Satisfiability as Abstract Interpretation

Leopold Haller
(joint work with Vijay D’Silva)
A Tale of Two Communities

AI (1977)
over-approximate, sound

GAP

under-approximate, precise

DP (1960)  DPLL (1962)

CDCL (1996)

Learning, Efficiency!
A Tale of Two Communities

AI (1977)
over-approximate, sound

GAP

under-approximate, precise

BMC (2003)
Find bugs

DP (1960)

DPLL (1962)

CDCL (1996)
Learning, Efficiency!

Tuesday, 13 September 11
A Tale of Two Communities

AI (1977)
over-approximate, sound

GAP

under-approximate, precise

DP (1960)  DPLL (1962)

BMC (2003)
Find bugs

CDCL (1996)
Learning. Efficiency!

SMT (early 00s)
Richer logics

---
Tuesday, 13 September 11
A Tale of Two Communities

- AI (1977)
  - over-approximate, sound
- DP (1960)
  - under-approximate, precise
- DPLL (1962)
  - Flexible Disjunction (2005)
  - Trace Partitioning (2005)
  - Find bugs
- CDCL (1996)
  - Learning, Efficiency!
- SMT (early 00s)
  - Richer logics
A Tale of Two Communities

AI (1977)

over-approximate, sound

GAP

under-approximate, precise

CDCL (1996)

Learning. Efficiency!

BMC (2003)

Find bugs

SMT (early 00s)

Richer logics

Trace Partitioning (2005)

Flexible Disjunction

Harris et al. (POPL 2010)

SMPP - Trace partition using SAT
A Tale of Two Communities

AI (1977)

over-approximate, sound

GAP

under-approximate, precise

DP (1960)

DPLL (1962)

BMC (2003)

CDCL (1996)

Learning, Efficiency!

Harris et al. (POPL 2010)

Cousot & Cousot & Mauborgne (FOSSACS 2011)

Trace Partitioning (2005)

Flexible Disjunction

SMPP - Trace partition using SAT

Combine AD and SMT

Find bugs

SMT (early 00s)

Richer logics

Tuesday, 13 September 11
A Tale of Two Communities

AI (1977) over-approximate, sound

DP (1960) under-approximate, precise

DPLL (1962)

BMC (2003)

CDCL (1996)

SMT (early 00s)

Richer logics

GAP

Trace Partitioning (2005)

Flexible Disjunction

SMPP - Trace partition using SAT

Harris et al. (POPL 2010)

Combine AD and SMT

Cousot & Cousot & Mauborgne (FOSSACS 2011)

Harris et al. (POPL 2010)

Find bugs

Monnieaux & Gonnord (SAS 2011)

CDCL (1996)

Learning. Efficiency!

SMT (early 00s)

Richer logics

Focus on path using SMT
A Tale of Two Communities

DPLL (1962)  CDCL (1996)  solvers

are

(proper) abstract interpreters

AI (1977)
Why does this matter?
“the practical success of SAT has come as a surprise to many in the computer science community. The combination of strong practical drivers and open competition in this experimental research effort created enough momentum to overcome the pessimism based on theory. Can we take these lessons to other problems and domains?”

– Malik & Zhang, 2009
Why does this matter?
Why does this matter?
Why does this matter?

ACDCL(A) \rightarrow \text{Abstract domain A}
Why does this matter?

Abstract domain

ACDCL(A)

FO Logic

Programs

Prop. Logic

Finite State Programs

Data

Control

ACDCL(A)

Abstract domain A
Conflict Driven Clause Learning

Interpreting Logic

CDCL is Abstract Interpretation

ACDCL(A)
The CDCL Algorithm
Jargon Slide

Propositions
finite set $V$

Literal
$p, \neg p \quad p \in V$

Clause
disjunction of literals

CNF formula
conjunction of clauses

Assignment
partial function $V \rightarrow \{t, f\}$

Satisfiability
Does there exists an assignment $V \rightarrow \{t, f\}$ such that $\varphi$ is true?
The CDCL Algorithm

Propositional CNF formula $\varphi = (p \lor \neg q) \land \ldots \land (\neg r \land w \land q)$
The CDCL Algorithm

Propositional CNF formula $\varphi = (p \lor \neg q) \land \ldots \land (\neg r \land w \land q)$
The CDCL Algorithm

Propositional CNF formula $\varphi = (p \lor \neg q) \land \ldots \land (\neg r \land w \land q)$
The CDCL Algorithm

Propositional CNF formula $\varphi = (p \lor \neg q) \land \ldots \land (\neg r \land w \land q)$
The CDCL Algorithm

Propositional CNF formula \( \varphi = (p \lor \neg q) \land \ldots \land (\neg r \land w \land q) \)
The CDCL Algorithm

Propositional CNF formula $\varphi = (p \lor \neg q) \land \ldots \land (\neg r \land w \land q)$
Propositional CNF formula \( \varphi = (p \lor \neg q) \land \ldots \land (\neg r \land w \land q) \)
Boolean Constraint Propagation (BCP)

Operates over a partial function (variable assignment)

\[ V \rightarrow \{ t, f \} \]
Boolean Constraint Propagation (BCP)

Unit Rule
Boolean Constraint Propagation (BCP)

Unit Rule

\[ p \mapsto t \]
\[ q \mapsto f \]
\[ r \mapsto f \]

\[ \ldots \land (\neg p \lor q \lor r \lor \neg w) \land \ldots \]
Boolean Constraint Propagation (BCP)

Unit Rule

\[ p \mapsto t, \quad q \mapsto f, \quad r \mapsto f \]

\[ \ldots \land (\neg p \lor q \lor r \lor \neg w) \land \ldots \]
Boolean Constraint Propagation (BCP)

Unit Rule

\[ \ldots \wedge (\neg p \lor q \lor r \lor \neg w) \wedge \ldots \]
Boolean Constraint Propagation (BCP)

Unit Rule

\[ \ldots \land (\neg p \lor q \lor r \lor \neg w) \land \ldots \]
Boolean Constraint Propagation (BCP)

Unit Rule

\[ \ldots \wedge (\neg p \lor q \lor r \lor \neg w) \wedge \ldots \]

- \( p \mapsto t \)
- \( q \mapsto f \)
- \( r \mapsto f \)
- \( w \mapsto f \)
Boolean Constraint Propagation (BCP)

BCP = Exhaustive application of unit rule

\[ \varphi = p \land (\neg p \lor \neg q) \land (q \lor r \lor \neg w) \land (q \lor r \lor w) \]
Boolean Constraint Propagation (BCP)

BCP = Exhaustive application of unit rule

\[ \varphi = (p) \land (\neg p \lor \neg q) \land (q \lor r \lor \neg w) \land (q \lor r \lor w) \]
Boolean Constraint Propagation (BCP)

BCP = Exhaustive application of unit rule

\[ \varphi = (p) \land (\neg p \lor \neg q) \land (q \lor r \lor \neg w) \land (q \lor r \lor w) \]

Tuesday, 13 September 11
Boolean Constraint Propagation (BCP)

BCP = Exhaustive application of unit rule

\[ \varphi = p \land (\neg p \lor \neg q) \land (q \lor r \lor \neg w) \land (q \lor r \lor w) \]

\[ p \rightarrow t \]
Boolean Constraint Propagation (BCP)

BCP = Exhaustive application of unit rule

\[ \varphi = p \land (\neg p \lor \neg q) \land (q \lor r \lor \neg w) \land (q \lor r \lor w) \]

\[ p \rightarrow t \quad q \rightarrow f \]
\( \varphi = p \land (\neg p \lor \neg q) \land (q \lor r \lor \neg w) \land (q \lor r \lor w) \)

Decisions

\[ p \mapsto t \]
\[ q \mapsto f \]
\[
\varphi = p \land (\neg p \lor \neg q) \land (q \lor r \lor \neg w) \land (q \lor r \lor w)
\]

Decisions

Pick an unassigned variable and assign a truth value

\[
p \mapsto t
\]
\[
q \mapsto f
\]
Decide
Learn
Backtrack
BCP

conflict

Decisions

Pick an unassigned variable and assign a truth value

\[ \varphi = p \land (\neg p \lor \neg q) \land (q \lor r \lor \neg w) \land (q \lor r \lor w) \]

\[ p \mapsto t \]
\[ q \mapsto f \]

\[ p \mapsto t \]
\[ q \mapsto f \]
\[ r \mapsto f \]
The diagram illustrates the process of BCP (Backward Chaining Proposition), Learn, Decide, and Backtrack leading to SAT (Satisfiability) and UNSAT (Unsatisfiability). The formula for \( \varphi \) is given as:

\[
\varphi = p \land (\neg p \lor \neg q) \land (q \lor r \lor \neg w) \land (q \lor r \lor w)
\]

With the given assignments:

- \( p \mapsto \text{t} \)
- \( q \mapsto \text{f} \)
- \( r \mapsto \text{f} \)
\[ \varphi = p \land (\neg p \lor \neg q) \land (q \lor r \lor \neg w) \land (q \lor r \lor w) \]

\[ p \mapsto t \quad p \mapsto t \]
\[ q \mapsto f \quad q \mapsto f \]
\[ r \mapsto f \quad r \mapsto f \]
\[ w \mapsto f \]
\[
\varphi = p \land (\neg p \lor \neg q) \land (q \lor r \lor \neg w) \land (q \lor r \lor w)
\]
$\varphi = p \land (\neg p \lor \neg q) \land (q \lor r \lor \neg w) \land (q \lor r \lor w)$
\( \varphi = p \land (\neg p \lor \neg q) \land (q \lor r \lor \neg w) \land (q \lor r \lor w) \)

\[ p \mapsto t \]
\[ \varphi = p \land (\neg p \lor \neg q) \land (q \lor r \lor \neg w) \land (q \lor r \lor w) \]

\[ p \mapsto t \quad q \mapsto f \]
\[ \varphi = p \land (\neg p \lor \neg q) \land (q \lor r \lor \neg w) \land (q \lor r \lor w) \]

\[ p \mapsto t \quad q \mapsto f \]

\[ r \mapsto f \]
\[ \varphi = p \land (\neg p \lor \neg q) \land (q \lor r \lor \neg w) \land (q \lor r \lor w) \]

- \( p \leftrightarrow t \), \( q \leftrightarrow f \), \( w \leftrightarrow f \), \( r \leftrightarrow f \)
\[ \varphi = p \land (\neg p \lor \neg q) \land (q \lor r \lor \neg w) \land (q \lor r \lor w) \]

- \( p \mapsto t \)
- \( q \mapsto f \)
- \( r \mapsto f \)
- \( w \mapsto f \)
\( \varphi = p \land (\neg p \lor \neg q) \land (q \lor r \lor \neg w) \land (q \lor r \lor w) \)
\[ \varphi = p \land (\neg p \lor \neg q) \land (q \lor r \lor \neg w) \land (q \lor r \lor w) \]

\[ p \mapsto t \]

\[ q \mapsto f \]

\[ w \mapsto f \]

\[ r \mapsto f \]

\[ q \mapsto f \text{ and } r \mapsto f \text{ is not possible} \]
\[ \varphi = p \land (\neg p \lor \neg q) \land (q \lor r \lor \neg w) \land (q \lor r \lor w) \]

- \( p \mapsto t \)
- \( q \mapsto f \)
- \( w \mapsto f \)
- \( r \mapsto f \)

- \( q \mapsto f \) and \( r \mapsto f \) is not possible
- learn lemma \( q \lor r \)
\[ \varphi = p \land (\neg p \lor \neg q) \land (q \lor r \lor \neg w) \land (q \lor r \lor w) \land (q \lor r) \]

Tuesday, 13 September 11
$\varphi = p \land (\neg p \lor \neg q) \land (q \lor r \lor \neg w) \land (q \lor r \lor w) \land (q \lor r)$
\[ \varphi = p \land (\neg p \lor \neg q) \land (q \lor r \lor \neg w) \land (q \lor r \lor w) \land (q \lor r) \]

\[ p \mapsto t \]
\[ q \mapsto f \]
\[ r \mapsto f \]
\[ w \mapsto f \]
\[
\varphi = p \land (\neg p \lor \neg q) \land (q \lor r \lor \neg w) \land (q \lor r \lor w) \land (q \lor r)
\]

\[
p \mapsto t
\]
\[
q \mapsto f
\]
\[
r \mapsto f
\]
\[
w \mapsto f
\]
\( \varphi = p \land (\neg p \lor \neg q) \land (q \lor r \lor \neg w) \land (q \lor r \lor w) \land (q \lor r) \)

\[ \begin{align*}
    p &\leftrightarrow t \\
    q &\leftrightarrow f \\
    r &\leftrightarrow f \\
    w &\leftrightarrow f
\end{align*} \]
The CDCL Algorithm
One Line Summaries

BCP and decisions construct an assignment

Learning infers new clauses

Important: CDCL is more than case splitting
Conflict Driven Clause Learning

Interpreting Logic

CDCL is Abstract Interpretation

ACDCL(A)
\[ \varphi = p \land (\neg p \lor \neg q) \land (q \lor r \lor \neg w) \land (q \lor r \lor w) \]
Imagine no assignments, it’s easy if you try

Imagine only Booleans, I wonder if you can
```c
int main(void)
{
    bool p, q, r, w;

    if(p && (!p || q) && (q || r || !w) && (q || r || w))
        assert(0);

    return 0;
}
```
Concrete Interpretation

\[ P = \{ \langle p \mapsto t, q \mapsto t \rangle, \langle p \mapsto t, q \mapsto f \rangle \} \]

\[ Q = \{ \langle p \mapsto t, q \mapsto t \rangle, \langle p \mapsto f, q \mapsto t \rangle \} \]

Shaded: Strongest post-condition for \texttt{assume}(\neg p \lor q)
Satisfiability as Concrete Analysis

\[ C = \langle \varphi(V \rightarrow \mathbb{B}), \subseteq, \cap, \cup \rangle \]
\[ \top = V \rightarrow \mathbb{B} \]
\[ \bot = \emptyset \]
\[ post_\varphi(X) = \{ \varepsilon \in X \mid \varepsilon \text{ satisfies } \varphi \} \]

Concrete domain
All environments
No environment
Strongest post-condition

Concrete Satisfiability:

\[ \varphi \text{ is satisfiable exactly if } post_\varphi(\top) \neq \emptyset \]
Cartesian Abstract Domain

\[ \mathcal{V}(V \rightarrow \mathbb{B}) \]

\[ \alpha \uparrow \gamma \]

\[ V \rightarrow \mathcal{V}(\mathbb{B}) \]

Concrete
Set of environments

Abstract
Environment of sets
Cartesian Abstract Domain

Shaded: Abstract strongest post-condition for \texttt{assume(!p || q)}
Cartesian Abstract Interpretation

\[ C = \langle \varnothing(V \to \mathbb{B}), \subseteq, \cap, \cup \rangle \]
\[ A = \langle V \to \varnothing(\mathbb{B}), \subseteq, \cap, \cup \rangle \]
\[ C \xrightleftharpoons[\alpha]{\gamma} A \]
\[ apost_\varphi = \alpha \circ post_\varphi \circ \gamma \]

Concrete domain

Abstract domain

Galois connection

Best abstract transformer

\[ P = \{ \varepsilon \mid \varepsilon(p) = t \} \]
\[ post_{p \land q}(\overline{P}) = \emptyset \]
\[ post_{p \lor \neg q}(\overline{P}) = \{ \langle p \mapsto f, q \mapsto f \rangle \} \]
\[ post_{p \oplus q}(\top) = \{ \langle p \mapsto f, q \mapsto t \rangle, \langle p \mapsto t, q \mapsto f \rangle \} \]
\[ \alpha(P) = \langle p \mapsto \{ t \}, q \mapsto \mathbb{B} \rangle \]
\[ apost_{p \land q}(\alpha(\overline{P})) = \perp \]
\[ apost_{p \lor \neg q}(\alpha(\overline{P})) = \langle p \mapsto \{ f \}, q \mapsto \{ f \} \rangle \]
\[ apost_{p \oplus q}(\top) = \top \]
Transformers are sound ...

Computing the best abstract transformer is SAT-hard

Use best abstract transformer only for literals

<table>
<thead>
<tr>
<th>conjunction</th>
<th>meet</th>
</tr>
</thead>
<tbody>
<tr>
<td>disjunction</td>
<td>join</td>
</tr>
</tbody>
</table>

If \( \text{apost}_\varphi = \bot \) then \( \varphi \) is unsatisfiable.

(follows from the standard soundness theorem of abstract interpretation)

but they are not complete ...
... but not complete

Abbreviate $\langle p \mapsto \{ t \}, q \mapsto \mathbb{B} \rangle$ as $\langle p \mapsto t \rangle$

$\varphi = p \land (\neg p \lor q)$

$$apost_\varphi(\top) = apost_p(\top) \sqcap (apost_{\neg p}(\top) \sqcup apost_q(\top))$$

$$= \langle p \mapsto t \rangle \sqcap (\langle p \mapsto f \rangle \sqcup \langle q \mapsto t \rangle)$$

$$= \langle p \mapsto t \rangle \sqcap \top$$

$$= \langle p \mapsto t \rangle$$

$$\neq$$

$$post_\varphi(\top) = \{ \langle p \mapsto t, q \mapsto f \rangle \}$$
Recovering Precision

**Theorem** (Cousot and Cousot 1979)

\[
p\text{ost}(\gamma(a)) \subseteq \gamma(\text{gfp}_x(\text{apost}(x \sqcap a))) \subseteq \gamma(\text{apost}(a))
\]

\[
\varphi = p \land (\neg p \lor q)
\]

\[
\text{apost}_\varphi(\top) = \text{apost}_p(\top) \sqcap (\text{apost}_{\neg p}(\top) \cup \text{apost}_q(\top))
\]

\[
= \langle p \mapsto t \rangle
\]

\[
\text{apost}_\varphi(\langle p \mapsto t \rangle) = \text{apost}_p(\langle p \mapsto t \rangle) \sqcap (\text{apost}_{\neg p}(\langle p \mapsto t \rangle) \cup \text{apost}_q(\langle p \mapsto t \rangle))
\]

\[
= \langle p \mapsto t \rangle \sqcap (\bot \cup \langle p \mapsto t, q \mapsto t \rangle)
\]

\[
= \langle p \mapsto t \rangle \sqcap \langle p \mapsto t, q \mapsto t \rangle
\]

\[
= \langle p \mapsto t, q \mapsto t \rangle
\]
Interpreting Logic
One Line Summaries

Satisfying assignments are fixed points of the semantics.

Cartesian abstract interpretation is sound but imprecise.

gfp improves precision in the abstract.
Conflict Driven Clause Learning

Interpreting Logic

CDCL is Abstract Interpretation

ACDCL(A)
A SAT solver and an abstract interpreter walk into a bar

```c
#define l_True (lbool (( uint8_t )0))
#define l_False (lbool (( uint8_t )1))
#define l_Undef (lbool (( uint8_t )2))

class lbool { [...] }

class Solver {
    [...] // FALSE means solver is in a conflicting state
    bool okay () const;
    vec<lbool> assigns; // The current assignments.
    // Enqueue a literal. Assumes value of literal is undefined.
    void uncheckedEnqueue (Lit p, CRef from = CRef_Undef);
    // Perform unit propagation. Return possibly conflicting clause.
    CRef propagate ();
};

MiniSAT 2.2.0
```
Partial assignments

A SAT solver uses partial assignments

\[ V \rightarrow 1_{\text{True}} \quad 1_{\text{False}} \]

\[ \neg \text{okay} \]

An element of the Cartesian abstraction is:

\[ V \rightarrow \{f\} \quad \{t\} \]

Partial assignments are order isomorphic to the reduced Cartesian abstraction
Unit rule

$$\text{unit}(\pi, C) = \begin{cases} \text{conflict} & \text{if } \pi \text{ makes all literals in } C \text{ false} \\ \pi[p \mapsto t] & \text{if } \pi \text{ makes all literals in } C \text{ but } p \text{ false} \\ \pi & \text{otherwise} \end{cases}$$

Unit rule and abstract transformer

$$h(\text{unit}(\pi, C)) = apost_C(h(\pi))$$

The unit rule is the best abstract transformer
BCP

\[ BCP(\varphi, \pi) \{
    \text{repeat}
    \begin{align*}
    \pi' &\leftarrow \pi; \\
    \text{for Clause } C \in \varphi \text{ do } &\pi \leftarrow \text{unit}(C, \pi')
    \end{align*}
    \text{until } \pi' = \pi;
\}
\]

**Theorem:** BCP as fixed point

\[ h(BCP(\varphi, \pi)) = \text{gfp}_x (\text{apost}_\varphi (h(\pi) \cap x)) \]

**BCP is a greatest fixed point**
A SAT solver and an abstract interpreter walk into a bar

```c
#define l_True (lbool((uint8_t)0))
#define l_False (lbool((uint8_t)1))
#define l_Undef (lbool((uint8_t)2))

class lbool { [...] }

class Solver {
    [...] // FALSE means solver is in a conflicting state
    bool okay () const;
    vec<lbool> assigns; // The current assignments.
    // Enqueue a literal. Assumes value of literal is undefined.
    void uncheckedEnqueue (Lit p, CRef from = CRef_Undef);
    // Perform unit propagation. Return possibly conflicting clause.
    CRef propagate () ;
};

MiniSAT 2.2.0
```
Another learning example

\[\neg 1 \land (1 \lor \neg 2 \lor \neg 3) \land (\neg 4 \lor 5) \land (\neg 6 \lor 7) \land (\neg 6 \lor \neg 8) \land (\neg 7 \lor 8 \lor \neg 9) \land (3 \lor 9 \lor 1)\]
Another learning example

\neg 1 \land (1 \lor \neg 2 \lor \neg 3) \land (\neg 4 \lor 5) \land (\neg 6 \lor 7) \land (\neg 6 \lor \neg 8) \land (\neg 7 \lor 8 \lor \neg 9) \land (3 \lor 9 \lor 1)

DL0

\overline{1}

DL1

2

\overline{3}
Another learning example

\[ \neg 1 \land (1 \lor \neg 2 \lor \neg 3) \land (\neg 4 \lor 5) \land (\neg 6 \lor 7) \land (\neg 6 \lor \neg 8) \land (\neg 7 \lor 8 \lor \neg 9) \land (3 \lor 9 \lor 1) \]

Every cut that disconnects the roots from the error is a reason
Another learning example

\[-\ell_1 \land (1 \lor \ell_2 \lor \ell_3) \land (\ell_4 \lor 5) \land (\ell_6 \lor 7) \land (\ell_6 \lor \ell_8) \land (\ell_7 \lor 8 \lor \ell_9) \land (3 \lor 9 \lor 1)\]

Every cut that disconnects the roots from the error is a reason for a conflict using an implication graph.
Another learning example

\[ \neg 1 \land (1 \lor \neg 2 \lor \neg 3) \land (\neg 4 \lor 5) \land (\neg 6 \lor 7) \land (\neg 6 \lor \neg 8) \land (\neg 7 \lor 8 \lor \neg 9) \land (3 \lor 9 \lor 1) \]
Another learning example

\[ \neg 1 \land (1 \lor \neg 2 \lor \neg 3) \land (\neg 4 \lor 5) \land (\neg 6 \lor 7) \land (\neg 6 \lor \neg 8) \land (\neg 7 \lor 8 \lor \neg 9) \land (3 \lor 9 \lor 1) \]
Another learning example

\[ \neg 1 \land (1 \lor \neg 2 \lor \neg 3) \land (\neg 4 \lor 5) \land (\neg 6 \lor 7) \land (\neg 6 \lor \neg 8) \land (\neg 7 \lor 8 \lor \neg 9) \land (3 \lor 9 \lor 1) \]

Cuts = Heuristic underapproximation of the weakest precondition
Another learning example

\[ \neg 1 \land (1 \lor \neg 2 \lor \neg 3) \land (\neg 4 \lor 5) \land (\neg 6 \lor 7) \land (\neg 6 \lor \neg 8) \land (\neg 7 \lor 8 \lor \neg 9) \land (3 \lor 9 \lor 1) \land (9 \lor 3) \]
Trace Partitioning
(Mauborgne and Rival, 2005)

Domain of Intervals:
\[ V \rightarrow \mathbb{Z} \times \mathbb{Z} \]

Transform program

Analysis too imprecise

Same analysis is precise

Changing the equation allows one to prove more with the same analysis.

Instance of a power domain (Cousot and Cousot, 1979)
Learning in SAT

Decisions and learning are dynamic “trace” partitioning

Transform formula

Conflict reason

Safe

Learned clause
Learning in SAT

Decisions and learning are dynamic “trace” partitioning
CDCL is Abstract Interpretation
One Line Summaries

CDCL implements the Cartesian abstract domain as its main data structure.

The unit rule is the application of the best abstract clause transformer.

BCP is fixed point computation.

Decisions & Learning are discovery of trace partitions.
CDCL is Abstract Interpretation
Summary of Summaries

CDCL = Partial assignments
    + Unit rule & BCP
    + Decisions & Learning
= Cartesian abstract domain
    + Abstract transformer & GFP
    + Trace partitioning

Not an ANALOGY but an ISOMORPHISM

Precise results using a strict abstraction!
Conflict Driven Clause Learning

Interpreting Logic

CDCL is Abstract Interpretation

ACDCL(A)
What about programs?

DL0

[Diagram of a program flow with conditions and assignments]

What about programs?
What about programs?

Abstract Implication Graph

DL0

\[ c_1 : a \leq -2 \]

\[ c_2 : a \leq -1 \]

\[ c_3 : a = 0 \]

\[ c_4 : a \geq 1 \]

\[ c_2 : a \geq -1 \]

\[ c_3 : a \geq 0 \]

\[ n_1 \]

\[ [a \leq -2] \]

\[ [a = -1] \]

\[ [a = 0] \]

\[ n_2 \]

\[ b := -1 \]

\[ b := 1 \]

\[ b := 2 \]

\[ b := -2 \]

\[ \not{[b = 0]} \]
What about programs?

DL0

\[
\begin{align*}
\text{c}_1 &: a \leq -2 \\
\text{c}_2 &: a \leq -1 \\
\text{c}_3 &: a \leq 0 \\
\text{c}_4 &: a \geq 1 \\
n_2 &: b \leq 2 \\
n_2 &: b \geq -2 \\
\end{align*}
\]

\[
\begin{align*}
\frac{1}{2} &: b \leq 0 \\
\frac{1}{2} &: b \geq 0 \\
n_2 
\end{align*}
\]

\[
\begin{align*}
[a = -1] \\
[a = 0] \\
[a = 2] \\
[a = 1] \\
n_1 \quad n_2 \quad n_3 \quad n_4 \\
b := -1 \\
b := 1 \\
b := 2 \\
b := -2 \\
[b = 0] \\
\end{align*}
\]
What about programs?

DL0

- c₂: a ≤ −1
- c₃: a ≤ 0
- c₃: a ≥ 0
- c₂: a ≥ −1
- c₁: a ≤ −2

- n₂: b ≤ 2
- n₂: b ≥ −2

- b: b ≤ 0
- b: b ≥ 0

DL1

- n₁: a ≤ −42

DPLL is Abstract Interpretation

SAFE

maximal wp-underapproximation transformer

¬ (n₂: b ≤ 2)∧(n₂: b ≥ −2)∧b = 0

find cut

Generalise!
What about programs?

DL0

\[ c_2 : a \leq -1 \quad c_3 : a \leq 0 \quad c_3 : a \geq 0 \quad c_2 : a \geq -1 \]

\[ c_1 : a \leq -2 \]

\[ n_2 : b \leq 2 \]

\[ n_2 : b \geq -2 \]

\[ \neg b \leq 0 \]

\[ \neg b \geq 0 \]

DL1

\[ n_1 : a \leq -42 \]

\[ c_1 : a \leq -42 \]

\[ c_2 : \bot \]

\[ c_3 : \bot \]

\[ c_4 : \bot \]

\[ n_2 : b \geq 2 \]

\[ \neg b \geq 0 \]

\[ \neg \]

SAFE
What about programs?

Heuristic underapproximation of the weakest pre-condition

SAFE  →  Generalise!
What about programs?

![Diagram showing DL0 and DL1 with conditions and implications]
What about programs?

DL0:

- $c_2 : a \leq -1$
- $c_3 : a \leq 0$
- $c_3 : a \geq 0$
- $c_2 : a \geq -1$
- $c_1 : a \leq -2$

Graph:

- $n_2 : b \leq 2$
- $n_2 : b \geq -2$
- $\bot : b \leq 0$
- $\top : b \geq 0$
- $n_2 : b \leq -1$

SAFE
ACDCL(A)
One Line Summaries

ACDCL(A) program analysers!

Techniques from SAT translate to programs

ACDCL(A) discovers small, property driven refinement
Something more practical

ACDCL(Interval) procedure over floating point and machine integer intervals

Automatically finds property-dependent partitioning

Example: Taylor expansion of sine-function

```c
int main()
{
    float IN;
    __CPROVER_assume(IN > -HALFPI && IN < HALFPI);
    float x = IN;
    float result = x - (x**x)/6.0f + (x**x**x**x)/120.0f + (x**x**x**x**x**x**x)/5040.0f;
    assert(result <= VAL && result >= -VAL);
    return 0;
}
```
Number of partitions vs. tightness of bound

Result $\leq 2.0$

Result $\geq -2.0$
Number of partitions vs. tightness of bound

\[
\begin{align*}
\text{result} & \leq 1.5 \\
-\frac{\pi}{2} & \quad \text{--} \quad \frac{\pi}{2} \\
\text{result} & \geq -1.5
\end{align*}
\]
Number of partitions vs. tightness of bound

\[ \text{result} \leq 1.2 \]

\[ \text{result} \geq -1.2 \]
Number of partitions vs. tightness of bound

result $\leq 1.1$

result $\geq -1.1$
Number of partitions vs. tightness of bound

result ≤ 1.01

result ≥ -1.01
Number of partitions vs. tightness of bound

Precise results using a strict abstraction!
Orders of magnitude faster than propositional SAT
Conclusion

SAT solvers are abstract interpreters

<table>
<thead>
<tr>
<th>partial assignments</th>
<th>Cartesian domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>unit rule</td>
<td>abstr. transformer</td>
</tr>
<tr>
<td>BCP</td>
<td>gfp</td>
</tr>
<tr>
<td>decisions</td>
<td>meet irreducibles</td>
</tr>
<tr>
<td>learning</td>
<td>trace partitioning</td>
</tr>
</tbody>
</table>

Abstract interpreters can be SAT solvers

ACDCL(A) for program analysis / SMT
precise results in an imprecise abstraction
... walk into a bar

AI looks toward SMT

post operators, widenings

SMT looks towards AI

Precise analysis, efficient handling of disjunction
Invited questions

Isn’t this just CEGAR? What if case splits are not enough?

Show me experiments!