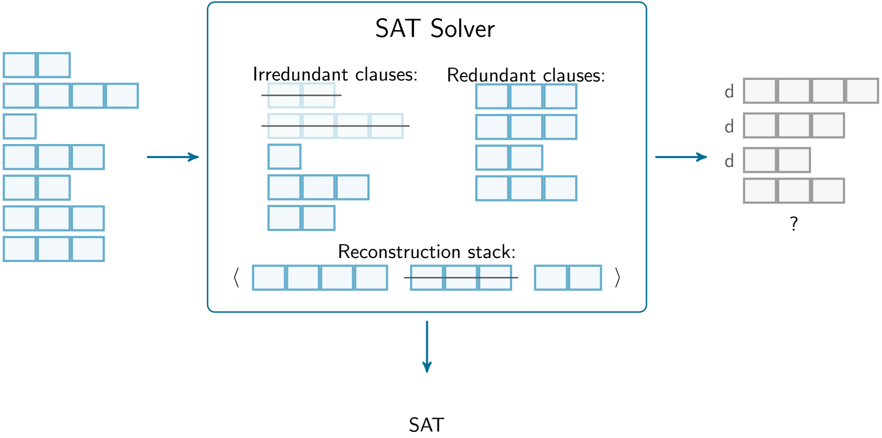


Problem 2: Solution Ideas (Format)

- Define incremental DIMACS
  - Standardize iCNF
  - Use as input for the proof checker

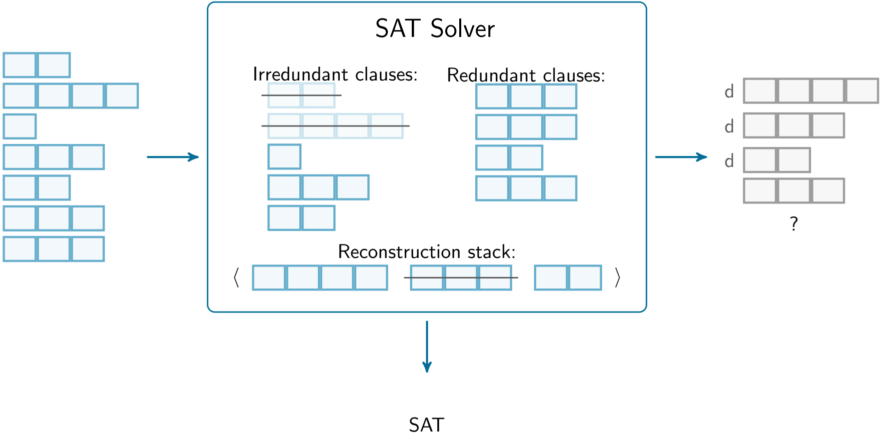
- Introduce proof conclusion explicitly: For each query, derive either
  - the empty clause or
  - a clause over the failed assumptions

Problem 3: Incremental Iprocessing & Proof Production



- Restore clause from reconstruction stack

Problem 3: Incremental Iprocessing & Proof Production



- Restore clause from reconstruction stack
- What if it gets deleted again in a later query?

Incremental Inprocessing Rules [FazekasBiereScholl-SAT'19]

$$\frac{\varphi\ [\,\rho\,]\ \sigma}{\varphi\ [\,\rho \land C\,]\ \sigma}\ \boxed{\sharp}$$

Learn⁻

$$\frac{\varphi\ [\,\rho \land C\,]\ \sigma}{\varphi\ [\,\rho\,]\ \sigma}$$

Forget

$$\frac{\varphi\ [\,\rho \land C\,]\ \sigma}{\varphi \land C\ [\,\rho\,]\ \sigma}$$

Strengthen

$$\frac{\varphi\ [\,\rho\,]\ \sigma}{\varphi \land \Delta\ [\,\rho\,]\ \sigma}\ \boxed{\mathcal{I}}$$

AddClauses

$$\frac{\varphi \land C\ [\,\rho\,]\ \sigma}{\varphi\ [\,\rho\,]\ \sigma \cdot (\omega : C)}\ \boxed{\flat}$$

Weaken⁺

$$\frac{\varphi \land C\ [\,\rho\,]\ \sigma}{\varphi\ [\,\rho\,]\ \sigma}\ \boxed{\emptyset}$$

Drop

$$\frac{\varphi\ [\,\rho\,]\ \sigma \cdot (\omega : C) \cdot \sigma'}{\varphi \land C\ [\,\rho\,]\ \sigma \cdot \sigma'}\ \boxed{\partial}$$

Restore

where $\boxed{\sharp}$ is $\varphi \land \rho \models C$, $\boxed{\flat}$ is $\varphi \land C \equiv^{\omega}_{sat} \varphi$, $\boxed{\emptyset}$ is $\varphi \models C$,

$\boxed{\partial}$ is C is clean w.r.t. σ' and $\boxed{\mathcal{I}}$ is that each clause in Δ is clean w.r.t. σ

Problem 3: Possible Solutions

- Undo corresponding delete step [Kiesl-ReiterWhalen-FMCAD'23]
  - □ What if restore happened only in a very late query?
  - – Proof trimming is reduced
- Reintroduce with original ID (LRAT)
  - + Can be kept deleted until restore
  - + Easy to verify?
  - – More information need to be stored on reconstruction stack
- Extend proof format to support incremental calculus
  - + Checkable deletion steps → proofs of satisfiable problems
  - + Clear report on what happens in the solver
  - – Calculus might need some optimizations to keep proofs shorter
  - – How to prove cleanness in rule Restore?