One Thousand and One Refinement: From CDCL to a Verified SAT Solver

Mathias Fleury
2020/01/28
When you start your proof
After a few days…
After a few days... Mistake!
Then you write your paper
Paper accepted = Proof correct
Then you extend your paper
Paper accepted = Proof correct

What must be updated?
What about ITPs?

When you start...
What about ITPs?

When you start...  Before you finish
Paper proofs vs proof assistants
IsaFoL project

Isabelle Formalisation of Logic
The IsaFoL project: motivation

Eat your own dog food

- case study for proof assistants and automatic provers

Build state-of-the-art libraries

- Automated Reasoning: The Art of Generic Problem Solving
  (ongoing textbook project by Christoph Weidenbach)

Focus on meta-theorems

- reuse proofs
- be general
Excerpts of the IsaFoL project:

- Resolution, ordered resolution, and prover by Schlichtkrull et al. [ITP’16, IJCAR’18, CPP’19]
- Superposition by Peltier [AFP’16]
- UNSAT Checker by Lammich [CADE 27]
- CDCL and SAT solver [IJCAR’16, JAR’16, IJCAI’17, CPP’19, NFM’19]
Excerpts of the IsaFoL project:

- Resolution, ordered resolution, and prover by Schlichtkrull et al. [ITP’16, IJCAR’18, CPP’19]
- Superposition by Peltier [AFP’16]
- UNSAT Checker by Lammich [CADE 27]
- CDCL and SAT solver [IJCAR’16, JAR’16, IJCAI’17, CPP’19, NFM’19]
Outline

CDCL

- Watched literals, heuristics
- Fancy data structure
- Generated code
- Executable

Formalisation length (total: 78 000 lines of code)
CDCL explanation

clauses
CDCL explanation

assignement clauses
CDCL explanation

assignement

clauses
CDCL explanation

assignement = trail

clauses
CDCL explanation

assignement = trail

clauses
assignement = trail

clauses
assignement = trail

clauses
assignement = trail

clauses
CDCL explanation

assignment = trail

clauses
Refinement by specialisation

Core of CDCL is DPLL+BJ

\[
\begin{align*}
\text{DPLL+BJ} &= \text{Propagate} + \text{Decide} + \text{Backjump} \\
\text{DPLL} &= \text{Propagate} + \text{Decide} + \text{Backtrack}
\end{align*}
\]

back to some decision

back to latest decision
Refinement by specialisation

Core of CDCL is DPLL+BJ

DPLL+BJ = Propagate + Decide + Backjump

DPLL = Propagate + Decide + Backtrack

How to maximize reuse?

Backtrack = Parametrised Backjump (Backtrack_cond)
**Backjump on paper**

if \( C \in N \) and \( M \models \neg C \) and there is a \( C' \) such that...
then \( (M, N) \Rightarrow_{CDCL} (M' L, N) \).

**Definition (Parametrised Backjump in Isabelle)**

if \( C \in N \) and \( M \models \neg C \) and there is a \( C' \) such that...
and \( BJ\_cond C' \)
then \( (M, N) \Rightarrow_{CDCL} (M' L, N) \).
Development hierarchy

DPLL < DPLL+BJ where
DPLL+BJ_Cond = DPLL_Cond
Development hierarchy

DPLL

specialises

DPLL+BJ

extends

CDCL

\[
\text{CDCL} = \text{DPLL+BJ} + \text{Learn} + \text{Forget}
\]
Development hierarchy

DPLL specialises DPLL+BJ

DPLL+BJ extends CDCL

CDCL refines CDCL+learn_BJ

Strategy used in most implementations: learn only backjump clause
Development hierarchy

- DPLL
- DPLL+BJ
- DPLL+BJ+restart
- CDCL
- CDCL+learn_BJ
- CDCL+learn_BJ+restart
- CDCL+learn_BJ+restart+T
- CDCL+restart
- specialises
- extends
- refines

CDCL = DPLL+BJ + Learn + Forget

Strategy used in most implementations: learn only backjump clause
Weidenbach’s CDCL

Definition (Parametrised Backjump (BJ_cond))

If $C \in N$ and $M \models \neg C$ and there is a $C'$ such that...

and $BJ\_cond \ C'$

then $(M, N) \Rightarrow_{CDCL} (L^\dagger M', N)$.

How to get a suitable $C'$?
Refinement by inclusion

CDCL_learn_BJ

Decide, propagate → Backjump + Learn

CDCL_W

Decide, propagate → Conflict
                 → Skip and resolve
                 → Jump + Learn
Refinement by inclusion

CDCL_learn_BJ

\[(M, N) \rightarrow (M, N + U)\]

CDCL_W

\[(M, N, U, D)\]
Refinement by inclusion

**CDCL\_learn\_BJ**

- Decide, propagate
- Backjump + Learn

**CDCL\_W**

- Decide, propagate
- Conflict
  - Skip and resolve
  - Jump+Learn

Terminating
Refining Data Structures

- CDCL
- Watched literals, heuristics, fancy data structure
- Generated code, executable

Heuristics: 36%

- WL: 33%
- CDCL: 10%
- Code: 21%

Formalisation length (total: 78,000 lines of code)
Watched literals explanation

Watched literals = sophisticated data structure to identify propagations and conflicts.
Watched literals = sophisticated data structure to identify propagations and conflicts.
Watched literals explanation

Watched literals = sophisticated data structure to identify propagations and conflicts.
Watched literals explanation

Watched literals = sophisticated data structure to identify propagations and conflicts.
First formalisation attempt failed.

Development done in two steps:
First formalisation attempt failed.

Development done in two steps:

1. watched literals...
2. ... extended with blocking literals
Watched literals

First formalisation attempt failed.

Development done in two steps:

1. watched literals...
2. ... extended with blocking literals

My Approach  non-deterministic transition system
Refinement in the non-determinism monad

Then we enter the non-determinism monad:

- closer to programs
- preserves non-determinism
Refinement in the non-determinism monad

Then we enter the non-determinism monad:

- closer to programs
- preserves non-determinism

Abstract level:

\[
\text{OBTAIN should\_restart such that } \text{should\_restart } \implies \#conflict > \text{threshold}
\]
Refinement in the non-determinism monad

Then we enter the non-determinism monad:

- closer to programs
- preserves non-determinism

Abstract level:

\[
\text{OBTAIN should\_restart such that}
\]

\[
\text{should\_restart} \implies \#\text{conflict} > \text{threshold}
\]

Concrete level:

\[
\text{should\_restart} \leftarrow \text{RETURN} (\#\text{conflict} > \text{threshold} \land \text{heuristic})
\]
Refinement to keep abstractions

CDCL
Weidenbach

Watched clauses
with multisets

Watched clauses
clauses as lists

Watch lists
efficient indexing
Refinement to keep abstractions

CDCL
Weidenbach

Watched clauses
with multisets

Watched clauses
clauses as lists

Watch lists
efficient indexing

Isasat
deterministic with heuristics
Generating Code

CDCL

- Watched literals, heuristics
- Fancy data structure

Generated code
- Executable

Formalisation length (total: 78,000 lines of code)

Heuristics

CDCL

- 36%

WL

- 33%

Code

- 21%

10%
What is the imperative code?

IsaSAT
deterministic

Impetative IsaSAT
in Imperative HOL

IsaSAT/Standard ML (SML)
or Scala, OCaml, Haskell
Abstract code:

```
ASSERT(i < length xs);
RETURN(xs!i);
```
Abstract code:

```plaintext
ASSERT(i < length xs);
RETURN(xs!i);
```

After synthesis by Sepref in Imperative HOL:

```plaintext
Array.nth xs i
```
Abstract code:
```ml
ASSERT(i < length xs);
RETURN(xs!i);
```

After synthesis by Sepref in Imperative HOL:
```ml
Array.nth xs i
```

After printing in SML, via `code equations` and printing:
```ml
Array.sub(xs, i)
```
Abstract code:

```plaintext
ASSERT(i < length xs);
RETURN(xs!i);
```

After synthesis by Sepref in Imperative HOL:

```plaintext
Array.nth xs i
```

After printing in SML, via `code equations` and printing:

```plaintext
Array.sub(xs, i)
```

A native array
Code synthesis and generation

Abstract code:

```plaintext
ASSERT(i < length xs);
RETURN(xs!i);
```

After synthesis by Sepref in Imperative HOL:

```plaintext
Array.nth xs i
```

After printing in SML, via code equations and printing:

```plaintext
if i < Array.size xs
then xs[i]
else raise OutOfBound
```
Code synthesis and generation

Abstract code:

```plaintext
ASSERT(i < length xs);
RETURN(xs!i);
```

Information is lost during translation

After synthesis by Sepref in Imperative HOL:

```plaintext
Array.nth xs i
```

After printing in SML, via code equations and printing:

```plaintext
if i < Array.size xs then xs[i] else raise OutOfBound
```

In IsaSAT removed by a compiler flag...
Abstract code:

```ml
ASSERT(i < length xs);
RETURN(xs!i);
```

In the nice Isabelle world GMP integer

After synthesis by Sepref in Imperative HOL:

```ml
Array.nth xs i
```

After printing in SML, via code equations and printing:

```ml
if i < Array.size xs
then xs[i]
else raise OutOfBound
```
Abstract code:

```
ASSERT(i < length xs);
RETURN(xs!i);
```

In IsaSAT, uint64 integer until it does not fit

After synthesis by Sepref in Imperative HOL:

```
Array.nth xs i       Array.nth_uint64 xs i
```

After printing in SML, via code equations and printing:

```
if i < Array.size xs
then xs[i]          Array.nth_uint64(xs, i)
else raise OutOfBound
```
Theorem

If the input is well formed and UNSAT (resp. SAT), then IsaSAT terminates and it returns UNSAT (resp. SAT with a model). \(^1\)

\(^1\)if the Standard ML compiler is able to allocate large enough arrays
Correctness theorem

Theorem

If the input is well formed and UNSAT (resp. SAT), then IsaSAT terminates and it returns UNSAT (resp. SAT with a model).\(^1\)

And the only other efficient verified solver

Theorem (Correctness versat)

If the input is well formed and the solver returns UNSAT, then the problem is UNSAT.

\(^1\)if the Standard ML compiler is able to allocate large enough arrays
Comparison of various SAT solvers on preprocessed instances
Comparison of various SAT solvers on preprocessed instances
Comparison of various SAT solvers on preprocessed instances
Conclusion
Conclusion

CDCL
extensible

Watched Literals
fancy data structure

Generated Code
executable
Conclusion

CDCL
extensible

Optimizing CDCL
CDCL+brand-and-bound

Watched Literals
fancy data structure

Generated Code
executable
O Captain! My Captain!

Now comes the appendix, go back to the previous slide
Appendix Outline

What is hard?
Refinement
Correctness and Trust
Features
Missing Features
CDCL
Complexity
Importing Correctness in Isabelle
IsaSAT/LLVM vs IsaSAT/MLton
Performance
OCDCL
Related Work
What is hard?
Why is it so hard?
<table>
<thead>
<tr>
<th>Formalisation part</th>
<th>Length (kloc)</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>CDCL Libraries</td>
<td>3</td>
<td>Entailment</td>
</tr>
<tr>
<td>CDCL</td>
<td>17</td>
<td></td>
</tr>
<tr>
<td>Refinement Libraries</td>
<td>6</td>
<td>Setup for machine words, arrays of arrays</td>
</tr>
<tr>
<td>Refinement except last layer</td>
<td>26</td>
<td></td>
</tr>
<tr>
<td>Heuristics</td>
<td>35</td>
<td>code synthesis, lots of code</td>
</tr>
<tr>
<td>Formalisation part</td>
<td>Length (kloc)</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------------</td>
<td>---------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>CDCL Libraries</td>
<td>3</td>
<td>Entailment</td>
</tr>
<tr>
<td>CDCL</td>
<td>17</td>
<td>Setup for machine words, arrays of arrays</td>
</tr>
<tr>
<td>Refinement Libraries</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>Refinement except last layer</td>
<td>26</td>
<td></td>
</tr>
<tr>
<td>Heuristics</td>
<td>35</td>
<td>code synthesis, lots of code</td>
</tr>
<tr>
<td>Formalisation part</td>
<td>Length (kloc)</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------------</td>
<td>---------------</td>
<td>------------------------------------------------------</td>
</tr>
<tr>
<td>CDCL Libraries</td>
<td>3</td>
<td>Entailment</td>
</tr>
<tr>
<td>CDCL</td>
<td>17</td>
<td>Setup for machine words, arrays of arrays</td>
</tr>
<tr>
<td>Refinement Libraries</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>Refinement except last layer</td>
<td>26</td>
<td></td>
</tr>
<tr>
<td>Heuristics</td>
<td>35</td>
<td>code synthesis, lots of code</td>
</tr>
<tr>
<td>Formalisation part</td>
<td>Length (kloc)</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------------</td>
<td>---------------</td>
<td>-------------</td>
</tr>
<tr>
<td>CDCL Libraries</td>
<td>3</td>
<td>Entailment</td>
</tr>
<tr>
<td>CDCL</td>
<td>17</td>
<td>Setup for machine words, arrays of arrays</td>
</tr>
<tr>
<td>Refinement Libraries</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>Refinement except last layer</td>
<td>26</td>
<td></td>
</tr>
<tr>
<td>Heuristics</td>
<td>35</td>
<td>code synthesis, lots of code</td>
</tr>
</tbody>
</table>
Mostly about definitions

Aliasing and ownership

Formalisation part

Length (kloc)

Single threaded

CDCL Libaries 3

Entailment

CDCL 17

Setup for machine words, arrays of arrays

Refinement Libraries 6

Refinement except last layer 26

Heuristics 35

code synthesis, lots of code
Refinement
Refinement in the non-determinism monad: Data structure

Abstract level:

\[ \text{OBTAIN } L \text{ s.t. } L \in C \]

Concrete level:

\[ \text{blit } \leftarrow \text{RETURN(}\text{watcher.blit}\text{)} \]
Correctness and Trust
And IsaSAT/LLVM:

**Theorem (Correctness IsaSAT/LLVM)**

*If the input is a valid input and the solver returns SAT (UNSAT), then the problem is SAT (UNSAT).*
Isabelle protects of:

- programming errors (out-of-bound)
- correctness errors (SAT instead of UNSAT)

But not of:

- performance bugs (restarts)
<table>
<thead>
<tr>
<th>IsaSAT/SML</th>
<th>IsaSAT/LLVM</th>
<th>CaDiCaL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parser</td>
<td>Parser</td>
<td>The parser</td>
</tr>
<tr>
<td>Code equations</td>
<td>Isabelle’s LLVM Semantics</td>
<td>CDCL Implementation</td>
</tr>
<tr>
<td>Compiler</td>
<td>LLVM</td>
<td>Compiler</td>
</tr>
<tr>
<td></td>
<td>~2 faster than SML, ~10 times less memory</td>
<td></td>
</tr>
</tbody>
</table>
### What do you trust?

<table>
<thead>
<tr>
<th></th>
<th>IsaSAT/SML</th>
<th>IsaSAT/LLVM</th>
<th>CaDiCaL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parser</td>
<td>Parser</td>
<td>The parser</td>
<td></td>
</tr>
<tr>
<td>Code equations</td>
<td>Isabelle’s LLVM Semantics</td>
<td>CDCL</td>
<td>Implementation</td>
</tr>
<tr>
<td>Compiler</td>
<td>LLVM</td>
<td>Compiler</td>
<td></td>
</tr>
</tbody>
</table>

~2 faster than SML, ~10 times less memory

There is no bug that happens after two years of calculation because you wrote `uint64_max - 4` instead of `uint64_max - 5`
Features
<table>
<thead>
<tr>
<th>Technique</th>
<th>Evaluation</th>
</tr>
</thead>
<tbody>
<tr>
<td>VMTF decision heuristic</td>
<td>Critical</td>
</tr>
<tr>
<td>Conflicts as hash-table and array</td>
<td>Critical</td>
</tr>
<tr>
<td>Recursive conflict minimization</td>
<td>Critical</td>
</tr>
<tr>
<td>Arena-based memory</td>
<td>I never saw a difference</td>
</tr>
<tr>
<td>Blocking literals + position saving</td>
<td>Helps a lot</td>
</tr>
<tr>
<td>EMA-14 restarts + trail reuse</td>
<td>Helps, but I still don’t understand what CaDiCaL does</td>
</tr>
<tr>
<td>Special handling of binary clauses</td>
<td>I never saw a difference</td>
</tr>
</tbody>
</table>
Missing Features
Missing Features

Two trivial but key features

• deletion of true clauses
• removal of false literals

Solution: “pragmatic CDCL” with resolution rules to simplify clauses set
Easy to add:

**Definition (Conflict Minimisation)**

Learn a clause $D' \lor L' \subseteq D \lor L$ if $N \models D' \lor L'$.

Impossible to add (it breaks invariants):

**Definition (Inprocessing)**

An irredundant clause is subsumed by a learned clause: make the latter irredundant.
but!

If we go with

$$(M, N, N_{\text{subsumed}}, U, U_{\text{subsumed}}, D)$$

and do not consider subsumed clauses, CDCL can see

$$(M, N + N_{\text{subsumed}}, U + U_{\text{subsumed}}, D)$$

and everything will work as expected.
Complexity
As for SAT implementations,

**Never-ending task** there is always one more heuristic or one more technique to implement...

**No tooling** ... makes it even harder

**Testing a heuristic** is hard
On the proof side

Proving Correctness: time consuming (overflow problems), Isabelle is slow

Side conditions of CDCL

<table>
<thead>
<tr>
<th>Property (CDCL Invariant)</th>
</tr>
</thead>
<tbody>
<tr>
<td>The set of all literals you consider is exactly the set of literals in the set of clauses.</td>
</tr>
<tr>
<td>Evaluator</td>
</tr>
<tr>
<td>-----------------</td>
</tr>
<tr>
<td>MLton</td>
</tr>
<tr>
<td>PolyML</td>
</tr>
<tr>
<td>nbe, simp</td>
</tr>
</tbody>
</table>
What makes refinement hard?

Refinement is easy when:

• you can ignore the result of operations
• i.e., reduce interdependency between components of the state

\[
M \leftarrow \text{RETURN (Decided L . M)}
\]

What is the impact on the other components?
What makes refinement hard?

- Trail
- Every literal is in clauses

- Clauses
- Every literal is in conflict clause

- Conflict clause
- Decision heuristic
What makes refinement hard?

trail clauses conflict clause decision heuristic

every literal is in every literal is in
Importing Correctness in Isabelle
Idea

Abstract code:

```plaintext
ASSERT(i < length xs);
RETURN (xs ! i);
```

After synthesis, done automatically by Sepref:

```plaintext
return xs[i]
```

Can we run it in Isabelle?

- result cannot be extracted from the `return` (imperative monad)...
- ... but we can generate a purely functional version...
- ... which is what I optimised for
<table>
<thead>
<tr>
<th>Evaluator</th>
<th>Performance</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLton</td>
<td>2.5 s</td>
<td>includes parsing</td>
</tr>
<tr>
<td>PolyML</td>
<td>43 s</td>
<td>requires 64-bit PolyML</td>
</tr>
<tr>
<td>value</td>
<td>?</td>
<td>do not know about Imperative HOL, so you cannot allocate arrays</td>
</tr>
<tr>
<td>nbe, simp</td>
<td>⊥</td>
<td></td>
</tr>
</tbody>
</table>
IsaSAT/LLVM vs IsaSAT/MLton
LLVM is better and has an easier job

- LLVM has more man-power: MLton’s LLVM backend produces slightly better code.
- LLVM’s IR is the target for tools vs target for humans (Isabelle’s code generator produces terrible and unreadable code).
- LLVM’s input is the code you would expect.
LLVM has more freedom to do a good job

- The code is not functional at all and contains barely any datatype
- ML enforces sharing, which is good until is not
  1. $\lambda(\#props, stats). (\#props + 1, stats)$ reallocates
  2. `clause_ref * (bool * literal)` needs more memory than
     `struct {clause_ref; struct {bool; literal};};` (cache problems!)
- Array access and conversions are checked\(^3\)

---
\(^2\) Isabelle is not able to generate `clause_ref * bool * literal` and using a tuple made things worse
\(^3\) although I deactivate these checks
Memory is not cheap

- IsaSAT/ML uses 10 times more memory
- IsaSAT/ML uses the GC... but I have no idea why: IsaSAT uses base types (or with in-place operations) and arrays resizing (freeing the old one is enough)
Performance
Comparison of various SAT solvers on preprocessed instances
OCDCL
Conjecture

OCDCL+stgy performs at most $3^n$ Backtrack steps.
<table>
<thead>
<tr>
<th>Lemma (verified in Isabelle)</th>
<th>Conjecture</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>ODPLL+stgy performs at most</em> (3^n) <em>Backtrack steps.</em></td>
<td><em>OCDCL+stgy performs at most</em> (3^n) <em>Backtrack steps.</em></td>
</tr>
<tr>
<td>Lemma (verified in Isabelle)</td>
<td>Conjecture</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td><em>ODPLL+stgy performs at most</em> 3(^n) <em>Backtrack steps.</em></td>
<td><em>OCDCL+stgy performs at most</em> 3(^n) <em>Backtrack steps.</em></td>
</tr>
</tbody>
</table>

**Proof.**

- trails are not repeated
- trails have a certain form
- and they are such 3\(^n\) such trail
<table>
<thead>
<tr>
<th>Lemma (verified in Isabelle)</th>
<th>Conjecture</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ODPLL+stgy performs at most</strong> $3^n$ Backtrack steps.</td>
<td><strong>OCDCL+stgy performs at most</strong> $3^n$ Backtrack steps.</td>
</tr>
</tbody>
</table>

**Proof.**

- trails are not repeated
- trails have a certain form
- and they are such $3^n$ such trail

**Proof.**

- trails are not repeated
- trails have a certain form
- and they are such $3^n$ such trail

Problem: backjump is nearly a restart.
Related Work
<table>
<thead>
<tr>
<th>Related Work</th>
<th>Marić</th>
<th>Les-cuyer</th>
<th>Schankar</th>
<th>Oe et al</th>
</tr>
</thead>
<tbody>
<tr>
<td>2008</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Isabelle</td>
<td>*</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Backjumping Learning</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Soundness</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Compeleteness</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Implementation</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Termination</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Restart+Forget</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>WL</td>
<td>~</td>
<td>-</td>
<td>-</td>
<td>~</td>
</tr>
</tbody>
</table>