IPASIR-UP: User Propagators for CDCL

A CaDiCaL Integration into CDCL($\mathcal{T}$)

Katalin Fazekas[1], **Aina Niemetz**[2], Mathias Preiner[2], Markus Kirchweger[1], Stefan Szeider[1], Armin Biere[3]

Centaur Annual Meeting, August 17, 2023

[1] TU Wien

[2] Stanford University

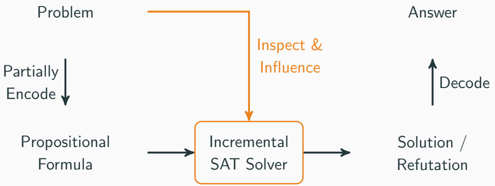
[3] University of Freiburg

Stanford | Center for Automated Reasoning

IPASIR-UP in a Nutshell

IPASIR-UP = **IPASIR** + **U**ser **P**ropagators

- a SAT solver **interface** for
- **interactive** incremental SAT solving



- **Our focus here:** Integration as CDCL($\mathcal{T}$) SAT solver

The CDCL($\mathcal{T}$) Lazy SMT Framework

- **propositional abstraction** of the input formula
- **iteratively refined** until abstraction is $\mathcal{T}$-consistent or unsat
- theory layer **guides** the search of the SAT solver
- **online, tight integration of SAT solver**
  - **theory layer** interacts with SAT solver **during the search**
  - backward communication channel to **notify theory layer** about variable assignments, decisions, backtracks
  - theory layer **derives** conflicts, **propagates** theory literals, **suggests** decisions based on theory-guided heuristics

CDCL($\mathcal{T}$) SAT solver: Current State-of-the-Art

- **no standardized SAT solver interface**
  for **interactive** incremental SAT solving

- **solver-specific** workarounds and modifications to the SAT solver
- **error prone**, high potential for unintentional **performance** hits
- **difficult to replace**
- **missed** opportunities to take advantage of **improvements** in SAT

IPASIR-UP: A New Interface for Interactive CDCL

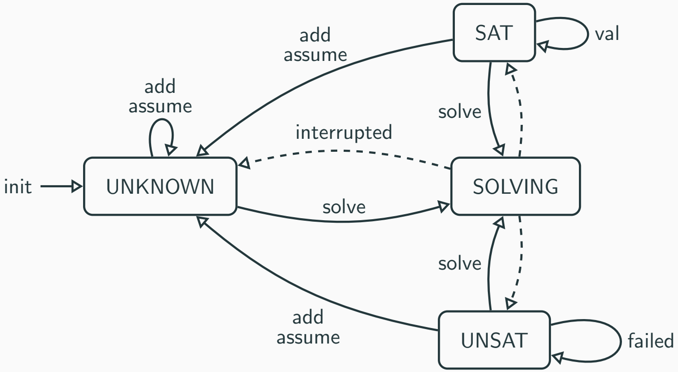
- interface to support **standardized interactions** with the SAT solver **during solving**
- extends the standardized IPASIR interface

− Needs to be implemented in SAT solvers (only once)

+ Easy to use
+ Solver independent application development
+ No more black-box SAT solving → new potentials
+ Standardized and clean interactions

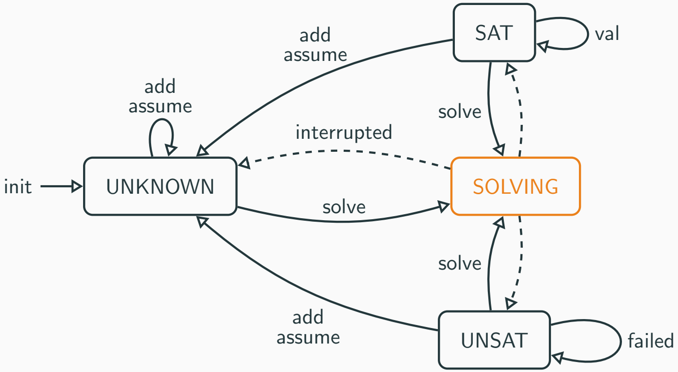
IPASIR Model of Incremental SAT Solvers

- **Re-entrant Incremental Satisfiability API (IPASIR)**
- Supports interactions **between** solve calls



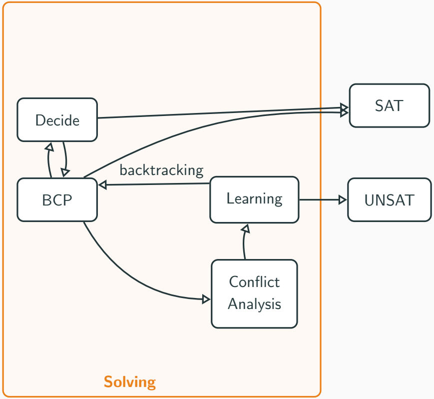
IPASIR Model of Incremental SAT Solvers

- **Re-entrant Incremental Satisfiability API (IPASIR)**
- Supports interactions **between** solve calls



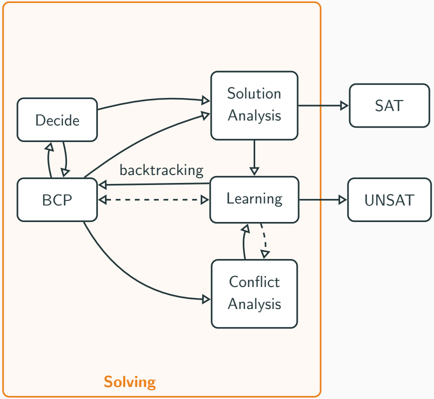
IPASIR Model of Incremental SAT Solvers

» Supports interactions **between** solve calls



IPASIR-UP: IPASIR with User Propagators

» Supports interactions **during** solve calls

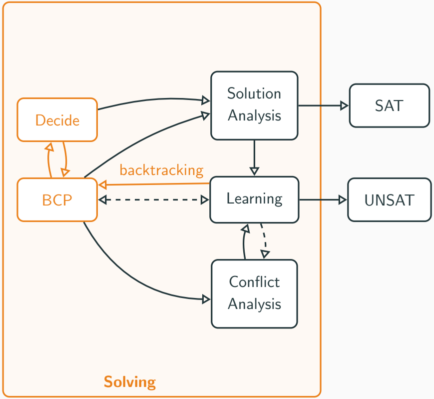


IPASIR-UP: IPASIR with User Propagators

» Supports interactions **during** solve calls

» **Inspect** search

- **notify** (all trail changes)
  - assignment, decision
  - backtrack



IPASIR-UP: IPASIR with User Propagators

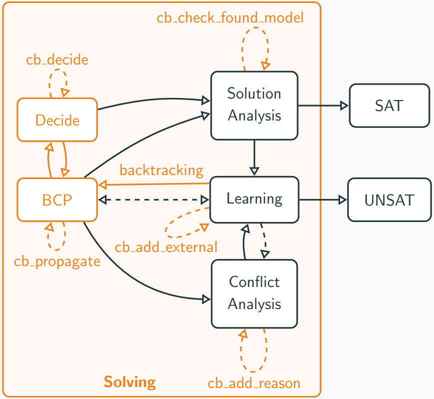
» Supports interactions **during** solve calls

» **Inspect** search

- **notify** (all trail changes)
  - assignment, decision
  - backtrack

» **Influence** search

1. overrule found **solutions**
2. **decide** decisions and phases
3. add **propagations** (without adding clauses)
4. add **new clauses** anytime
5. **explain** propagations



IPASIR-UP in cvc5

- **state-of-the-art** SMT solver
- based on **CDCL($\mathcal{T}$)** framework
- integrates **highly customized version of MiniSat**
  - supports production of resolution proofs
  - push/pop of assertion levels
  - custom theory-guided decision heuristics
- **difficult** to replace

With CaDiCaL via IPASIR-UP

- $\sim$**700 C++ LOC** for integration via IPASIR-UP
- easily replaced with any SAT solver implementing IPASIR-UP

IPASIR-UP in cvc5

- **Full utilization** of interface
  - notify_assignment
    - construct partial assignment for observed theory literals
  - notify_new_decision_level and notify_backtrack
    - manage incremental solver state
  - cb_propagate
    - theory propagations
  - cb_add_reason_clause_lit
    - theory explanations
  - cb_decide
    - implementation of custom decision heuristics
  - cb_add_external_clause_lit
    - add lemmas and conflict clauses
  - cb_check_found_model
    - check if current assignment is $\mathcal{T}$-satisfiable

Evaluation

- **non-incremental** benchmarks of SMT-LIB 2022
- 300s time limit, 8GB memory limit
- compare against cvc5 1.0.5 with customized MiniSat
- **promising performance** without much tuning or optimizations
- $+1080$ solved instances
- $\sim 2\times$ **faster** in several logics
- **13 of 19** SMT-COMP divisions **improved**
- **solid baseline** for future tuning with IPASIR-UP interface

Evaluation: SMT-COMP Divisions

	cvc5		cvc5-ipasirup	
Division	solved	time [s]	solved	time [s]
Arith (6,865)	**6,303**	173,628	6,299	176,278
BitVec (6,045)	**5,552**	153,899	5,529	161,482
Equality (12,159)	5,320	2,062,804	**5,322**	2,061,758
Equality+LinearArith (53,453)	45,902	2,288,230	**45,906**	2,288,352
Equality+MachineArith (6,071)	983	1,533,646	**987**	1,532,782
Equality+NonLinearArith (21,104)	**13,314**	2,419,535	13,053	2,486,588
FPArith (3,965)	3,145	268,628	**3,155**	266,245
QF_Bitvec (42,472)	**40,321**	984,880	40,320	985,946
QF_Datatypes (8,403)	8,077	110,704	**8,168**	82,878
QF_Equality (8,054)	8,044	9,394	**8,047**	7,169
QF_Equality+Bitvec (16,585)	15,817	307,558	**16,015**	234,369
QF_Equality+LinearArith (3,442)	**3,388**	23,041	3,381	23,465
QF_Equality+NonLinearArith (709)	627	27,428	**629**	27,598
QF_FPArith (76,238)	76,054	94,487	**76,081**	76,700
QF_LinearIntArith (16,387)	11,670	1,575,635	**12,004**	1,512,696
QF_LinearRealArith (2,008)	1,721	130,408	**1,766**	113,919
QF_NonLinearIntArith (25,361)	13,037	4,094,712	**13,682**	3,840,933
QF_NonLinearRealArith (12,134)	11,166	333,933	**11,238**	316,728
QF_Strings (69,908)	**69,357**	203,677	69,296	230,918
Total (391,363)	339,798	16,796,234	**340,878**	16,426,813

Conclusion

- **Generic interface** to **inspect** and **influence** CDCL search
  - Simple & Flexible » **relatively easy to implement**
  - Sufficient to simplify several use cases
- Implemented in a **complex**, **modern** SAT solver
  - Allows inprocessing of non-changing parts
- Evaluated in **representative** use cases (SMS, SMT)
  - Captures the necessary interactions of a very wide range of use cases
  - **promising results**

Future Work

- **SAT:** more inprocessing, external proofs of external clauses
- **cvc5:** DRAT proof integration

Appendix: 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; }
};
```

Appendix: Additional Functions

```
// VALID = UNKNOWN | SATISFIED | UNSATISFIED
//
// require (VALID) -> ensure (VALID)
//
void connect_external_propagator (ExternalPropagator * propagator);

// require (VALID) -> ensure (VALID)
//
void disconnect_external_propagator ();

// require (VALID_OR_SOLVING) /\ CLEAN(var) -> ensure (VALID_OR_SOLVING)
//
void add_observed_var (int var);

// require (VALID) -> ensure (VALID)
//
void remove_observed_var (int var);

// require (VALID_OR_SOLVING) -> ensure (VALID_OR_SOLVING)
//
bool is_decision (int observed_var);

// require (VALID_OR_SOLVING) -> ensure (VALID_OR_SOLVING)
//
void phase (int lit);

// require (VALID_OR_SOLVING) -> ensure (VALID_OR_SOLVING)
//
void unphase (int lit);
```

Appendix: Evaluation: Logics