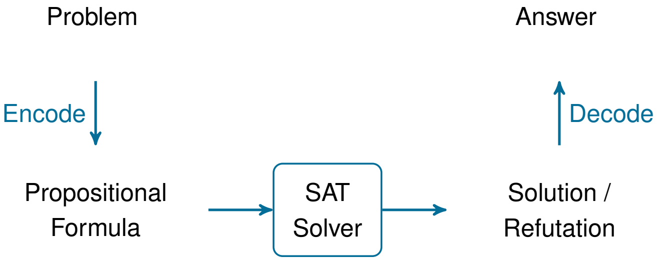
IPASIR-UP: User Propagators for CDCL

SAT Conference, July 7, 2023
Alghero, Italy

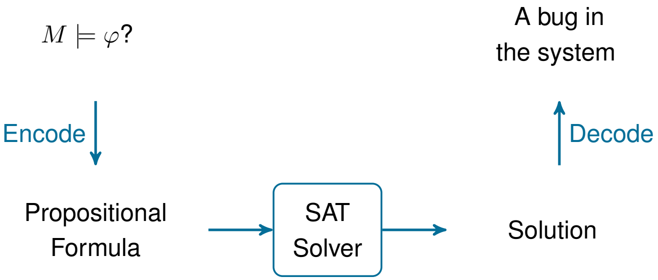
Katalin Fazekas[1], Aina Niemetz[2], Mathias Preiner[2],
Markus Kirchweger[1], Stefan Szeider[1], Armin Biere[3]

[1]TU Wien, Vienna, Austria
[2]Stanford University, Stanford, USA
[3]University of Freiburg, Freiburg, Germany

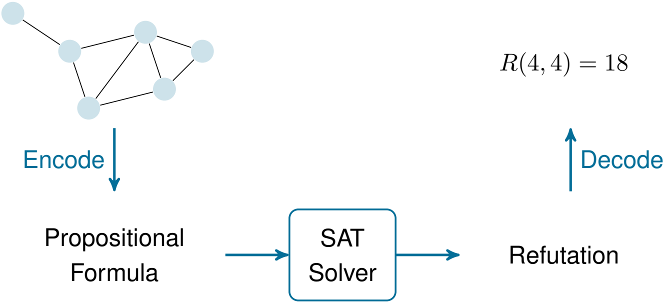
Usual Use of SAT Solvers



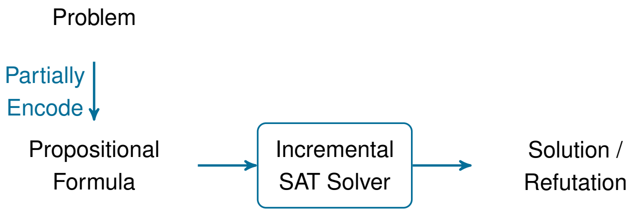
Usual Use of SAT Solvers



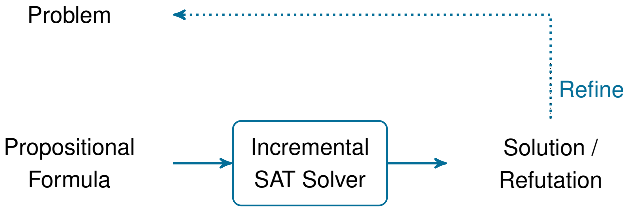
Usual Use of SAT Solvers



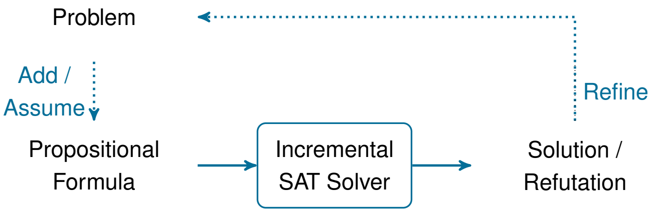
Usual Use of Incremental SAT Solvers



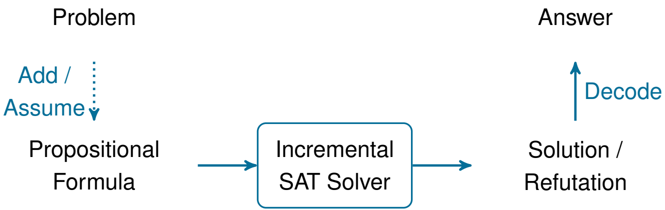
Usual Use of Incremental SAT Solvers



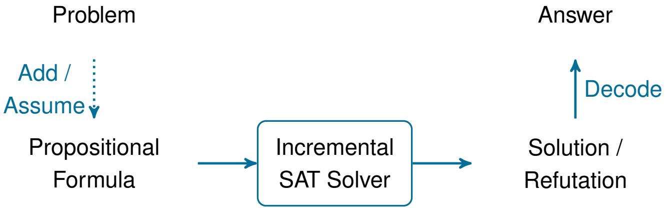
Usual Use of Incremental SAT Solvers



Usual Use of Incremental SAT Solvers

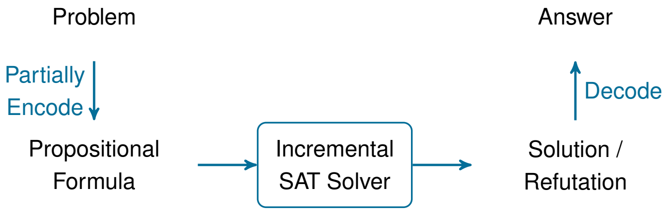


Usual Use of Incremental SAT Solvers



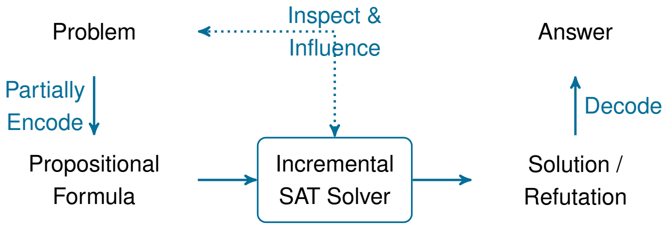
- Model enumeration, model checking, CEGAR, symbolic execution, . . .

Use of Incremental SAT Solvers via User Propagators



- Combinatorial problems, SMT, symmetry breaking, model enumeration, MaxSAT, . . .

Use of Incremental SAT Solvers via User Propagators

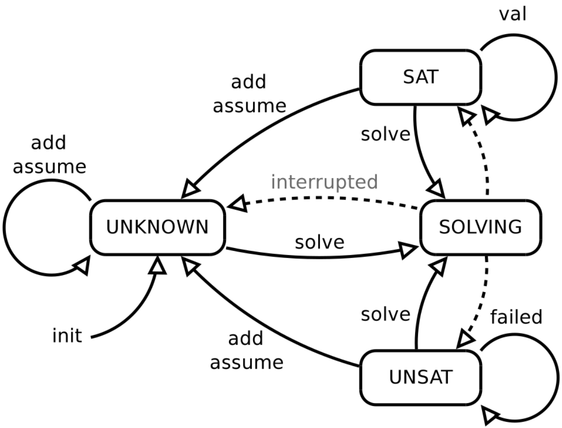


- Combinatorial problems, SMT, symmetry breaking, model enumeration, MaxSAT, . . .

IPASIR-UP: A New Interface for Interactive CDCL

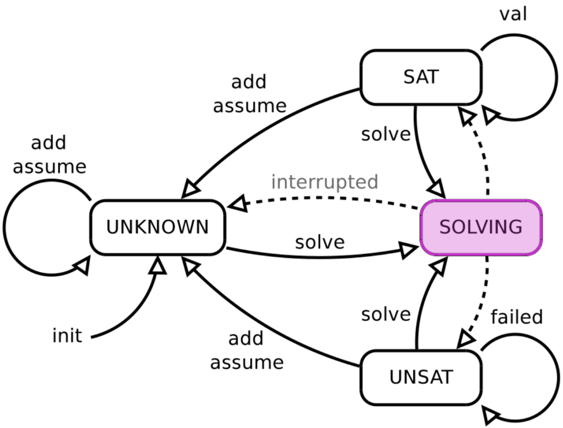
- Interface to support standardized interactions with the solver during solving
  - – Needs to be implemented in SAT solvers (only once)
  - + Easy to implement and use
  - + Solver independent application development
  - + No more black-box SAT solving $\rightarrow$ new potentials
  - + Standardized and clean interactions

IPASIR Model of Incremental SAT Solvers [BalyoBiereIserSinz'16]



- IPASIR: *"Re-entrant Incremental Satisfiability Application Program Interface"*
- Supports interactions **between** solve calls

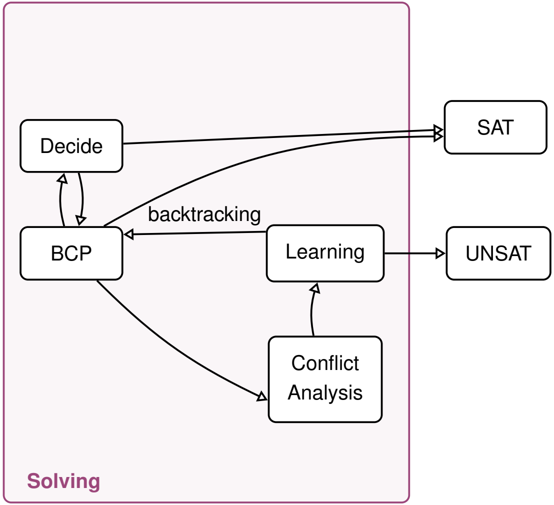
IPASIR Model of Incremental SAT Solvers [BalyoBiereIserSinz'16]



- IPASIR: *"Re-entrant Incremental Satisfiability Application Program Interface"*
- Supports interactions **between** solve calls

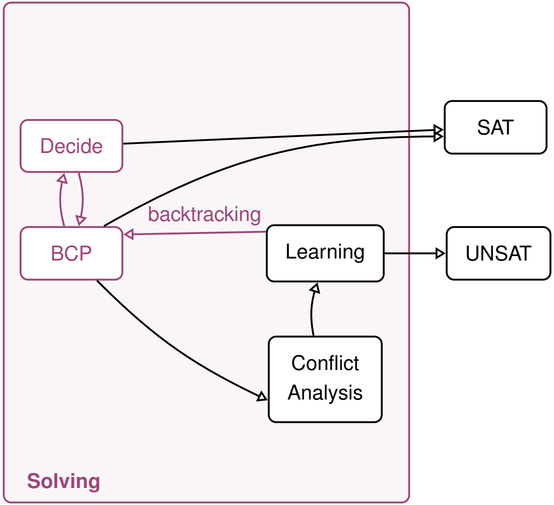
IPASIR-UP: IPASIR with User Propagators

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



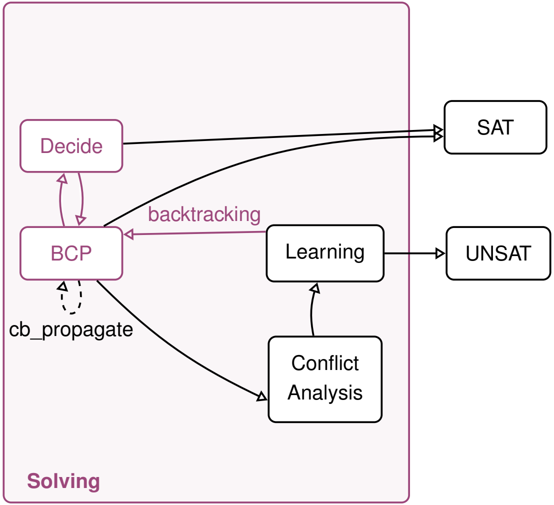
IPASIR-UP: IPASIR with User Propagators

- Supports interactions **during** the solve () calls
- Inspect search
  - Notify all changes to the trail



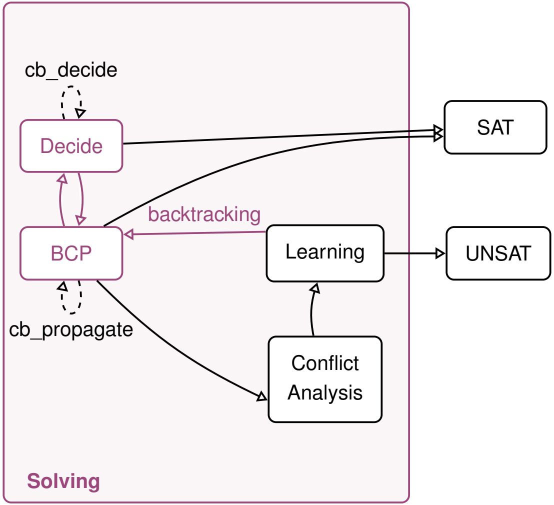
IPASIR-UP: IPASIR with User Propagators

- Supports interactions **during** the solve () calls
- Inspect search
  - Notify all changes to the trail
- Influence search
  1. Add propagations (without adding reason clauses)



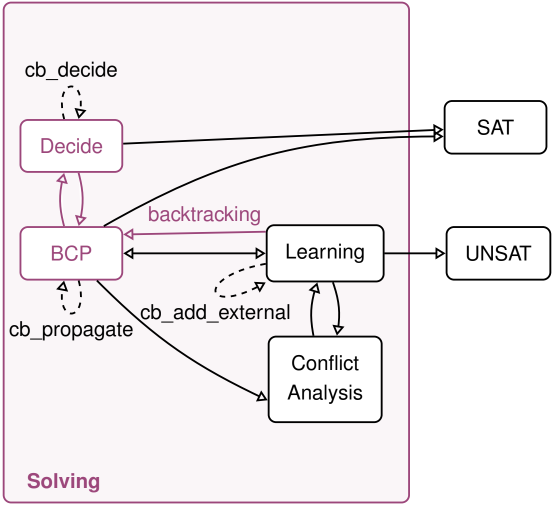
IPASIR-UP: IPASIR with User Propagators

- Supports interactions **during** the solve () calls
- Inspect search
  - Notify all changes to the trail
- Influence search
  1. Add propagations (without adding reason clauses)
  2. Dictate decisions & phases



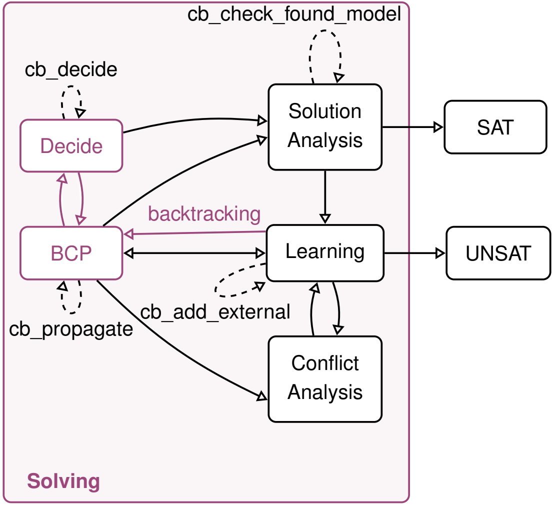
IPASIR-UP: IPASIR with User Propagators

- Supports interactions **during** the solve () calls
- 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!)



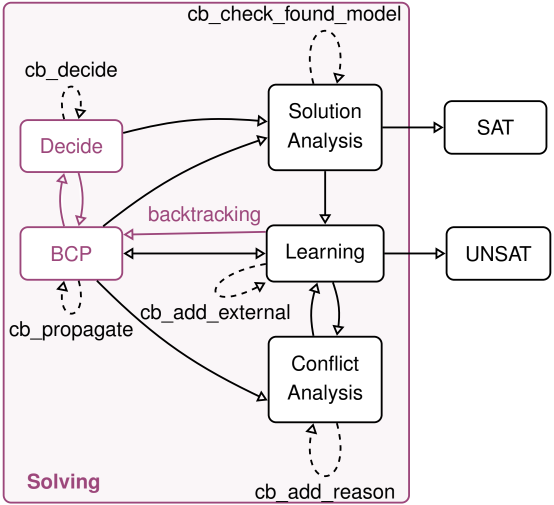
IPASIR-UP: IPASIR with User Propagators

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



IPASIR-UP: IPASIR with User Propagators

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



Related Work

- clingo [GebserKaminskiKaufmannOstrowskiSchaubWanko'16]
  - A state-of-the-art ASP solver
  - Supports *theory propagators*
- IntelSAT [Nadel'22]
  - Add clauses during search while maintaining propagation levels
- CP solvers [GentMiguelMoore'10]
  - Lazy explanation, lazy clause generation
- SAT solvers of SMT solvers [NieuwenhuisOliverasTinelli'06]
  - SAT worker interface [CimattiGriggioSchaafsmaSebastiani'13]
  - User propagators of z3 [BjørnerEisenhoferKovács'22]

Propagators are widely used, but there is no standard about how to do it.

IPASIR-UP Experiments

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

- Evaluated on two representative use cases
  - Combinatorial problem solving: SAT modulo Symmetries (SMS)
  - Satisfiability modulo Theories: cvc5

IPASIR-UP for SAT modulo Symmetries – Example

- Goal: Find complete set of non-isomorphic graphs over n vertices
  - Enumerate graphs with lexicographically minimal adjacency matrix
  - Would require exponential number of additional clauses
- `cb_add_clause`:
  - Enforce minimality
  - Block models without restarting the search
- `cb_propagate`:
  - Reduce number of added clauses

IPASIR-UP for SAT modulo Symmetries – Example

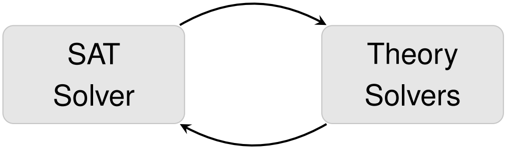
- Goal: Find complete set of non-isomorphic graphs over n vertices
  - Enumerate graphs with lexicographically minimal adjacency matrix
  - Would require exponential number of additional clauses
- `cb_add_clause`:
  - Enforce minimality
  - Block models without restarting the search
- `cb_propagate`:
  - Reduce number of added clauses

			CaDiCaL+IPASIR-UP [s]			Clingo [s]	
	#vertices	#graphs	*default*	*enum-IPASIR*	*no-prop*	*red*	*irred*
All graphs	6	156	0.01	0.02	0.01	0.02	0.01
	7	1044	0.09	0.13	0.09	0.10	0.09
	8	12346	0.95	1.59	1.00	1.15	1.07
	9	274668	34.24	64.27	34.31	81.67	94.65
	10	12005168	50815.60	109443.72	57616.47	213959.23	196576.58

IPASIR-UP for Satisfiability Modulo Theories – cvc5

Satisfiability Modulo Theories (SMT):

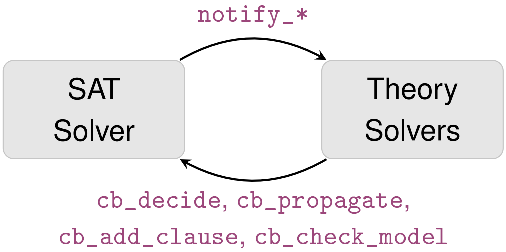
- Satisfiability of a first-order formula w.r.t. some background theories
- Example theories: Arrays, bit-vectors, Arithmetic, . . .
- Back-end solvers in verification, synthesis, planning, optimization, etc.
- Lazy CDCL($\mathcal{T}$) approach:



IPASIR-UP for Satisfiability Modulo Theories – cvc5

Satisfiability Modulo Theories (SMT):

- Satisfiability of a first-order formula w.r.t. some background theories
- Example theories: Arrays, bit-vectors, Arithmetic, ...
- Back-end solvers in verification, synthesis, planning, optimization, etc.
- Lazy CDCL($\mathcal{T}$) approach:



IPASIR-UP in cvc5

- **cvc5:** State-of-the-art SMT solver
  - Best ranking in several tracks of SMT'22 competition
- Supports all standard background theories
- Functions beyond SMT: abduction, interpolation, syntax-guided synthesis

IPASIR-UP in cvc5

- **cvc5:** State-of-the-art SMT solver
  - Best ranking in several tracks of SMT'22 competition
- Supports all standard background theories
- Functions beyond SMT: abduction, interpolation, syntax-guided synthesis
- Highly tuned for a customized version of `MiniSat`

IPASIR-UP in cvc5

- **cvc5:** State-of-the-art SMT solver
  - Best ranking in several tracks of SMT'22 competition
- Supports all standard background theories
- Functions beyond SMT: abduction, interpolation, syntax-guided synthesis
- Highly tuned for a customized version of `MiniSat`

cvc5 with CaDiCaL through IPASIR-UP:

- Additional $\sim$700 LOC to work with IPASIR-UP
- Encouraging performance without much tuning or optimizations
  - $+$1080 solved instances
  - $\sim 2\times$ faster in several logics
  - 13 out of 19 divisions are improved
- Any IPASIR-UP supporting SAT solver can be plugged in

  $\rightarrow$ solid baseline to tune and improve cvc5's internals for the IPASIR-UP interface

Summary

- 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

Summary

- Generic interface to inspect and influence CDCL search
  - Simple & Flexible $\rightarrow$ 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)
  - $\rightarrow$ Captures the necessary interactions of a very wide range of use cases

Future Work

- Consider further extensions
  - Propagate assumptions, guide backtrack, query/change variable scores, ...
- Proofs
  - Incremental (inprocessing) proofs
  - External proofs of external clauses
- More inprocessing

Thank you for your attention!

IPASIR Interface

```c
// Get solver name and version
const char* ipasir_signature();
// Initialize a solver instance and return a pointer to it
void* ipasir_init();
// Destroy the solver instance
void ipasir_release(void* solver);
// Set a callback function for aborting solving
void ipasir_set_terminate(void* solver, void* state,
                          int (*terminate)(void* state));
// Add a literal or finalize clause
void ipasir_add(void* solver, int lit_or_zero);
// Assume a literal for the next solve call
void ipasir_assume(void* solver, int lit);
// Solve the formula
int  ipasir_solve(void* solver);
// Retrieve a variables truth value (SAT case)
int  ipasir_val(void* solver, int lit);
// Check for a failed assumption (UNSAT case)
int  ipasir_failed(void* solver, int lit);
```

Functions to Manage and Configure

```
// 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);
```

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; }
};
```

Complete SMS Results

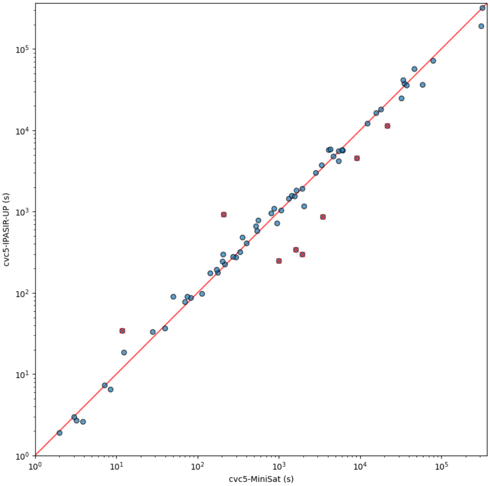
	#vertices	#graphs	CaDiCaL+IPASIR-UP [s]			Clingo [s]	
			default	*enum-IPASIR*	*no-prop*	*red*	*irred*
All graphs	6	156	0.01	0.02	0.01	0.02	0.01
	7	1044	0.09	0.13	0.09	0.10	0.09
	8	12346	0.95	1.59	1.00	1.15	1.07
	9	274668	34.24	64.27	34.31	81.67	94.65
	10	12005168	50815.60	109443.72	57616.47	213959.23	196576.58
	#vertices	#graphs	*default*	*no-inpro*	*no-prop*	*red*	*irred*
KS candidates	16	0	10.58	9.14	13.58	25.07	18.56
	17	1	39.82	31.48	44.58	122.28	87.92
	18	0	190.16	59.37	187.29	872.98	493.17
	19	8	1220.51	1253.96	1341.80	10542.41	3348.14
	20	147	13647.66	16449.50	13493.86	67728.42	82871.65

- Task 2: Generate up to isomorphism all non-010-colorable graphs with a minimum degree of at least three not containing a cycle of length 4 (used for Kochen-Specker Theorem)

cvc5 Results (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

cvc5 Results (Logics)



- Considered only the commonly solved instances