IPASIR-UP: User Propagators for CDCL
A CaDiCaL Integration into CDCL(\(T\))

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  \hspace{1cm}  \(^2\) Stanford University  \hspace{1cm}  \(^3\) University of Freiburg
IPASIR-UP in a Nutshell

**IPASIR-UP** = **IPASIR** + **User Propagators**

» a SAT solver **interface** for

» **interactive** incremental SAT solving

» **Our focus here:** Integration as CDCL($T$) 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(\text{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
» **Re-entrant Incremental Satisfiability API (IPASIR)**

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

- **BCP**
  - Decide
  - backtracking
- **Learning**
- **Conflict Analysis**

Solving

- **SAT**
- **UNSAT**
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(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
» **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

<table>
<thead>
<tr>
<th>Division</th>
<th>CVC5 solved</th>
<th>CVC5 time [s]</th>
<th>CVC5-IPASIRUP solved</th>
<th>CVC5-IPASIRUP time [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arith (6,865)</td>
<td>6,303</td>
<td>173,628</td>
<td>6,299</td>
<td>176,278</td>
</tr>
<tr>
<td>BitVec (6,045)</td>
<td>5,552</td>
<td>153,899</td>
<td>5,529</td>
<td>161,482</td>
</tr>
<tr>
<td>Equality (12,159)</td>
<td>5,320</td>
<td>2,062,804</td>
<td>5,322</td>
<td>2,061,758</td>
</tr>
<tr>
<td>Equality+LinearArith (53,453)</td>
<td>45,902</td>
<td>2,288,230</td>
<td>45,906</td>
<td>2,288,352</td>
</tr>
<tr>
<td>Equality+MachineArith (6,071)</td>
<td>983</td>
<td>1,533,646</td>
<td>987</td>
<td>1,532,782</td>
</tr>
<tr>
<td>Equality+NonLinearArith (21,104)</td>
<td>13,314</td>
<td>2,419,535</td>
<td>13,053</td>
<td>2,486,588</td>
</tr>
<tr>
<td>FPArith (3,965)</td>
<td>3,145</td>
<td>268,628</td>
<td>3,155</td>
<td>266,245</td>
</tr>
<tr>
<td>QF_Bitvec (42,472)</td>
<td>40,321</td>
<td>984,880</td>
<td>40,320</td>
<td>985,946</td>
</tr>
<tr>
<td>QF_Datatypes (8,403)</td>
<td>8,077</td>
<td>110,704</td>
<td>8,168</td>
<td>82,878</td>
</tr>
<tr>
<td>QF_Equality (8,054)</td>
<td>8,044</td>
<td>9,394</td>
<td>8,047</td>
<td>7,169</td>
</tr>
<tr>
<td>QF_Equality+Bitvec (16,585)</td>
<td>15,817</td>
<td>307,558</td>
<td>16,015</td>
<td>234,369</td>
</tr>
<tr>
<td>QF_Equality+LinearArith (3,442)</td>
<td>3,388</td>
<td>23,041</td>
<td>3,381</td>
<td>23,465</td>
</tr>
<tr>
<td>QF_Equality+NonLinearArith (709)</td>
<td>627</td>
<td>27,428</td>
<td>629</td>
<td>27,598</td>
</tr>
<tr>
<td>QF_FPArith (76,238)</td>
<td>76,054</td>
<td>94,487</td>
<td>76,081</td>
<td>76,700</td>
</tr>
<tr>
<td>QF_LinearIntArith (16,387)</td>
<td>11,670</td>
<td>1,575,635</td>
<td>12,004</td>
<td>1,512,696</td>
</tr>
<tr>
<td>QF_LinearRealArith (2,008)</td>
<td>1,721</td>
<td>130,408</td>
<td>1,766</td>
<td>113,919</td>
</tr>
<tr>
<td>QF_NonLinearIntArith (25,361)</td>
<td>13,037</td>
<td>4,094,712</td>
<td>13,682</td>
<td>3,840,933</td>
</tr>
<tr>
<td>QF_NonLinearRealArith (12,134)</td>
<td>11,166</td>
<td>333,933</td>
<td>11,238</td>
<td>316,728</td>
</tr>
<tr>
<td>QF_Strings (69,908)</td>
<td>69,357</td>
<td>203,677</td>
<td>69,296</td>
<td>230,918</td>
</tr>
<tr>
<td><strong>Total (391,363)</strong></td>
<td>339,798</td>
<td>16,796,234</td>
<td><strong>340,878</strong></td>
<td>16,426,813</td>
</tr>
</tbody>
</table>
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
```cpp
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_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