SMT Solving Fundamentals

Nikolaj Bjørner, Microsoft Research, RiSE
TU Wien, 2025

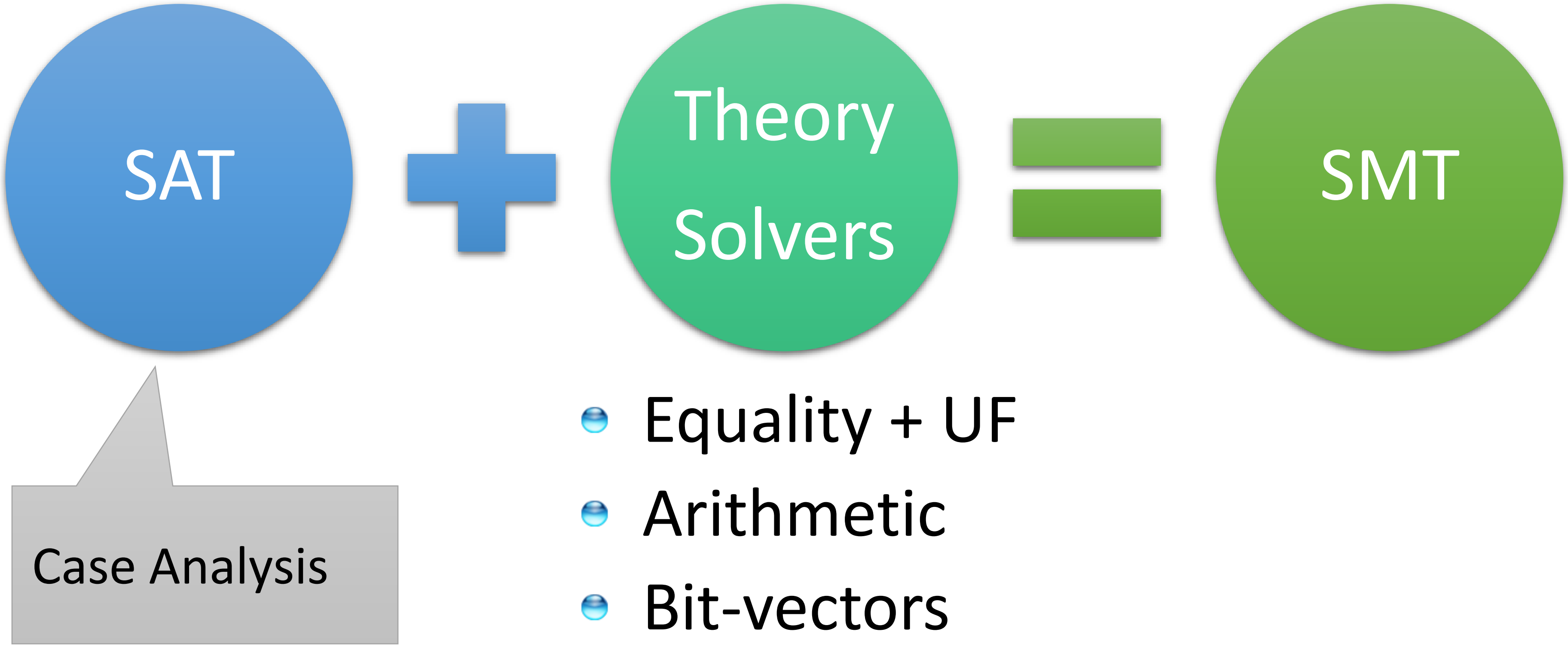
A Laura Kovacs guest lecture production

Satisfiability Modulo Theories (SMT)

Is formula φ satisfiable modulo theory *T* ?

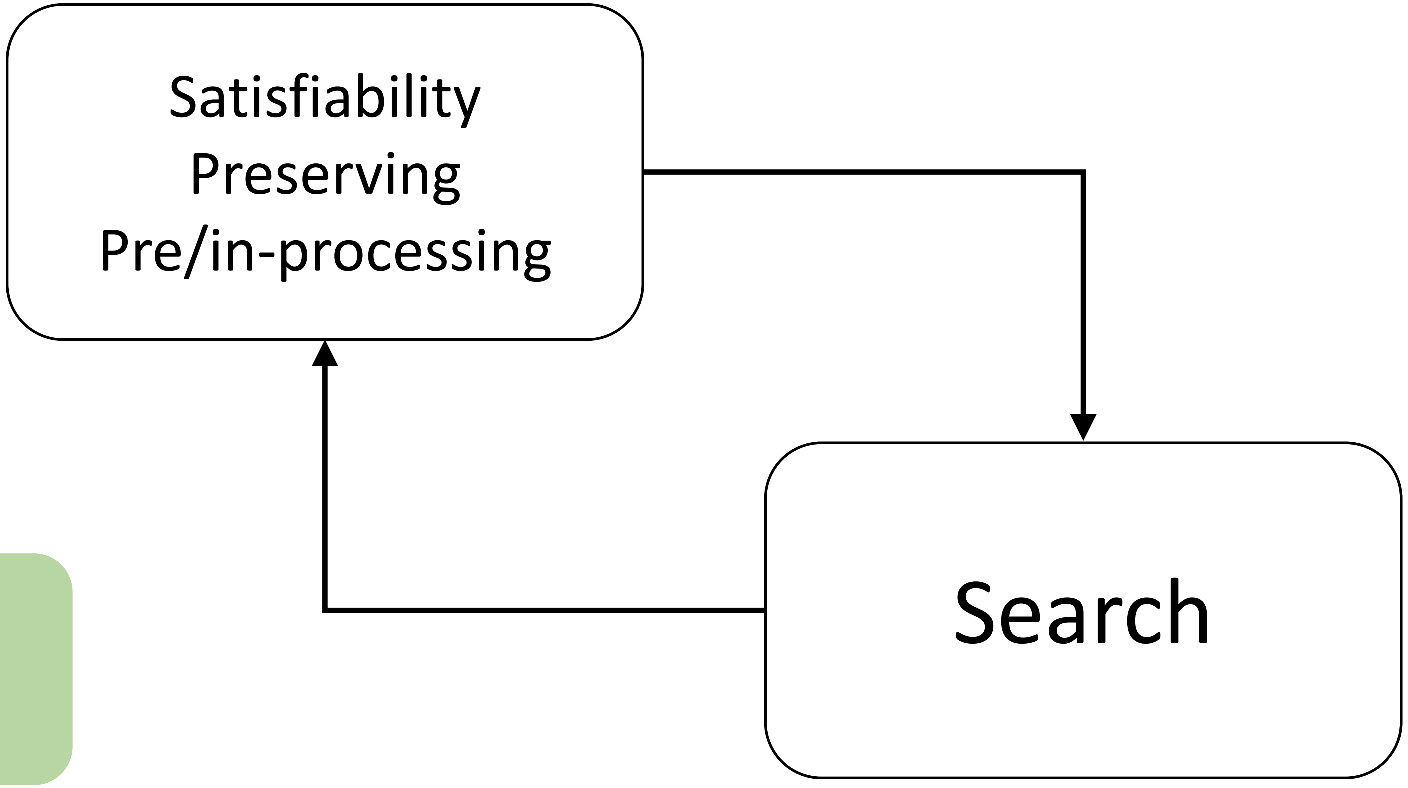
SMT solvers have specialized algorithms for *T*

CDCL(T)



- Equality + UF
- Arithmetic
- Bit-vectors
- …

Elements of Solving

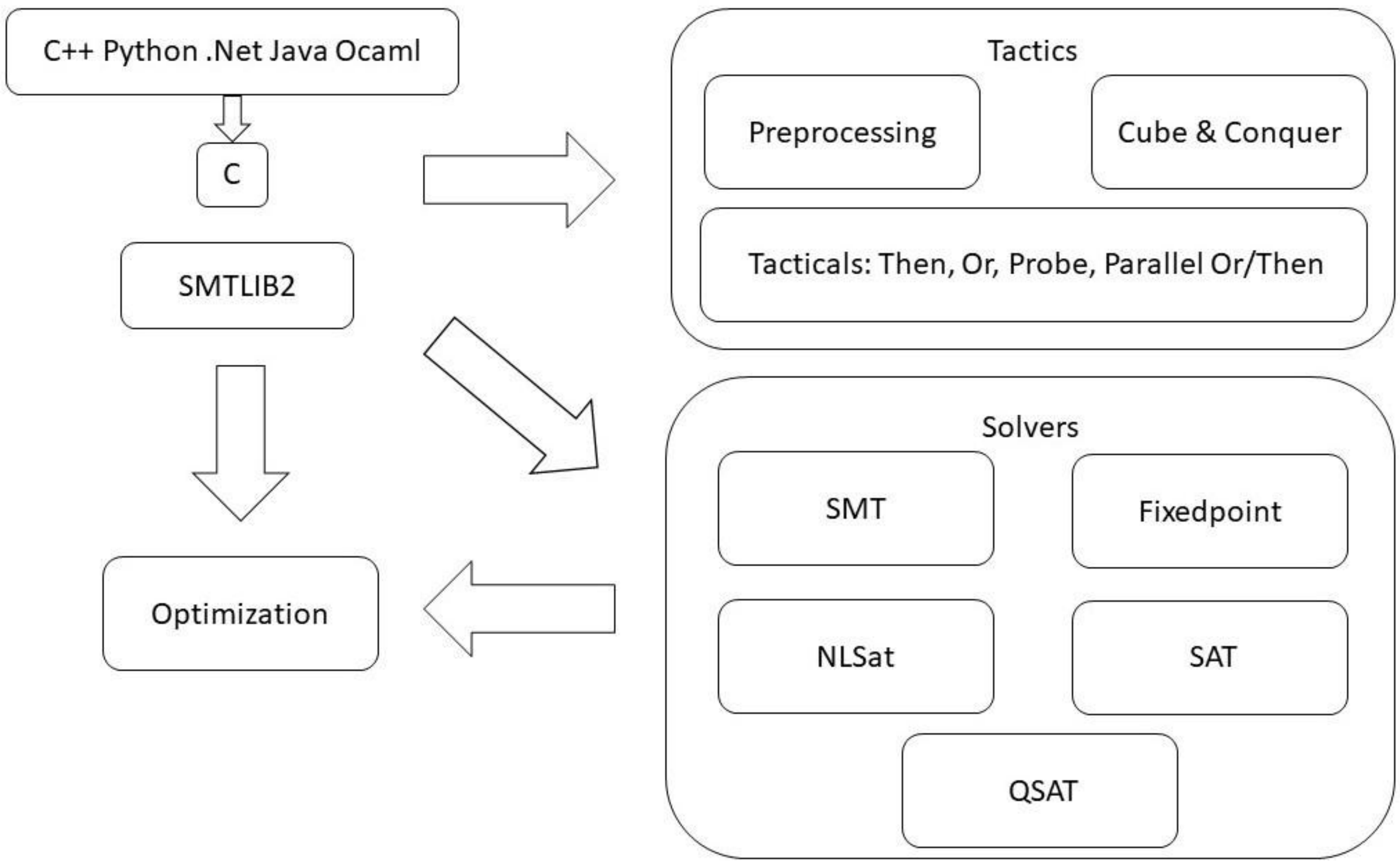


Encoding and re-encoding can also be considered an element of solving loop.

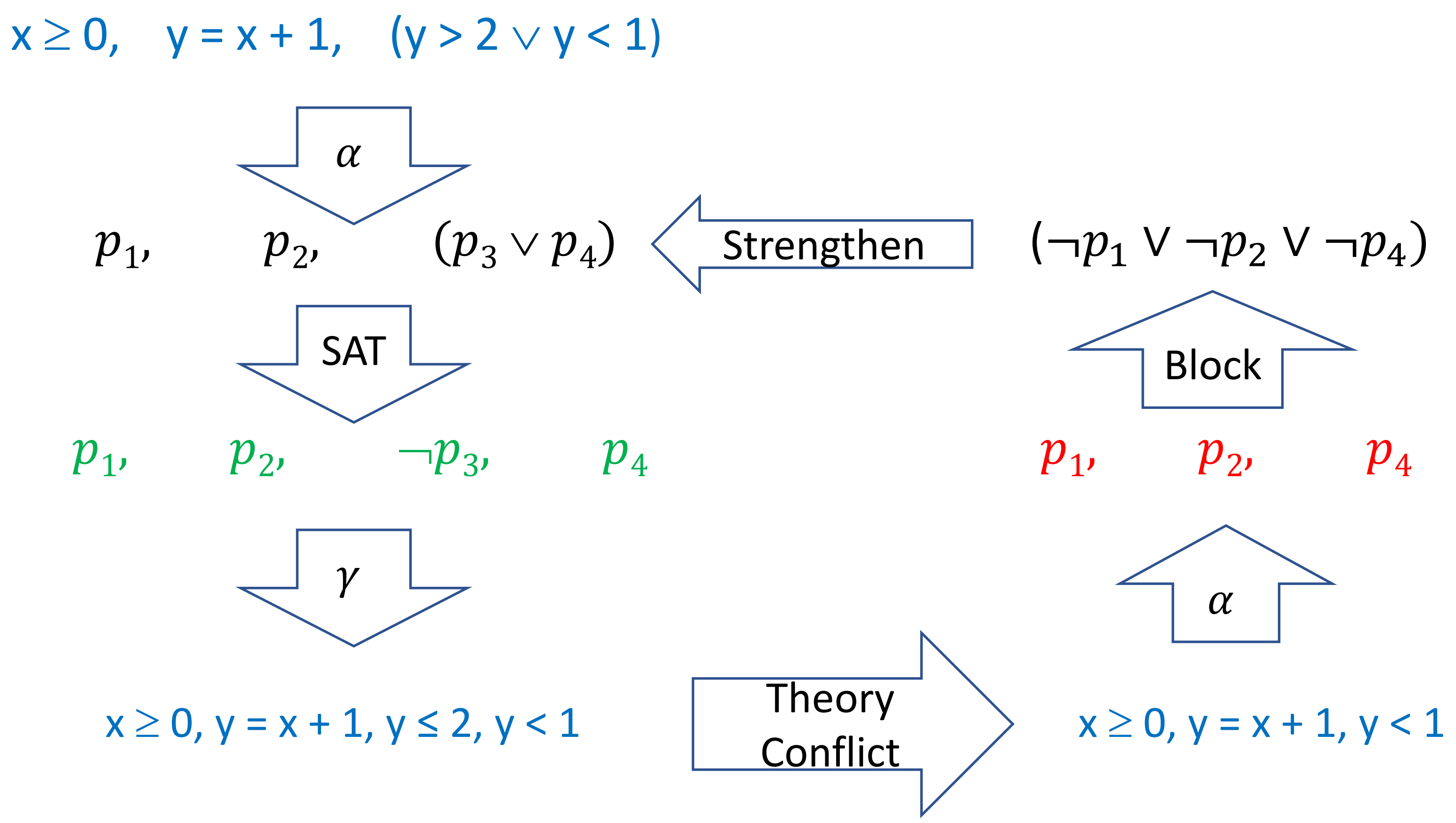
Search Engines

CDCL(T) SPACER NLSAT QSAT

Z3 overview


C++ Python .Net Java Ocaml
C
SMTLIB2
Optimization
Tactics
Preprocessing
Cube & Conquer
Tacticals: Then, Or, Probe, Parallel Or/Then
Solvers
SMT
Fixedpoint
NLSat
SAT
QSAT

CDCL(T)


$x \geq 0, \quad y = x + 1, \quad (y > 2 \vee y < 1)$
α
$p_1, \quad p_2, \quad (p_3 \vee p_4)$
Strengthen
$(\neg p_1 \vee \neg p_2 \vee \neg p_4)$
SAT
Block
$p_1, \quad p_2, \quad \neg p_3, \quad p_4$
$p_1, \quad p_2, \quad p_4$
γ
α
$x \geq 0, y = x + 1, y \leq 2, y < 1$
Theory Conflict
$x \geq 0, y = x + 1, y < 1$

CDCL(T) – Main State Variables

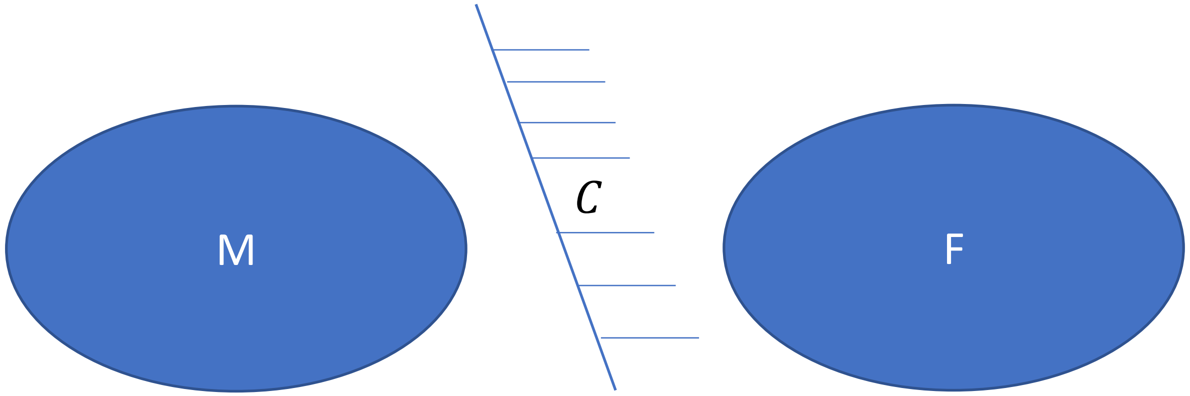
Search State $\langle M; F \rangle$

$\langle M; F; C \rangle$ Conflict State

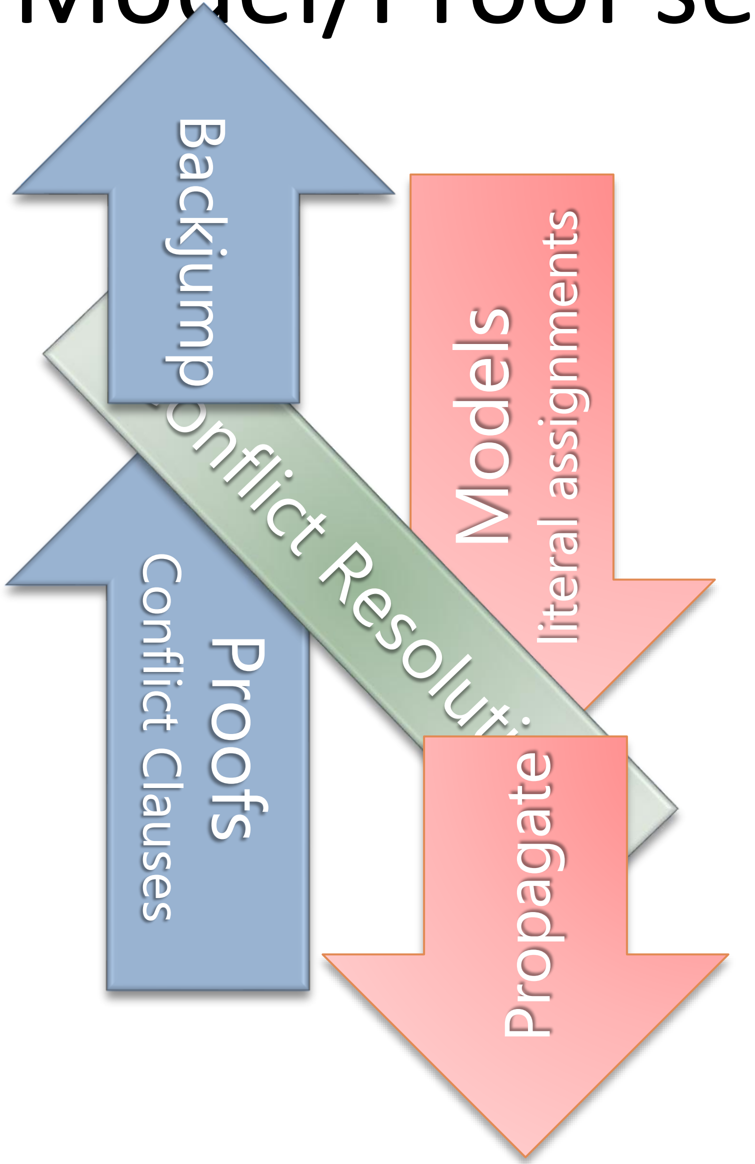
- *F* – set of clauses
  - Split into *irredundant* and *redundant* clauses.
  - Redundant clauses can be garbage collected.
  - 2 literal watch list and binary clause optimization
- *M* – a trail of assigned literals
- *C* – a conflict clause

CDCL(T) - Invariants

- The conflict clause C is false in M and a consequence of F. Thus, for state $\langle M; F; C \rangle$ we have $F \models_T C$ as well as $\overline{C} \in M$.
- A propagated literal is justified by the current partial model M. Thus, for state $\langle M, \ell^C; F \rangle$ we have $F \models_T C, \ell \in C$, and for each $\ell' \in C \setminus \{\ell\} : \overline{\ell}' \in M$.



CDCL(T) – Dual Model/Proof search



Dichotomy – Proofs and Models

Farkas Lemma

1. There is an x such that: $Ax = b \wedge x \geq 0$

2. There is a y such that: $yA \geq 0 \wedge yb < 0$

For every matrix A, vector b it is the case that either (1) or (2) holds (and not both).

From DPLL to CDCL

1. There is $M' \supseteq M$ such that $M' \vDash F$

2. There is $M' \subseteq M$ and proof Π such that $F \vdash_{\Pi} \overline{M'}$

Given M can it be extended to M' to satisfy (1)?
If not, find subset M' to establish (2).
(that is inconsistent with *F*)

Corollary

Conflict learning (resolution) extends *F* by clauses that block shorter models

If $M \vdash \neg F$ then
- $\overline{C, \ell} \subseteq M$ for some $F \vdash C \vee \ell$ (or F contains $\emptyset$)
- for every D, where
  - $\overline{D}, \bar{C} \subseteq M' \subseteq M$,
  - $M' \vdash (D \vee \neg \ell)$

  it is not possible to extend M' to satisfy F

CDCL(T) as inference rules

Sat	$\langle M;\ F\,\rangle$	$\Rightarrow SAT$	$SAT\ =\ Theory(M,\ F)$
Conflict	$\langle M;\ F\,\rangle$	$\Rightarrow \langle M;\ F;\ C\,\rangle$	$C\ =\ Theory(M,\ F)$
Augment	$\langle M;\ F\,\rangle$	$\Rightarrow \langle M,\ A;\ F\,\rangle$	$A\ =\ Theory(M,\ F)$
Unsat	$\langle M;\ \emptyset,\ F\,\rangle$	$\Rightarrow UNSAT$	
Resume	$\langle M,\ \overline{\ell}^{\delta};\ F;\ C\,\rangle$	$\Rightarrow \langle M,\ \ell^{C};\ F\,\rangle$	$\ell \in C$
Resolve	$\langle M,\ \ell^{C'};\ F;\ C\rangle$	$\Rightarrow \langle M;\ F;\ (C \setminus \{\overline{\ell}\}) \cup (C' \setminus \{\ell\})\,\rangle$	$\overline{\ell} \in C$
Backtrack	$\langle M,\ A;\ F;\ C\,\rangle$	$\Rightarrow \langle M;\ F;\ C\,\rangle$	$otherwise$

CDCL(T) – SAT vs SMT

SAT engine

- Truth assignment is symmetric for Boolean variables
- Probing (for failed literals)
  - L is failed if asserting L & F infers false by unit propagation.
  - Cost of propagation controlled by clause watch list
- Boolean Variables are fixed during search
- "Fast restart" introduced to prioritize variables used in conflicts

SMT engine

- Truth values of Booleans are not independent
  - $x \leq 0, x \leq 1$ are dependent
- Cost of propagation depends on theories
- Quantifier instantiation, theory lemmas introduce fresh literals (all the time)
- Fast restarts appears likely not a great idea

CDCL – CaDiCaL loop

```
def CDCL():
  while True:
    if [] in clauses:        return UNSAT
    elif in_conflict():      learn(); backtrack()
    elif not free_vars:      return SAT
    elif should_propagate(): propagate()
    elif should_simplify():  simplify()
    elif should_restart():   restart()
    elif should_prune():     prune_clauses()
    else:
      var  = choose_var(free_vars)
      sign = choose_sign(var)
      assign(var, sign)
```

CDCL(T)

```
def CDCL():

  while True:
    if [] in clauses:         return UNSAT
    elif in_conflict():       learn(); backtrack()
    elif not free_vars:       if theory.delay_propagate() return SAT
    elif should_propagate(): propagate(); theory.propagate()
    elif should_simplify():  simplify(); theory.simplify()
    elif should_restart():   restart()
    elif should_gc():         gc(); theory.gc()
    else:
      theory.push()
      var  = choose_var(free_vars)
      sign = choose_sign(var)
      assign(var, sign)
      theory.assign(var, sign)
```

Solver Internals

Terms and Formulas

de Bruijn index

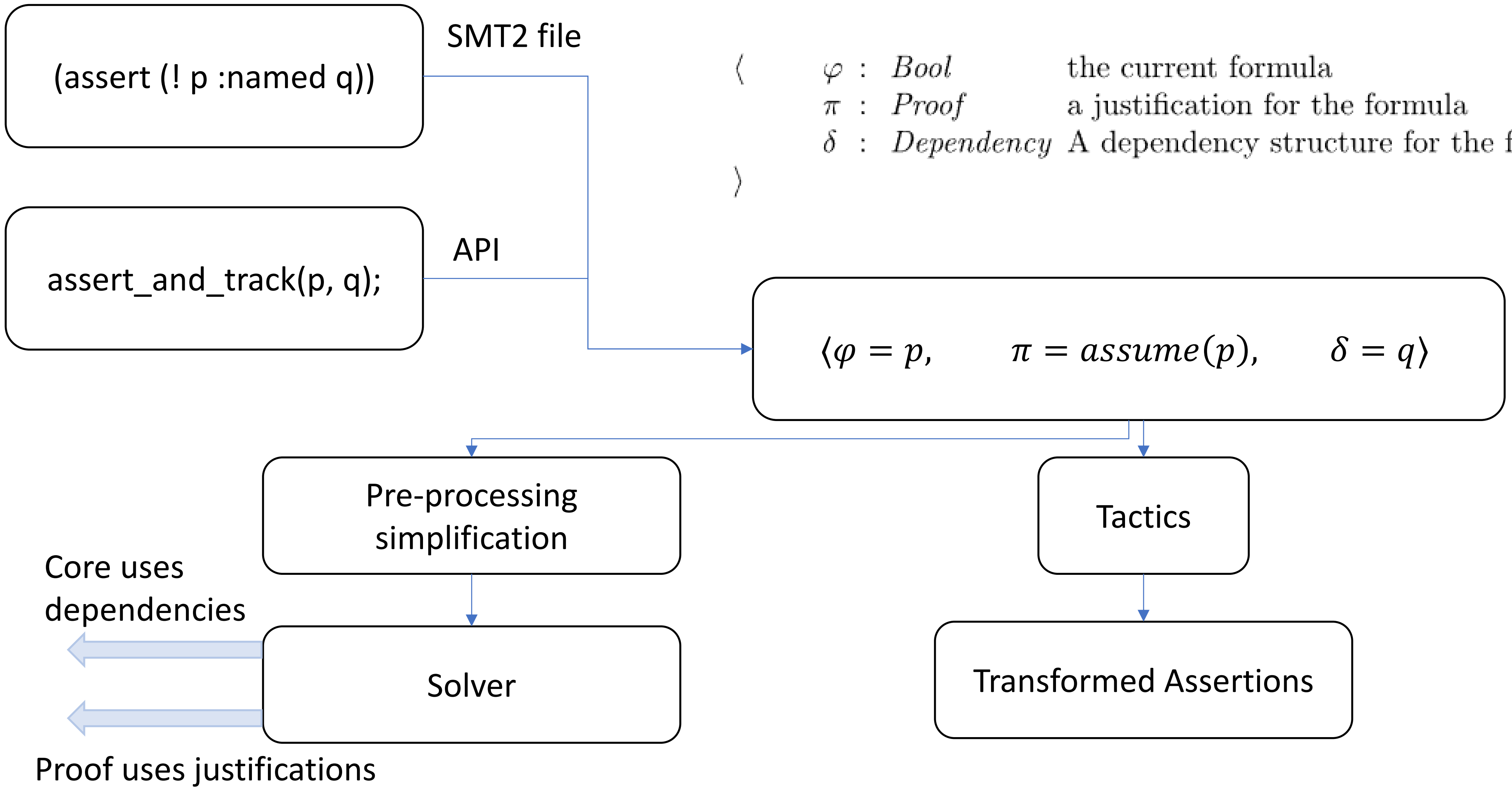
```
type Expr
   Var{ index : int; sort : Sort }
   App{ f : FuncDecl; args : list<Expr> }
   Quantifier{ b : Binder; decl : list<Declaration>; body : Expr; ...}
```

Terms are *hash-consed*

mkApp(f, args) =
let *t* = *App(f,args)*
let *t'* = *termTable[t]*
if *t'* = *nil* **then**
    termTable[t] ← *t*; *t*
else *t'*

Assertion Internals

$$
\left\langle
\begin{array}{lcll}
\varphi & : & \mathit{Bool} & \text{the current formula} \\
\pi & : & \mathit{Proof} & \text{a justification for the formula} \\
\delta & : & \mathit{Dependency} & \text{A dependency structure for the formula}
\end{array}
\right\rangle
$$



From Assertions to Solver State

$$x \geq 0 \vee (p \wedge q(x))$$

Clauses

$$(1\ 2)\ (1\ 3)$$

bool_var2expr

$$1 \mapsto x \geq 0, 2 \mapsto p, 3 \mapsto q(x)$$

expr2enode

$$\begin{aligned} n_{23}: \langle f &= x, args = \epsilon, \\ P &= [n_{27}], \\ th &= [(arith, 4)] \rangle \end{aligned}$$

z3 search.smt2 /tr:euf tactic.default_tactic=smt sat.smt=true

```
(sat
1: -3 none @1
   1 binary 3@1
2: -0 none @2
3: -2 none @3
(1 2)
(1 3)
updates 25
newlits 0 qhead: 0
neweqs  0 qhead: 0
(declare-fun q (Int) Bool): un 27
#32 := 1 [t 5:0]
#33 := 1.0 [t 5:1]
#24 := 0 [t 5:2]
#34 := 0.0 [t 5:3]
#25 := (>= x 0) [b1 := T no-cgc] [t 5:5]
#23 := x [p 27] [t 5:4]
#26 := p [b2 := F]
#27 := (q x) [b3 := F]
bool-vars
1: 25 l_true (>= x 0) arith
2: 26 l_false p
3: 27 l_false (q x)
number of constraints = 10
(0) j0 >= 1
(1) j0 <= 1
(2) j1 >= 1
(3) j1 <= 1
(4) j2 >= 0
(5) j2 <= 0
(6) j3 >= 0
(7) j3 <= 0
(8) j4 >= 0

[0]    := (1, 0)                [(1, 0), (1, 0)]
[1]    := (1, 0)                [(1, 0), (1, 0)]
[2]    := (0, 0)                [(0, 0), (0, 0)]
[3]    := (0, 0)                [(0, 0), (0, 0)]
[4]    := (0, 0)                [(0, 0), oo]
v0 j0, int := #32: 1
v1 j1 := #33: 1.0
v2 j2, int := #24: 0
v3 j3 := #34: 0.0
v4 j4, int, shared := #23: x
v5 1 l_true := #25: (>= x 0)
)
```

Core $\leftrightarrow$ Solver interface

CDCL

$assigned(\ell@1)$ $\longrightarrow$

$propagate(\ell, Explain)$ $\longleftarrow$

EUF Core

$$n_{23}: \langle f = x, args = \epsilon,\\ P = [n_{27}],\\ th = [(arith, 4)]\rangle$$

Dispatch Theory Solver

$$n_{25}: \langle \geq, args = \epsilon,\\ boolVar = 1,\\ th = [(arith, 5)]$$

Does not participate in congruence closure

Theory variable

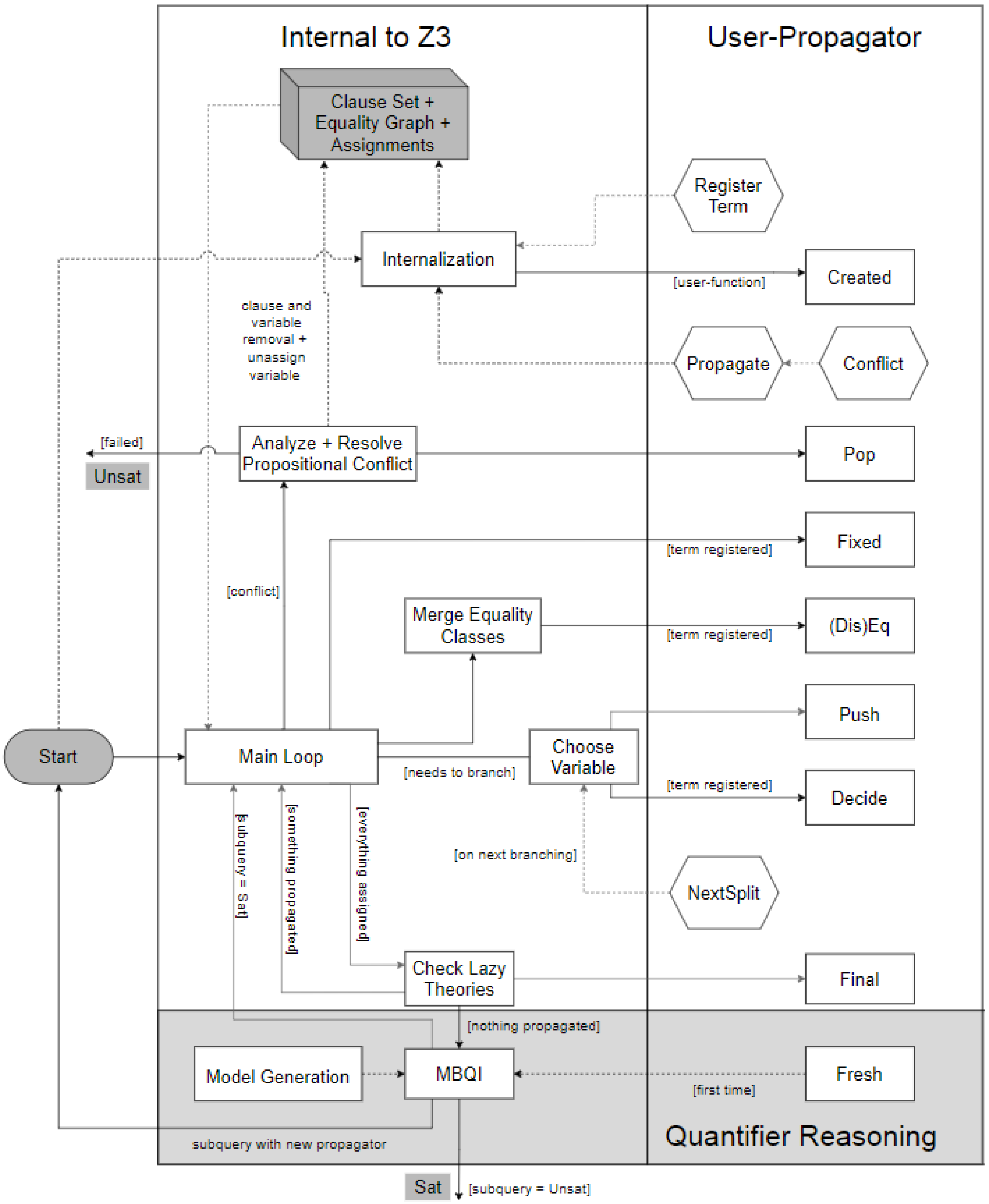
Solver

$propagate(x \geq 0@5, Explain)$ $\longleftrightarrow$

$propagate(n \simeq n', Explain)$ $\longleftrightarrow$

Custom Theories

- fixed: The CDCL core assigned a boolean/bit-vector value to a registered expression.
- eq: The EUF solver merged two equivalence classes. The two merged representatives will be reported.
- created: A new instance of a function symbol is encountered. e.g., $f(x)$ was instantiated to $f(a)$.
- final: The solver got a consistent assignment to all boolean variables. All theories get the final chance to intervene.
- Further: push, pop, fresh, decide, and diseq



Model-based Theory Combination

$$\boxed{x = f(z), f(x) \not\simeq f(y), \qquad 0 \le x \le 1, 0 \le y \le 1, z = y - 1}$$

$x = \star_1\ y = \star_2\ z = \star_3$

$f(\star_1) = \star_1\ f(\star_2) = \star_2\ f(\star_3) = \star_1$

$x = 1, y = 1, z = 0$

Create fresh literal $x \simeq y$

Case split on $x \simeq y \leftarrow$ T

Conflict, backtrack $\boldsymbol{x \neq y}$

$$\boxed{x = f(z), f(x) \not\simeq f(y), \qquad 0 \le x \le 1, 0 \le y \le 1, z = y - 1, \boldsymbol{x \neq y}}$$

Create fresh literal $x \simeq z$

Case split on $x \simeq z \leftarrow$ T

$x \simeq z$

$x = 0\ y = 1, z = 0$

$x = \star_1\ y = \star_2\ z = \star_1$

$f(\star_1) = \star_1\ f(\star_2) = \star_2$

Relevancy Filtering

Purpose: expose only subset of literal assignments to T solvers

Reason: Delays introduction of terms for T-and quantifier instantiation

Idea: Simulate tableau reasoning on top of CDCL

$((a \wedge b) \vee c)$

$\big((a \wedge b) \vee c\big)$ $(\neg(a \wedge b) \vee a)$ $(\neg(a \wedge b) \vee b)$ $\big((a \wedge b) \vee \neg a \vee \neg b\big)$

Root clause Definition clauses

Scenario 1: c is assigned to T, root clause is satisfied. Atoms $(a \wedge b), a, b$ are never set relevant

Scenario 2: c is assigned to F, $(a \wedge b)$ is assigned T. Atoms a, b are marked relevant (and propagated to T)

`smt.relevancy={0,1,2} (least to most use of relevancy filter)`

Ackermann reductions

$$a_0 \not\simeq a_{100} \wedge \bigwedge_{0 \leq i < 100} (a_i \simeq b_i \vee a_i \simeq c_i) \wedge (a_i \simeq b_i \implies b_i \simeq a_{i+1}) \wedge (a_i \simeq c_i \implies c_i \simeq a_{i+1})$$

The proofs are linear if we admit clauses using fresh literals of the form

$$(a_i \simeq b_i \wedge b_i \simeq a_{i+1} \implies a_i \simeq a_{i+1})$$
$$(a_i \simeq c_i \wedge c_i \simeq a_{i+1} \implies a_i \simeq a_{i+1})$$

Z3 dynamically introduces such auxiliary clauses based on transitivity of equality and congruence rules of the form

$$t_1 \simeq s_1, \ldots, t_k \simeq s_k \implies f(t_1, \ldots, t_k) \simeq f(s_1, \ldots, s_k)$$

`smt.dack.threshold = 10,` `smt.dack.eq = false`

Iterative Deepening

```
(define-fun-rec length ((ls (List Int))) Int
  (ite ((_ is nil) ls) 0 (+ 1 (length (tail ls)))))

(define-fun-rec nat-list ((ls (List Int))) Bool
  (ite ((_ is nil) ls)
       true
       (and (>= (head ls) 0) (nat-list (tail ls)))))

(declare-const list1 (List Int))
(declare-const list2 (List Int))
(assert (> (length list1) (length list2)))
(assert (not (nat-list list2)))
(assert (nat-list list1))
```

- Assume ((_ is nil) list1) ((_ is nil) list2)
- Unsat core: ((_ is nil) list1) `(assert (> (length list1) (length list2)))`

- Assume ((_ is nil) (tail list1)) ((_ is nil) list2)
- Unsat core: ((_ is nil) list2) `(assert (not (nat-list list2)))`

- Assume ((_ is nil) (tail list1)) ((_ is nil) (tail list2))
- Unsat core: ((_ is nil) (tail list1) `(assert (> (length list1) (length list2)))` `(assert (not (nat-list list2)))`

- Assume ((_ is nil) (tail (tail list1))) ((_ is nil) (tail list2))
- SAT

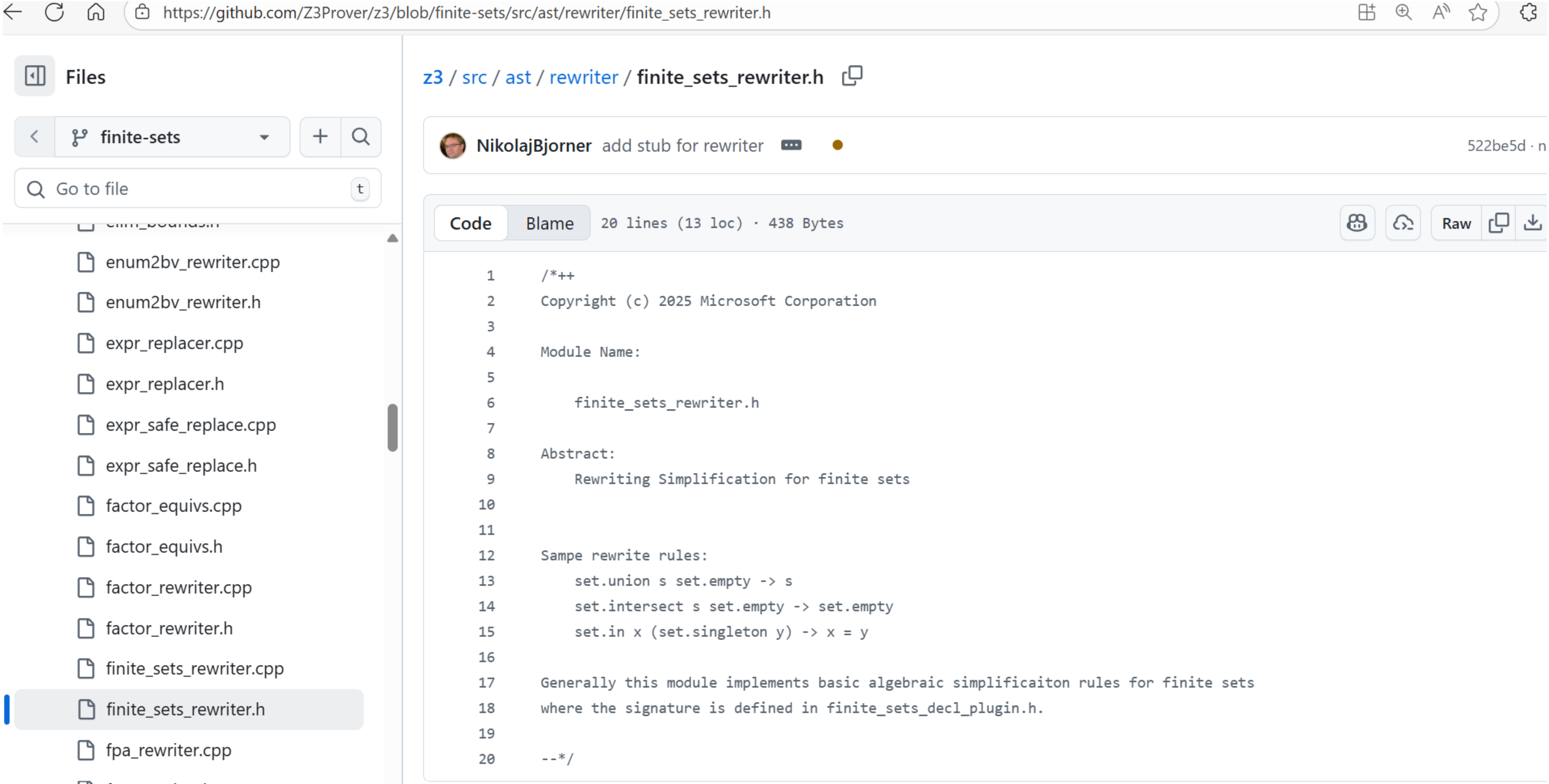
Pre-processing Rewriting Simplification

For Finite Sets

Finite Set Algebraic Simplification rules

- $S \cap \emptyset \rightarrow \emptyset$
- $S \cup T \rightarrow T \cup S$ if $\text{code}(T) < \text{code}(S)$
- $x \in \{ y \} \rightarrow x = y$

Finite Sets Rewriter




```
/*++
Copyright (c) 2025 Microsoft Corporation

Module Name:

    finite_sets_rewriter.h

Abstract:
    Rewriting Simplification for finite sets


Sampe rewrite rules:
    set.union s set.empty -> s
    set.intersect s set.empty -> set.empty
    set.in x (set.singleton y) -> x = y

Generally this module implements basic algebraic simplificaiton rules for finite sets
where the signature is defined in finite_sets_decl_plugin.h.

--*/
```

Let’s be lazy, but not too lazy, but verify

1. Ask copilot to produce rewrite rules and implementation
2. Axiomatize finite sets for a 3-4 variables. Enumerate terms and mine for equalities.

Q: how would ***you*** address the following?

- Correctness of simplification rules and code.
- Adequacy of simplification rules? Do they cover useful cases, what could be covered.

Pre-processing
Global Simplification

For Finite Sets

Global Simplification

$F[S \cap X]$ - suppose this is the only occurrence of X in F.

Can we solve equisatisfiable F without $S \cap X$?

Example: If F is monotone in $S \cap X$, we could replace $S \cap X$ by S.

Task 3: develop global simplification rules for finite sets. Integrate rules into