CaDiCaL(\(\mathcal{T}\)): CaDiCaL as CDCL(\(\mathcal{T}\)) Engine in cvc5

Aina Niemetz and Mathias Preiner

Shonan Meeting 180, October 2–5, 2023
The cvc5 SMT Solver

» state-of-the-art SMT solver
» most recent incarnation of the CVC tools
» joint project led by Stanford University and University of Iowa in collaboration with
  Universidade Federal de Minas Gerais (Brazil)
  Bar Ilan University (Israel)
» based on CDCL(\text{T}) framework
» supports wide range of theories in combination with quantifiers
  all SMT-LIB theories + non-standard theories and theory extensions
» support for proofs (incl. preprocessing, rewriting)
» capabilities beyond standard SMT
  SyGuS, abduction, interpolation, quantifier elimination, optimization (WIP)
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
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
CDCL(✓) 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

Situation in cvc5 (until recently)

» integrates highly customized version of MiniSat
  ■ produces resolution proofs
  ■ push/pop for adding/deleting clauses and variables
  ■ custom theory-guided decision heuristics
IPASIR-UP in a Nutshell

\[ \text{IPASIR-UP} = \text{IPASIR} + \text{User Propagators} \ (Fazekas \ et \ al. \ 2023) \]

» presented at SAT 2023
» a SAT solver \textbf{interface} for
» \textit{interactive} incremental SAT solving
IPASIR-UP in a Nutshell

**IPASIR-UP** = **IPASIR** + **U**ser **P**ropagators (Fazekas et al. 2023)

- presented at SAT 2023
- a SAT solver **interface** for
- **interactive** incremental SAT solving

» **our focus**: Integration as **CDCL(T)** SAT solver
CaDiCaL(\(\mathcal{T}\)) Integration (via IPASIR-UP)

» **Full utilization** of interface

» ~**500 LOC in C++** for implementing interface
  (~800 including comments)

» “easily” replaced with any SAT solver implementing IPASIR-UP

» **supports all**\(^*\) **cvc5 features**

» *proof support** work-in-progress

» **changes** compared to MiniSat
  - resolution proofs \(\rightarrow\) DRAT/LRAT proofs (WIP)
  - native push/pop \(\rightarrow\) activation literals
IPASIR-UP Interface

Notifications (for Inspecting CDCL Search)

» notify_assignment
» notify_new_decision_level
» notify_backtrack

Callbacks (for Influencing CDCL Search)

» cb_propagate
» cb_add_reason_clause_lit
» cb_decide
» cb_add_external_clause_lit
» cb_check_found_model
**notify_assignment**

» sends assignment notification for observed variables

» **track assignment** for theory literals
  
  ■ constructs (partial) assignment of propositional abstraction

» track whether assignment is
  
  ■ decision
  ■ fixed

» **notify** theory solvers about assigned theory literal,
  
e.g., if (observed) variable corresponding to theory literal $a < 42$ is assigned to
  
  ■ **true**: send $a < 42$ to arithmetic solver
  ■ **false**: send $a \geq 42$ to arithmetic solver
Decision Notifications

» Used to manage the incremental state of cvc5
» backtrackable data structures (context-dependent), associated with a context
  ■ SAT context, backtracks when SAT solver backtracks (decision-level push/pop)
  ■ user context (SMT-level push/pop)

notify_new_decision_level
  » push SAT context
  » track decision + level

notify_backtrack \((L)\)
  » pop SAT context to \(L\)
  » undo assignments at level \(> L\)
  » resend “popped” fixed theory literals
    ■ theory literals fixed at levels \(> L\) are popped
    ■ fixed assignments only notified once
    ■ resend fixed theory literals at level \(L\)
cb_check_found_model

» called when SAT solver found \textit{satisfying assignment}
  ▷ returns \texttt{true} if assignment \(\tau\)-consistent, \texttt{false} otherwise

» checks if assignment is \(\tau\)-consistent (\textit{full effort} check)
  ■ theory solvers check \(\tau\)-consistency of assigned theory literals
    » send \texttt{conflict} clause
    » send \texttt{lemmas}
    » send \texttt{theory propagations}
      ▷ adds \texttt{eager explanations} at this point to force SAT solver to propagate

■ \(\tau\)-consistent if:
  » theory solvers performed all checks and
  » no new \texttt{variables} were added and
  » no new \texttt{lemmas or conflicts} sent, i.e., no new clauses added
cb_decide

» called before SAT solver makes decision
» used to inject theory-guided decisions
  ■ theory decisions (required)
    ▷ decision strategies used by theory solvers
  ■ decision requests (optional)
    ▷ custom decision heuristics
    ▷ e.g.: justification heuristic, chooses next decision based on structure of formula

» may discover partial satisfying assignment
  ■ triggers full effort check, i.e., calls cb_check_found_model
  ■ stops search if current assignment is $\mathcal{T}$-consistent
Adding Clauses (during Search)

**cb_add_external_clause_lit**

- clauses added **during search are buffered**
  - theory lemmas
  - theory conflicts
- buffered clauses are only **added during callback**

**cb_has_external_clause**

- checks whether new clauses are pending
cb_propagate

» called after SAT solver is done with propagation
» performs lightweight checks in theory solvers (standard effort check)
» theory propagations

cb_add_reason_clause_lit (prop_lit)

» called when theory propagation prop_lit is involved in conflict
» explain theory propagation
» adds explanation (reason clause)
SMT push/pop via Activation Literals

» **happens between** SAT solver calls, **not during search**

» **push** assertion level
  ■ create fresh activation literal $l_n$ for pushed level $n$
  ■ add $l_n$ to each clause added in level $n$
  ■ prior to solving, assume $\neg l_i$ for $i \in \{1, n\}$

» **pop** assertion level
  ■ add unit clause $\{l_n\}$ for popped level $n$
    ▷ garbage collects all clauses added at level $n$
  ■ **unobserve** and **fix value** of variables introduced in $n$
    (important for performance)

» **renotify fixed literals** with fixed level $> \text{intro level}$
  ■ **requires** keeping track of assertion levels when
    » variable was introduced
    » variable assignment was fixed

(set-logic ...)
...
(assert ...) ; A1
(assert ...) ; A2
(push 1)
  (assert ...) ; A3
  (check-sat)
(pop 1)
  (check-sat)
...
Evaluation

- all **incremental** and **non-incremental** benchmarks of SMT-LIB 2023
  - 434,212 non-incremental benchmarks
  - 43,287 incremental benchmarks

- 300s time limit, 8GB memory limit

- comparison of cvc5-1.0.8-dev with
  - **MiniSat** (custom, based on 2.2.0)
  - **CaDiCaL** (IPASIR-UP, version 1.7.4)
# Evaluation: SMT-COMP Non-Incremental Divisions

<table>
<thead>
<tr>
<th>Division</th>
<th>CVC5+MiniSat solved</th>
<th>CVC5+CaDiCAL solved</th>
<th>CVC5+MiniSat time [s]</th>
<th>CVC5+CaDiCAL time [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arith (6,925)</td>
<td><strong>6,341</strong></td>
<td>181,329</td>
<td>6,332</td>
<td>183,417</td>
</tr>
<tr>
<td>BitVec (6,185)</td>
<td><strong>5,645</strong></td>
<td>168,844</td>
<td>5,625</td>
<td>175,110</td>
</tr>
<tr>
<td>Equality (12,159)</td>
<td>5,331</td>
<td>2,060,608</td>
<td><strong>5,337</strong></td>
<td>2,059,279</td>
</tr>
<tr>
<td>Equality+LinearArith (56,562)</td>
<td><strong>45,970</strong></td>
<td>3,196,706</td>
<td><strong>45,966</strong></td>
<td>3,198,129</td>
</tr>
<tr>
<td>Equality+MachineArith (10,911)</td>
<td>1,073</td>
<td>2,958,372</td>
<td><strong>1,075</strong></td>
<td>2,958,746</td>
</tr>
<tr>
<td>Equality+NonLinearArith (21,162)</td>
<td><strong>13,333</strong></td>
<td>2,425,551</td>
<td><strong>13,123</strong></td>
<td>2,474,917</td>
</tr>
<tr>
<td>FPArith (3,979)</td>
<td>3,133</td>
<td>275,579</td>
<td><strong>3,138</strong></td>
<td>272,751</td>
</tr>
<tr>
<td>QF_Bitvec (46,191)</td>
<td><strong>43,735</strong></td>
<td>1,092,892</td>
<td><strong>43,713</strong></td>
<td>1,092,907</td>
</tr>
<tr>
<td>QF_Datatypes (8,403)</td>
<td>8,083</td>
<td>109,941</td>
<td><strong>8,158</strong></td>
<td>84,593</td>
</tr>
<tr>
<td>QF_Equality (8,054)</td>
<td>8,043</td>
<td>9,338</td>
<td><strong>8,047</strong></td>
<td>6,968</td>
</tr>
<tr>
<td>QF_Equality+Bitvec (16,801)</td>
<td>15,922</td>
<td>355,232</td>
<td><strong>16,132</strong></td>
<td>263,786</td>
</tr>
<tr>
<td>QF_Equality+LinearArith (3,644)</td>
<td>3,464</td>
<td>45,242</td>
<td><strong>3,497</strong></td>
<td>52,176</td>
</tr>
<tr>
<td>QF_Equality+NonLinearArith (906)</td>
<td><strong>721</strong></td>
<td>61,692</td>
<td>711</td>
<td>64,217</td>
</tr>
<tr>
<td>QF_FPArith (76,252)</td>
<td>76,072</td>
<td>93,150</td>
<td><strong>76,087</strong></td>
<td>77,682</td>
</tr>
<tr>
<td>QF_LinearIntArith (16,389)</td>
<td>11,530</td>
<td>1,604,847</td>
<td><strong>12,017</strong></td>
<td>1,489,186</td>
</tr>
<tr>
<td>QF_LinearRealArith (2,008)</td>
<td>1,686</td>
<td>142,921</td>
<td><strong>1,784</strong></td>
<td>107,522</td>
</tr>
<tr>
<td>QF_NonLinearIntArith (25,446)</td>
<td>13,076</td>
<td>4,800,649</td>
<td><strong>14,058</strong></td>
<td>3,696,580</td>
</tr>
<tr>
<td>QF_NonLinearRealArith (12,134)</td>
<td>11,155</td>
<td>336,630</td>
<td><strong>11,247</strong></td>
<td>309,251</td>
</tr>
<tr>
<td>QF_Strings (100,101)</td>
<td>98,407</td>
<td>619,928</td>
<td><strong>98,870</strong></td>
<td>483,260</td>
</tr>
<tr>
<td>Total (434,212)</td>
<td>372,720</td>
<td>19,839,459</td>
<td><strong>374,917</strong></td>
<td>19,050,487</td>
</tr>
</tbody>
</table>

» **+2197** solved instances

» **∼ 25% faster** on commonly solved instances

» **2–4× faster** in several logics

» **13 of 19** divisions **improved**
  - quantifier-free better overall
  - quantified logics a bit behind

» **promising performance** without much tuning or optimizations

» **solid baseline** for future tuning with IPASIR-UP interface
Evaluation: SMT-COMP Incremental Divisions

<table>
<thead>
<tr>
<th>Division</th>
<th>cvc5+MiniSat</th>
<th></th>
<th></th>
<th>cvc5+CaDiCaL</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>solved</td>
<td>time [s]</td>
<td>solved</td>
<td>time [s]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arith (11)</td>
<td>41,362</td>
<td>233</td>
<td>41,362</td>
<td>240</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BitVec (18)</td>
<td>36,114</td>
<td>2,992</td>
<td>36,117</td>
<td>3,031</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Equality (4,067)</td>
<td>46,256</td>
<td>620,984</td>
<td>46,216</td>
<td>623,400</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Equality+LinearArith (1,894)</td>
<td>431,172</td>
<td>57,390</td>
<td>430,552</td>
<td>59,637</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Equality+MachineArith (4)</td>
<td>818</td>
<td>310</td>
<td>818</td>
<td>309</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Equality+NonLinearArith (4,374)</td>
<td>82,721</td>
<td>651,804</td>
<td>83,801</td>
<td>644,742</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FPArith (10)</td>
<td>3,422</td>
<td>1,849</td>
<td>3,421</td>
<td>1,849</td>
<td></td>
<td></td>
</tr>
<tr>
<td>QF_Bitvec (2,590)</td>
<td>51,334</td>
<td>63,165</td>
<td>51,260</td>
<td>62,036</td>
<td></td>
<td></td>
</tr>
<tr>
<td>QF_Equality (1,778)</td>
<td>29,981</td>
<td>4,616</td>
<td>29,982</td>
<td>4,588</td>
<td></td>
<td></td>
</tr>
<tr>
<td>QF_Equality+Bitvec (3,633)</td>
<td>7,677</td>
<td>148,084</td>
<td>7,620</td>
<td>153,446</td>
<td></td>
<td></td>
</tr>
<tr>
<td>QF_Equality+Bitvec+Arith (664)</td>
<td>959</td>
<td>51,776</td>
<td>985</td>
<td>44,466</td>
<td></td>
<td></td>
</tr>
<tr>
<td>QF_Equality+LinearArith (3,947)</td>
<td>2,266,894</td>
<td>130,331</td>
<td>1,893,335</td>
<td>133,167</td>
<td></td>
<td></td>
</tr>
<tr>
<td>QF_Equality+NonLinearArith (1,018)</td>
<td>96,917</td>
<td>24,307</td>
<td>92,813</td>
<td>23,932</td>
<td></td>
<td></td>
</tr>
<tr>
<td>QF_FPArith (19,188)</td>
<td>538,936</td>
<td>955,264</td>
<td>560,379</td>
<td>745,166</td>
<td></td>
<td></td>
</tr>
<tr>
<td>QF_LinearIntArith (69)</td>
<td>1,332,173</td>
<td>17,582</td>
<td>1,089,226</td>
<td>17,109</td>
<td></td>
<td></td>
</tr>
<tr>
<td>QF_LinearRealArith (10)</td>
<td>482</td>
<td>3,004</td>
<td>571</td>
<td>2,918</td>
<td></td>
<td></td>
</tr>
<tr>
<td>QF_NonLinearIntArith (12)</td>
<td>349,862</td>
<td>3,603</td>
<td>326,463</td>
<td>3,603</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total (43,287)</td>
<td>5,317,080</td>
<td>2,737,301</td>
<td>4,694,921</td>
<td>2,523,646</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

» improvements in some logics
» overall performance not there yet
» poor performance on benchmarks with many check-sat calls
» overhead of activation literals?
Discussion on Incremental Performance

**Observation**
Performance poor on benchmarks with 
large number of check-sat calls

**Example:** kundu_true-*.smt2 (QF_LIA)

- 900k+ check-sat calls
- solved queries within 300 seconds
  - **MiniSat:** 148,997
  - **CaDiCaL:** 103,843
Discussion on Incremental Performance

Observation
Performance poor on benchmarks with large number of check-sat calls

Example: kundu_true-*.smt2 (QF_LIA)
  » 900k+ check-sat calls
  » solved queries within 300 seconds
    ■ MiniSat: 148,997
    ■ CaDiCaL: 103,843

Activation Literal Overhead Experiment

(push 1)  ... fresh literal $l_n$
(assert true)  ... add clause $(l_n \lor \top)$
(check-sat)  ... assume $\neg l_n$
(pop 1)  ... add clause $(l_n)$

» Repeated $N$ times in one benchmark
Discussion on Incremental Performance

Observation
Performance poor on benchmarks with large number of check-sat calls

Example: kundu_true-*.smt2 (QF_LIA)
- 900k+ check-sat calls
- solved queries within 300 seconds
  - MiniSat: 148,997
  - CaDiCaL: 103,843

Activation Literal Overhead Experiment
(push 1) ... fresh literal \( l_n \)
(assert true) ... add clause \( (l_n \lor \top) \)
(check-sat) ... assume \( \neg l_n \)
(pop 1) ... add clause \( (l_n) \)

Repeated \( N \) times in one benchmark

<table>
<thead>
<tr>
<th>( N )</th>
<th>MiniSat</th>
<th>CaDiCaL</th>
<th>Slowdown</th>
</tr>
</thead>
<tbody>
<tr>
<td>10k</td>
<td>265ms</td>
<td>462ms</td>
<td>1.7×</td>
</tr>
<tr>
<td>25k</td>
<td>625ms</td>
<td>1.8s</td>
<td>2.8×</td>
</tr>
<tr>
<td>50k</td>
<td>1.2s</td>
<td>5.8s</td>
<td>4.8×</td>
</tr>
<tr>
<td>75k</td>
<td>1.8s</td>
<td>11.9s</td>
<td>6.6×</td>
</tr>
<tr>
<td>100k</td>
<td>2.5s</td>
<td>20.3s</td>
<td>8.1×</td>
</tr>
</tbody>
</table>
Conclusion

Summary

» non-incremental performance solid
» incremental performance still lagging behind
» IPASIR-UP integration
  ■ simple and flexible
  ■ captures all functionality required by cvc5

What’s Next?

» DRAT/LRAT proof integration (WIP)
» SAT solver tuning (currently default options)
» improve performance on quantified problems
» improve incremental performance
» IPASIR-UP: reduce callbacks, notifications

https://cvc5.github.io
Figure 1: QF_S* (Logics with Strings)

Figure 2: QF_A* (Logics with Arrays)
Quantifier-free and Quantified Logics

Figure 3: Quantifier-free Logics

Figure 4: Quantified Logics
References