The *Eos* SMT/SMA-Solver: A Preliminary Report [1]

Giulio Mazzi

Università Degli Studi di Verona

Lisbon, 7th July 2019

[1] Joint work with Maria Paola Bonacina

Conflict-driven Reasoning

Conflict-Driven SATisfiability[2]

- **CDCL**: propositional conflict-driven reasoning
- **DPLL($\mathcal{T}$)**: CDCL + black-box theories $\rightarrow$ conflict-driven reasoning: only propositional
- **MCSAT**: lifts CDCL to SMT for one theory $\rightarrow$ not a combination calculus
- **CDSAT**: generalizes MCSAT to generic combination of disjoint theories

[2] [Bonacina, Graham-Lengrand, Shankar, CADE2017, JAR2019]

The CDSAT trail

- Sequence of **assignments** (variable/value pairs)
- Either **decisions** (Boolean or first-order) or **justified assignments**
- **SMT**: only Boolean input (as assignments with empty justification)
- **SMA**: Boolean and first-order assignments as input
- Each assignment has a **level**, not necessarily in increasing order ($\neq$ CDCL)

Example of trail

Example

A trail with two input formulas, a first-order decision and a Boolean propagation

$$\underbrace{{}_{\{\}\vdash} y < 0, {}_{\{\}\vdash}\, x + y > 0}_{\text{lv. } 0}, \underbrace{{}_{?} x \leftarrow 0}_{\text{lv. } 1}, \underbrace{{}_{\{y<0, x+y>0\}\vdash} x > 0}_{\text{lv.}0}, \ldots$$

- $y < 0$, $x + y > 0$ are input formulas (empty justification)
- $x > 0$ is propagated at level 0. Since it is lower than the highest level this is called a *late propagation*
- $x > 0$ is not an input term. These non-trivial inferences are only to explain a conflict

Overview of *Eos*

- Written in C++
- Implements CDSAT as the central class
- Extensible: defines a *theory module* class that gets instantiated for each theory module
- Three theory modules already implemented:
  - SAT $\rightarrow$ Propositional logic
  - LRA $\rightarrow$ Linear Real Arithmetic
  - UF $\rightarrow$ Uninterpreted Functions
- All three *quantifier-free*
- `QF_UF`, `QF_LRA` and `QF_UFLRA` in SMT-LIB

The CDSAT trail in *Eos*

- Every non-input justified assignment stores the ID of the responsible module
- The justification can be built lazily from this ID on demand
- This is crucial for fast propagation (both Boolean and theory)

Example

Given the trail:

$$a \vee (x + y > 0),\ {}_{?}x \leftarrow 1,\ {}_{?}y \leftarrow 2,\ \underbrace{{}_{\{x \leftarrow 1, y \leftarrow 2\} \vdash}}_{ID_{LRA}}(x + y > 0)$$

The CDSAT transition system in *Eos*

Two main functions:

- `check_sat`: implements the search for a model of the input problem, covering the trail rules Deduce, Decide, Fail, and ConflictSolve
- `conflict_analysis`: implements the conflict-state rules Resolve, Backjump, UndoClear, and UndoDecide

Use of the Deduce rule

- **Propagation**: trivial inferences (e.g. BCP in CDCL). In *Eos* this is applied exhaustively in the `propagate()` function
- **Conflict explanation**: non-trivial inferences (e.g. resolution in CDCL)

Propagation

```
function check_sat
    loop
        propagate( )                                          ▷ rule Deduce
        if conflict then           ▷ the propagation has generated a conflict
            if conflict at level zero then
                return unsatisfiable                            ▷ rule Fail
            else
                conflict_analysis( )                   ▷ rule ConflictSolve
        else                        ▷ everything was propagated without conflict
            if decision order is empty then   ▷ every term has a value assigned?
                return satisfiable                                   ▷ SAT
            else
                make_decision( )                               ▷ rule Decide
```

propagate() example

Example

Given the trail:

$$\underbrace{\ldots,\ (x<0)\vee(y<0),\ \ \ldots,}_{\text{lv. } 0}\ \ \underbrace{{}_{?}x \leftarrow 1,}_{\text{lv. } 1}$$

LRA can deduce that $x < 0$ is false:

$$\ldots,\ \underbrace{{}_{\{x\leftarrow 1\}\vdash}\neg(x<0),}_{\text{lv. } 1}$$

SAT can deduce that $y < 0$ is true:

$$\ldots,\ \underbrace{{}_{\{\neg(x<0),(x<0)\vee(y<0)\}\vdash}(y<0)}_{\text{lv. } 1}$$

Decisions

- If no more trivial inferences are possible, a decision must be made
- *Eos* selects a term for a decision, and it asks the appropriate theory module to assign an *acceptable* value to the term.
  - SAT module $\rightarrow$ Boolean terms
  - LRA module $\rightarrow$ Real terms
  - UF module $\rightarrow$ terms of uninterpreted sort

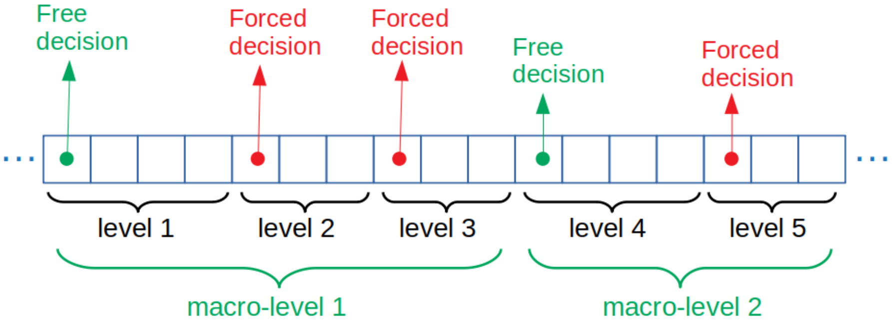
Example

if $y > 3$ is true an *acceptable* value for y must be greater than 3

Decision Order

- The selection of terms for decisions is based on a generalization of the *VSIDS* heuristic to handle both Boolean and first-order terms
- *Eos* increases the activity of both Boolean and first-order terms during conflict analysis
- A theory module can request a higher priority for a first-order term that has a single acceptable value (*forced decision*)

Collecting levels in *macrolevels*



- *Eos* makes forced decisions as soon as possible
- A *free decision* (i.e. a non-forced decision) open a new *macrolevel*: it collects a free decision and their related forced decisions
- Macrolevels are useful in heuristics

Decision and satisfiability

A decision is made only if at least a term has no value. If everything is already assigned without any conflict, the problem is satisfiable

```
function check_sat
    loop
        propagate( )                              ▷ rule Deduce
        if conflict then                          ▷ the propagation has generated a conflict
            if conflict at level zero then
                return unsatisfiable              ▷ rule Fail
            else
                conflict_analysis( )              ▷ rule ConflictSolve
        else                                      ▷ everything was propagated without conflict
            if decision order is empty then       ▷ every term has a value assigned?
                return satisfiable                ▷ SAT
            else
                make_decision( )                  ▷ rule Decide
```

Conflicts during `propagate()`

- Theory modules can find conflicts during `propagation`
- If a conflict is at level zero, the problem is unsatisfiable
- Otherwise `conflict_analysis()` takes care of the conflict

```
function check_sat
    loop
        propagate( )                                            ▷ rule Deduce
        if conflict then              ▷ the propagation has generated a conflict
            if conflict at level zero then
                return unsatisfiable                              ▷ rule Fail
            else
                conflict_analysis( )                     ▷ rule ConflictSolve
        else                          ▷ everything was propagated without conflict
            if decision order is empty then   ▷ every term has a value assigned?
                return satisfiable                                     ▷ SAT
            else
                make_decision( )                                ▷ rule Decide
```

Conflict example

Late Propagation

This trail is in conflict

$$\underbrace{y < 0,\ x + y > 1}_{\text{lv. } 0},\ \underbrace{{}_{?}x \leftarrow 0}_{\text{lv.}1},\ \underbrace{{}_{\{y<0, x+y>1\}\vdash} x > 1}_{\text{lv. } 0}$$

Arithmetic conflict

$$\text{conflict: } [\ \underbrace{x \leftarrow 0}_{\text{lv. } 1}, \underbrace{x > 1}_{\text{lv. } 0}\ ]$$

The level of the conflict is 1

Conflict Analysis

```
procedure conflict_analysis
    conflict ← get_reason()                                    ▷ get the reason of the conflict
    conflict_level ← get_max_level(conflict)                   ▷ higher level of conflict values
    backjump(conflict_level)                                   ▷ undo everything after the conflict
    while conflict has two or more terms at conflict_level do
        last ← pop_from_trail( )                               ▷ get the last Boolean propagation on the trail
        if last.level() = conflict_level and last is in conflict then    ▷ rule Resolve
            conflict.remove(last)                              ▷ resolve this value with the conflict
            ▷ get the justification of this propagation
            justification ← get_justification(last)
            for all  Term just in justification do
                ▷ is this propagation justified by a first order decision at the conflict level?
                if just is non-Boolean and at conflict level then
                    new_value ← ¬ trail.get_value(last)        ▷ flip the value of the propagation
                    backjump_one_level( )                      ▷ rule UndoDecide: undo
                    add_decision(last,new_value)               ▷ rule UndoDecide: decide
                    return
                else
                    conflict.add(just)                         ▷ add just to the conflict
    ▷ here, the conflict has a single term assigned at the level of the conflict
    topmost_var ← get_outstanding(conflict)
    if topmost_var is non-Boolean then
        backjump_one_level( )                                  ▷ rule UndoClear
        return
    clause ← create_clause(conflict)                           ▷ learn a new clause
    bt_level ← compute_backjump_level(conflict)
    backjump(bt_level)                                         ▷ rule Backjump
    learn_new_clause(clause)
```

Conflict analysis - Preliminaries

The procedure retrieves the conflict terms and computer the highest level among the assignments in conflict. Every level higher than the level of the conflict can be immediately pruned.

procedure `conflict_analysis`
    conflict ← get_reason()   ▷ get the reason of the conflict
    conflict_level ← `get_max_level`(*conflict*)   ▷ higher level of conflict values
    `backjump`(*conflict_level*)   ▷ undo everything after the conflict
    while *conflict* has two or more terms at *conflict_level* **do**
        ...
    ▷ here, the conflict has a single term assigned at the level of the conflict
    ...

Propositional resolution

Resolve is applied until a Backjump, UndoClear, or UndoDecide is possible

```
procedure conflict_analysis
    ...
    while conflict has two or more terms at conflict_level do
        last ← pop_from_trail( )              ▷ get the last Boolean propagation on the trail
        if last.level() = conflict_level and last is in conflict then        ▷ rule Resolve
            conflict.remove(last)             ▷ resolve this value with the conflict
            ▷ get the justification of this propagation
            justification ← get_justification(last)
            for all Term just in justification do
                ▷ is this propagation justified by a first order decision at the conflict level?
                if just is non-Boolean and at conflict level then
                    ...
                else
                    conflict.add(just)        ▷ add just to the conflict
    ...
```

UndoClear

- The first-order decision at conflict level was acceptable, now it is in conflict → a late propagation explains why this decision is no longer acceptable
- it is useful to compare the macrolevel

```
procedure conflict_analysis
    ...
    while conflict has two or more terms at conflict_level do
        ...
    ▷ here, the conflict has a single term assigned at the level of the conflict
    topmost_var ← get_outstanding(conflict)
    if topmost_var is non-Boolean then
        backjump_one_level( )                         ▷ rule UndoClear
        return
    clause ← create_clause(conflict)                  ▷ learn a new clause
    bt_level ← compute_backjump_level(conflict)
    backjump(bt_level)                                ▷ rule Backjump
    learn_new_clause(clause)
```

Backjump

- A backjump to the second highest level in the conflict
- CDSAT can learn lemmas using the LearnBackjump extension → *Eos learns only purely Boolean conflicts!*

```
procedure conflict_analysis
    ...
    while conflict has two or more terms at conflict_level do
        ...
    ▷ here, the conflict has a single term assigned at the level of the conflict
    topmost_var ← get_outstanding(conflict)
    if topmost_var is non-Boolean then
        backjump_one_level( )                              ▷ rule UndoClear
        return
    clause ← create_clause(conflict)                       ▷ learn a new clause
    bt_level ← compute_backjump_level(conflict)
    backjump(bt_level)                                     ▷ rule Backjump
    learn_new_clause(clause)
```

UndoDecide

```
procedure conflict_analysis
    ...
    while conflict has two or more terms at conflict_level do
        last ← pop_from_trail( )        ▷ get the last Boolean propagation on the trail
        if last.level() = conflict_level and last is in conflict then        ▷ rule Resolve
            ...
            for all Term just in justification do
                ▷ is this propagation justified by a first order decision at the conflict level?
                if just is non-Boolean and at conflict level then
                    new_value ← ¬ trail.get_value(last)        ▷ flip the propagated value
                    backjump_one_level( )        ▷ rule UndoDecide: undo
                    add_decision(last,new_value)        ▷ rule UndoDecide: decide
                    return
                else
                    ...
```

SAT module - MiniSAT inspired

- Equisatisfiable CNF problem: Tseitin transformation
- Main focus: very efficient *Boolean Clausal Propagation* (BCP)
- *Two watched literals scheme*
- A specialized memory manager is used to store the clauses in a compact area of memory

Example

$$(a \lor b \lor c),\ \ {}_{?}\neg a,\ \ {}_{?}\neg b,\ \ {}_{(a \lor b \lor c), \neg a, \neg b} \vdash c$$

implied literal: c, unit clause: $a \lor b \lor c$

UF and LRA - Overview

- Inspired by the implementation of MCSAT (CVC4 version)
- The UF module handles equalities and inequalities between terms of uninterpreted sorts and checks the congruence axioms hold
- The LRA module handles linear constraints for real variables

Generalization of the Two watched literals scheme

Example

The constraint $2x + y - z > 0$ can be seen as a generalized clause:

$$\{\ x,\ y,\ z,\ 2x + y - z > 0\ \}$$

- If x, y, and z are assigned the truth value of $2x + y - z > 0$ can be propagated
- If x, y, and $2x + y - z > 0$ are assigned the acceptable values for z changes
- If everything is assigned the module check that this is a consistent assignment, otherwise the module is in error

UF - Conflicts

- The equality propagation can identify transitivity conflicts
- *Eos* can build new terms to explain these conflicts

Transitivity Conflict

The trail:

$$a \simeq b,\ a \simeq c,\ {}_?b \leftarrow \mathfrak{q}_1,\ {}_?c \leftarrow \mathfrak{q}_2$$

is in conflict. The UF module creates and propagates the term $b \simeq c$ and builds the conflict as:

$$[a \simeq b,\ a \simeq c,\ \neg(b \simeq c)]$$

UF - Uninterpreted Functions

- The UF module also checks that the congruence axioms hold
- The arguments of function applications are *watched*, when all the arguments are assigned the module checks that the congruence axiom is respected

Congruence Conflict

Given function $f : \texttt{Real} \to \texttt{Bool}$, the trail

$$x \leftarrow 5,\ f(x),\ \neg f(y),\ y \leftarrow 5 \tag{1}$$

is in conflict, builds the conflict

$$[x \simeq y,\ \neg(f(x) \simeq f(y))] \tag{2}$$

where $x \simeq y$ and $(f(x) \simeq f(y))$ may be new terms

LRA - Fourier-Motzkin

- The module keeps a set of acceptable values for every real term: lower bound, un upper bound, and list of equalities and disequalities
- If a real term has no acceptable values the module find a conflict

FM Conflict

The trail

$$y < x,\ x < 0,\ {}_?y \leftarrow 1$$

is in conflict. The module makes a non-trivial inference by FM-resolution:

$$y < x,\ x < 0 \vdash y < 0$$

and builds the conflict:

$$[y \leftarrow 1, y < 0]$$

LRA - Disequality Elimination

- Another rule is required to handle a special case
- If a real symbol has the same upper and lower bound but this bound is not an acceptable value the disequality elimination rule applies

$$t_1 \leq x,\ x \leq t_2,\ t_1 = t_0,\ t_2 = t_0,\ x \neq t_0\ \vdash \bot$$

Disequality Conflict

The trail

$$x - y \geq 0,\ x \leq 0,\ \neg(x = z),\ {}_?z \leftarrow 0,\ {}_?y \leftarrow 0$$

is in conflict. The symbol x should be ≤ 0, ≥ 0 and $\neq 0$
The conflict is now:

$$[x - y \geq 0,\ x \leq 0,\ y = z,\ 0 = z,\ \neg(x = z)]$$

Fun Fact: Who is *Eos*?

In Greek mythology, Eos is a Titaness and the goddess of the dawn, who rose each morning from her home at the edge of the Oceanus.

Inferences in *Eos* vs MCSAT

Eos

$$\underbrace{y < 0,\ x + y > 1}_{\text{lv. } 0},\ \underbrace{{}_{?}x \leftarrow 0}_{\text{lv.}1},\ \underbrace{{}_{\{y<0,x+y>1\}\vdash} x > 1}_{\text{lv. } 0}$$

conflict: $[\ \underbrace{x \leftarrow 0}_{\text{lv. } 1}, \underbrace{x > 1}_{\text{lv. } 0}\ ] \leftarrow$ The level of the conflict is 1

MCSAT

This trail is in conflict

$$\underbrace{y < 0,\ x + y > 1}_{\text{lv. } 0},\ \underbrace{{}_{?}x \leftarrow 0}_{\text{lv.}1},\ \underbrace{{}_{\{x\leftarrow 0\}\vdash} \neg(x > 1)}_{\text{lv. } 1}$$

conflict: $[\ \underbrace{y < 0}_{\text{lv. } 0}, \underbrace{x + y > 1}_{\text{lv. } 0}, \underbrace{\neg(x > 1)}_{\text{lv. } 1}\ ] \leftarrow$ The level of the conflict is 1