Research summary
Maria Paola Bonacina

(May 23, 2025)

My research area is *automated reasoning*, or how to make computers reason, not necessarily like humans, but rather in their own way. In automated reasoning, logical formulæ are used to express properties of programs, data structures, protocols, circuits, or other systems. Reasoning problems involve *validity* queries (does a *conjecture* φ follow from a set H of *assumptions*?) or *satisfiability* queries (does a set S of constraints admit solution?). The answer to a validity query is a *proof* (that φ follows from H or that $S = H \cup \{\neg\varphi\}$ is unsatisfiable) or a *counter-example* (a *model* of S). The answer to a satisfiability query is a *model* (of S) or a *proof* (that S is unsatisfiable). Automated reasoning is about designing methods to solve these problems, proving properties of such methods (e.g., *soundness*, *completeness*, *termination*), and implementing them in *automated reasoners*, interfaced with human or software users.

My research is motivated by both fundamental challenges and applications. A constant challenge is improving the trade-off between *generality* (how large is the class of problems a method can handle) and *efficiency*. The use of automated reasoning for the *analysis*, *verification*, *synthesis*, and *optimization* of *software* is a major application, because logic is the calculus of computation [46]. Automated reasoners are used for discharging verification or synthesis conditions, refining abstractions, generating tests for testing, and examples for synthesis. Correct-by-construction software, provable privacy, verification of distributed protocols and systems, and verification of randomized algorithms are objectives at the frontier of this application area. Thus, automated reasoning plays an increasingly crucial rôle in ensuring features, such as reliability and privacy, that are of extreme importance for society.

The application of automated reasoning to *improve generative AI techniques* [41] and the *combination of automated reasoning and generative AI techniques* are fundamental challenges towards realizing a more sophisticated artificial intelligence. Automated reasoning and machine learning are complementary, as automated reasoning allows the machine to reason based on laws, principles, knowledge, whereas machine learning allows the machine to learn from data. Automated reasoning is connected with symbolic computation (e.g., constraint problem solving, computer algebra), and computational logic (e.g., foundations, rewriting), and it also finds applications in deductive knowledge bases, computer mathematics, mathematical libraries, and education.

My research to date can be presented in four overlapping threads:

A. Theorem-Proving Strategies and Satisfiability Procedures

B. Interpolation of Proofs

C. Distributed Automated Deduction

D. Strategy Analysis

summarized in the sequel, with citations referring to the publications in my curriculum vitae.

A. Theorem-Proving Strategies and Satisfiability Procedures

Most reasoning methods transform the problem $H \cup \{\neg\varphi\}$ into an equisatisfiable set S of *clauses*, a standard machine format. In first-order logic (FOL) unsatisfiability is semidecidable, satisfiability is not even semidecidable, and reasoning methods are *semidecision procedures* called *theorem-proving strategies* [91, 76, 87, 74, 73, 86, 5, 4] and implemented in *theorem provers.* A theorem-proving strategy is characterized by an *inference system* and a *search plan.* An inference system is a set of *inference rules* that manipulate clauses, until a *refutation* is found, and a *proof* can be reconstructed from the *derivation.* A strategy may reason mostly *forward* (i.e., from the assumptions H) or mostly *backward* (i.e., from the clauses in the clausal form of $\neg\varphi$, called *goal clauses*), to the point of being *goal-sensitive*, if all generated clauses are connected with a goal clause. A strategy may be *semantically-guided* by a given *fixed* interpretation, and it is *proof confluent*, if it does not need to undo inferences by backtracking.

Ordering-based strategies work with a set of clauses, initially the input set S. *Expansion* inference rules, such as *resolution*, *paramodulation*, and *superposition*, generate and add clauses, consequences of the existing ones. *Contraction* inference rules, such as *subsumption* and *simplification*, delete or replace *redundant* clauses. A refutation is reached when the empty clause $\Box$ is generated. *Well-founded orderings* on terms, literals, clauses, or proofs, are used to restrict expansion, and to define contraction and redundancy. Contraction inference rules and an *eager-contraction* search plan characterize *contraction-based* strategies, that are a default choice when *equality* is involved. Typical ordering-based strategies reason primarily *forward*, as $\neg\varphi$ is treated as an additional hypothesis, may be *semantically-guided*, and are *proof confluent* [91, 76, 73, 4].

Subgoal-reduction strategies, based on *linear resolution*, *model elimination*, or *tableaux*, apply inferences to reduce a current goal to subgoals [91, 76, 74, 73]. They operate on a stack of goals and use *depth-first search with backtracking and iterative deepening.* The stack of goals is the *frontier* of a tree-like structure, a tableau, where branches represent possible models. A refutation is found when all branches are *closed* as contradictory. *Lemmaizing* (i.e., turning solved goals into lemmas) and *caching* (i.e., storing solved or failed goals in a look-up table) counter the redundancy of repeated subgoals. Typical subgoal-reduction strategies reason mostly *backward* and are *goal-sensitive. Instance-based strategies* generate *instances* of clauses, and invoke a satisfiability procedure to test sets of ground instances for unsatisfiability. Typical instance-based strategies reason mostly *forward*, and are *model-driven*, if they generate instances that are false in the model found by the satisfiability procedure when it detects satisfiability.

In propositional logic and in decidable fragments of FOL or of a first-order theory $\mathcal{T}$, satisfiability is decidable, and reasoning methods are *decision procedures* called *satisfiability procedures* and implemented in *satisfiability solvers.* A satisfiability procedure is characterized by a *transition system* and a search plan. A transition system is a set of *transition rules* that transform a *trail* representing a *candidate model*, until either a model is found or an unsolvable *conflict* reveals that no model exists. These procedures are *model-based*, as they build and discard candidate models, and *conflict-driven*, as they apply nontrivial inferences mostly to *explain* and *solve* conflicts [74,39, 86, 72, 31]. The archetypal satisfiability procedure is the *Conflict-Driven Clause Learning* (CDCL) procedure for SAT. It works by *deciding* truth assignments to literals and *propagating* their consequences (*Boolean clausal propagation*). When a conflict arises (a clause is false in the current assignment), the procedure *explains* it by *resolution* and *learns* a *lemma* to avoid hitting that conflict again. The CDCL($\mathcal{T}$) procedure integrates a satisfiability procedure for a theory $\mathcal{T}$ in CDCL. If $\mathcal{T}$ is a union of theories $\mathcal{T}_1, \ldots, \mathcal{T}_n$, the existence of a satisfying $\mathcal{T}$-model depends on that of $\mathcal{T}_i$-models agreeing on the interpretation of shared symbols and on the cardinalities of shared sorts. The *Nelson-Oppen* or *equality sharing* scheme assumes that the theories are *stably infinite*, meaning that they admit models with countably infinite domains for all sorts, and *disjoint*, meaning that they do not share function or predicate symbols other than equality, so that the $\mathcal{T}_i$-models only need to agree on which shared constants (or free variables) are equal. The equality sharing method combines the $\mathcal{T}_i$-satisfiability procedures as *black-boxes* that need to propagate all entailed disjunctions of equalities between shared constants [72, 31]. CDCL was generalized to *conflict-driven satisfiability procedures* for quantifier-free fragments of arithmetic. Key features of such procedures are *assignments to first-order variables* and conflict explanation by lemmas that may contain *new* (i.e., non-input) atoms. The MCSAT procedure (surveyed in [74, 39, 86]) integrates CDCL and one conflict-driven theory satisfiability procedure.

My research develops *cross-fertilization methods* uniting features from different paradigms, understanding inference and transition rules as *transformation of candidate proofs/models.*

A1. Target-Oriented Completion. I investigated how to make ordering-based strategies *goal-sensitive*, or equivalently *target-oriented* (φ is the *target* theorem) in the context of *completion procedures* [110, 71, 101, 68, 109]. Completion was understood as generation of confluent rewrite systems for equational theories. Theorem proving was either a side-effect or the successive application of the confluent rewrite system to decide the word problem by rewriting. The latter way is impractical, since few equational theories have finite confluent rewrite systems. The first one is intrinsically inefficient, because in order to achieve confluence, generating a *saturated set*, the procedure performs inferences that are *unnecessary* to prove φ. The key point is *fairness.* The pre-existing notion, *uniform fairness*, captures the inferences needed to generate a saturated set. I proposed a new notion of *fairness* that captures the inferences needed to prove φ. My *target-oriented* framework for completion is based on applying *proof orderings* to the proofs of φ [68, 109, 67, 100, 108, 26, 24]. It covered all known completion procedures for equational logic, including practical *target-oriented techniques* [106, 69, 65], *inductionless induction*, and the generation of saturated sets as a special case. In experiments, I obtained the *first automated proof of the Dependency of the Fifth Axiom in Łukasiewicz's many-valued logic* [107, 69, 99], introducing it as a source of benchmarks (this story appears in the introduction of [4]). The *Linear Completion* procedure that interprets *rewrite programs* is another instance of target-oriented completion. I defined the *operational* and *denotational semantics* of rewrite programs, disproving the folklore that they are the same as Prolog, showing the different expressive power of programming with *bi-implications* versus *implications*, and the effect of *simplification* on *termination* [70, 109, 30]. I also gave *counter-examples to the completeness of the RUE/NRF inference systems* [98].

A2. Lemmatization from Model Elimination to Semantic Resolution. Understanding that *lemmaizing* is a form of forward reasoning in subgoal-reduction strategies, I had the idea of using it to increase the *goal-sensitivity* of *semantic resolution* strategies [59, 21] Semantic guidance orients the strategy towards forward or backward reasoning. Lemmaizing is a *meta-level inference rule*, because it derives a lemma based on a whole fragment of the derivation. I showed that it can be used to add backward inferences to a forward strategy and vice versa. I defined a set of inference rules that implement lemmaizing in semantic resolution strategies, and I showed how to add contraction, including *purity deletion*, to such strategies. Thus, lemmaizing and contraction can coexist, and contraction can take advantage of the generation of unit lemmas. For subgoal-reduction strategies, I formalized *caching* and *depth-dependent caching* as inference rules justified by the meta-rules for lemmaizing. I observed that lemmaizing and caching allow subgoal-reduction strategies to keep some generated clauses, and therefore make *subsumption* possible, bringing a feature of ordering-based strategies to subgoal-reduction strategies.

A3. Inferences and Canonicity. I expanded my early work on target-oriented completion in an analysis of how inferences *reduce proofs* and *transform presentations* [16]. A presentation is *contracted*, if it is made of the premises of the *minimal proofs*; *canonical*, if it is made of the premises of the minimal proofs in the whole theory (*normal form proofs*); *complete*, if it offers at least a normal form proof for each theorem; and *saturated*, if it features all normal form proofs for all theorems. Therefore, canonical and saturated coincide only if normal form proofs are unique, and *complete*, rather than saturated, is sufficient for theorem proving. Accordingly, while a *uniformly fair* derivation produces a *saturated* presentation, a *fair* derivation only yields a *complete* one. In practice, a search plan should schedule enough expansion and contraction inferences to get in the limit a complete and contracted presentation. I applied this framework to *implicational systems*, that are presentations of *propositional Horn theories* [48, 75]. Given an implicational system $\mathcal{T}$ and a set X of propositional variables, the problem is to find the least $\mathcal{T}$-model that satisfies X. Implicational systems can be translated into propositional rewrite systems, where the rewrite relation is bi-implication as in Linear Completion. I defined a completion procedure for propositional rewrite systems that computes the least $\mathcal{T}$-model of X and transforms $\mathcal{T}$ into a *canonical* equivalent presentation. I also studied canonicity in *conditional equational theories*, where proof normalization yields decision procedures based on saturated presentations [75].

A4. Composing Theorem-Proving Inference Systems and Satisfiability Procedures

If a theorem-proving strategy is *guaranteed to halt* on problems in a class, it is a *decision procedure* for that class. Applying theorem-proving strategies to *satisfiability modulo theories* (SMT) offers several benefits: existing theorem provers can be used *off the shelf*, correctness and completeness do not need to be proved for each theory, combination of theories reduces to giving as input the union of their presentations, and proof generation is a native feature of theorem provers. With this motivation, I produced results showing that a standard *ordering-based inference system*, known as the *superposition calculus*, or *superposition* for short, generates finitely many clauses from certain satisfiability problems, so that every fair strategy with that inference system is a decision procedure [82, 81, 80, 53, 52, 104, 14, 51, 50, 103, 90, 15]. Arithmetic or bitvectors do not lend themselves to reasoning by inferences from axioms. Thus, I investigated how to get decision procedures by *pipelining* [49, 13] or by *integrating* [47, 12] superposition and CDCL(T).

A4.1. Superposition Decision Procedures. We showed that superposition decides the satisfiability of sets of ground literals in the theories of *records with or without extensionality*, *possibly empty lists*, *arrays with or without extensionality*, *integer offsets*, *integer offsets modulo* [80, 53, 14], and *recursive data structures* [51], including *acyclic non-empty lists* as a special case. I discovered a condition, called *variable-inactivity*, whereby if the theories are *disjoint* and *variable-inactive*, and superposition terminates on satisfiability problems in each theory, it terminates also on satisfiability problems in their union [53, 14]. All the theories above are variable-inactive. Contrary to the folklore that a generic prover cannot compete with solvers with built-in theories, the experimental comparison of the *E prover* with the CVC and CVC Lite SMT-solvers was overall favorable to the E prover [80, 81, 53, 14]. If a theory is *not stably infinite*, superposition is guaranteed to generate eventually an *at-most cardinality constraint*, so that the theory is *not variable-inactive* [52, 104]. Thus, *variable-inactivity implies stable-infiniteness*, and superposition can discover the *lack of infinite models* by generating an at-most cardinality constraint [52, 14]. Our superposition decision procedures for *records without extensionality* and for *integer offsets modulo* [51, 14], as well as those for *integer offsets* and for *records with extensionality* [90, 15], are *polynomial* (for the latter theory ours was the first polynomial decision procedure). Then, we showed that for *variable-inactive* theories, if superposition decides the satisfiability of sets of ground literals, it also decides the satisfiability of sets of ground clauses [90, 15]. This result applies to the theories of *equality*, *non-empty possibly cyclic lists*, *arrays with or without extensionality*, *injective arrays* [50, 103], *finite sets with or without extensionality*, *records with or without extensionality*, *possibly empty possibly cyclic lists*, *integer offsets modulo*, *recursive data structures*, and all their unions.

A4.2. Pipelining Superposition and CDCL(T). In problems from applications the input clause set S may contain very long clauses, and while CDCL-based solvers break clauses apart by case analysis, superposition generates longer and longer clauses. *Decision procedures by stages* addresses this obstacle by pipelining superposition with an SMT-solver [49, 13]. S is partitioned into a set S_1 of unit clauses and a set S_2 of non-unit clauses. Superposition saturates $\mathcal{T} \cup S_1$ into $\mathcal{T} \cup \bar{S}$, where $\bar{S}$ is finite, ground, and capable of entailing all clauses that can be generated from $\mathcal{T} \cup S_1$, and $\bar{S} \cup S_2$ is fed to the SMT-solver. We found sufficient conditions to ensure that $\bar{S}$ has these properties, and we obtained *$\mathcal{T}$-decision procedures by stages* for *arrays with or without extensionality*, *records with or without extensionality*, *integer offsets*, and their unions. If the problem involves two theories $\mathcal{T}_1$ and $\mathcal{T}_2$ such that superposition is a decision procedure for each, we decompose S into $\mathcal{T}_1 \cup S_1$, $\mathcal{T}_2 \cup S_2$ and S_3, where S_1 contains unit $\mathcal{T}_1$-clauses, S_2 contains unit $\mathcal{T}_2$-clauses, and S_3 contains the remaining clauses. The procedure saturates $\mathcal{T}_1 \cup S_1$ into $\mathcal{T}_1 \cup \bar{S}_1$ and $\mathcal{T}_2 \cup S_2$ into $\mathcal{T}_2 \cup \bar{S}_2$, and passes $\bar{S}_1 \cup \bar{S}_2 \cup S_3$ on to the SMT-solver. Thus, the part of the problem involving for example arithmetic or bitvectors can be given directly to the SMT-solver.

A4.3. The CDCL($\Gamma + \mathcal{T}$) Procedure with Speculative Inferences. In problems from applications the input clause set S often contains ground clauses with $\mathcal{T}$-symbols and a subset $\mathcal{R}$ of non-ground clauses without $\mathcal{T}$-symbols. The CDCL(Γ+$\mathcal{T}$) procedure integrates a superposition-based inference system Γ in CDCL($\mathcal{T}$), in such a way that Γ works with non-ground $\mathcal{R}$-clauses and ground unit $\mathcal{R}$-clauses on the trail, while CDCL($\mathcal{T}$) takes care of ground clauses [47, 12, 44]. Since trail literals may be withdrawn upon backjumping, they are memorized in clauses as *hypotheses*, that are inherited through inferences. When backjumping removes literals from the trail, the clauses depending on them are also removed. Contraction rules are adjusted to take this dynamic effect into account. If $\mathcal{R}$ is *variable-inactive*, CDCL($\Gamma+\mathcal{T}$) is *refutationally complete* [47, 12]. Indeed, since variable-inactivity implies stable infiniteness [52], and superposition is guaranteed to generate clauses that entails all disjunctions of equalities between shared constants [13], the completeness requirements for equality sharing are fulfilled. As CDCL($\mathcal{T}$) uses depth-first search with backtracking, the *fairness* of CDCL($\Gamma+\mathcal{T}$) requires *iterative deepening* on the number of Γ-inferences. If S is unsatisfiable, CDCL($\Gamma+\mathcal{T}$) is guaranteed to halt with a contradiction; otherwise, it may either halt with a model, or get *stuck* at the current limit on number of Γ-inferences. The third oucome is excluded for those theories for which Γ is a decision procedure. In order to get more decision procedures, CDCL($\Gamma+\mathcal{T}$) features *speculative inferences*: it can add to the current set an arbitrary clause, with as hypothesis a new propositional variable added to the trail to keep track of the decision [47, 12]. If S is satisfiable, and the added clause causes a contradiction, CDCL($\Gamma+\mathcal{T}$) handles it as a conflict, undoing the speculative inference by backjumping. If we can provide a sequence of clauses (e.g., equalities) whose addition enforces termination, CDCL($\Gamma+\mathcal{T}$) is a decision procedure. This is the case for several *axiomatizations of type systems* [47, 12].

A5. SGGS: Semantically Guided Goal-Sensitive Theorem Proving

We designed SGGS to be the first theorem-proving method that is *simultaneously* first-order, *semantically guided*, *goal-sensitive*, *model-based*, *instance-based*, *proof confluent*, and *conflict-driven* [88, 78, 43, 9, 8, 39, 86]. SGGS is the first method that succeeded in *generalizing CDCL to FOL.*

A5.1. Model Representation in SGGS. SGGS is *semantically guided* because it assumes an *initial interpretation* I. Given I and input set S of clauses, unless $I \models S$, SGGS seeks to build a model of S by determining which literals that are true in I (*I-true* literals) should be falsified to satisfy S [9]. The candidate model is represented by a trail Γ, which is a sequence of (possibly constrained) *non-ground* clauses with *selected literals* (e.g., $x \not\equiv b \rhd P(x)$ represents all ground instances of $P(x)$ except $P(b)$) [78]. Since variables are implicitly $\forall$-quantified, if literal L is true, all its ground instances are, but one false ground instance suffices to make L false. We say that L is *uniformly false* if all its ground instances are (i.e., $\neg L$ is true). We call *I-false* a literal that is uniformly false in I. SGGS builds the trail Γ in such a way that all literals in all clauses in Γ are either I-true or I-false, and I-false literals are preferred for selection. An I-true literal is selected only in a clause where all literals are I-true (*I-all-true* clause). The associated interpretation $I[\Gamma]$ is I modified to satisfy the selected literals in Γ. Thus, *literal selection* plays the role of *decision.* Literal L is uniformly false in $I[\Gamma]$, if all its ground instances appear negated among those that selected literal M makes true in $I[\Gamma]$. If L is I-true, SGGS *assigns* L to (the clause of) M. A clause C is a *conflict clause* if all its literals are uniformly false in $I[\Gamma]$. If all literals in C, except the selected literal L, are uniformly false in $I[\Gamma]$, literal L is *implied* and C is its *justification.* SGGS ensures that every I-all-true clause in Γ is either a *conflict clause* (all literals assigned) or the *justification* of its selected literal (all literals assigned except the selected one). By computing these assignments SGGS inferences perform *first-order clausal propagation* [9].

A5.2. The SGGS Inference System. *SGGS-extension* adds to the trail an instance of an input clause and selects one of its literals [8]. The added instance is built in order to capture ground instances of the input clause not satisfied by the current $I[\Gamma]$. *SGGS-deletion* deletes clause C from $\Gamma C\Gamma'$, if C is satisfied by $I[\Gamma]$. Similar to CDCL, if SGGS-extension adds to Γ a conflict clause E, *SGGS-resolution explains* the conflict resolving upon an I-false literal in E and the I-true selected literal of a justification. SGGS ensures that all I-false literals in E can be resolved away in this manner, yielding either $\square$ or an I-all-true conflict clause C. *SGGS-move* solves the conflict and *learns* C, by moving it to the left of the clause whose selected literal makes C's selected literal uniformly false: C's selected literal becomes an implied literal. Thus, SGGS gets out of conflict without undoing inferences by backtracking or backjumping. *SGGS-splitting* of clause C by clause D replaces C by a *partition*, where all ground instances that a specified literal in C has in common with D's selected literal are confined to one element. This enables SGGS-resolution or SGGS-deletion to remove such *intersections*, ridding Γ of contradictions or duplications. SGGS *makes progress* in two ways: either it extends Γ by an SGGS-extension, or it repairs $I[\Gamma]$ by either explaining and solving a conflict or removing an intersection in Γ. *Fairness* ensures that SGGS-deletion and other clause removals are applied eagerly, trivial splitting is avoided, progress is made whenever possible, every SGGS-extension generating a conflict clause is *bundled* with explanation and conflict-solving inferences to solve the conflict before further extensions, and inferences applying to shorter prefixes of the trail are never neglected in favor of others applying to longer prefixes. SGGS is *refutationally complete* and *model complete* in the limit (if the input is satisfiable, the limit of every fair SGGS-derivation represents a model) [8].

A5.3. SGGS, Decision Procedures, and Horn Theories. We proved that SGGS decides several known decidable fragments of FOL: *stratified* [37, 3], *positive variable dominated* (PVD) [37, 3], *bounded depth increase* (BDI) [3], and *Datalog* [3]. The stratified fragment is a many-sorted generalization of the *Bernays-Schönfinkel class*, whose clausal version is known as *Effectively PRopositional logic* (EPR). On the other hand, SGGS with *sign-based semantic guidance* (i.e., I is either *all-negative* – all negative literals are true – or *all-positive* – all positive literals are true) does *not* decide other known decidable fragments of FOL: *Ackermann*, *monadic*, FO^2, and *guarded* [3]. These counterexamples show that the existence of a finite model does *not* imply the termination of SGGS with sign-based semantic guidance. Other examples show that SGGS terminates and represents with a finite trail an infinite Herbrand model, so that termination does *not* imply the existence of a finite Herbrand model. We discovered several new decidable fragments of FOL by showing that SGGS decides them: *positively/negatively restrained* [37, 3], *positively/negatively sort-restrained*, and *sort-refined-PVD* [3]. Since the size of SGGS-generated models can be upper-bounded, these new fragments enjoy the *small model property*. As restrainedness is an ordering-based property, it can be reduced to termination of rewriting: this means that it is undecidable in general, but in practice termination tools can be applied to find restrained sets [37, 3]. We also investigated the behavior of SGGS on Horn clauses [34], showing that SGGS with all-negative I *generates the least fixpoint model* of a set of definite clauses, and the first negative conflict clause announces a refutation. SGGS with all-negative (all-positive) I reasons forward (backward) on Horn clauses. The SGGS prototype Koala exhibited good experimental results [37, 34, 3].

A6. CDSAT: Conflict-Driven SATisfiability Modulo Theories and Assignment

CDSAT is a *conflict-driven* method for deciding the satisfiability of a formula modulo a *union of theories* and a possibly empty *initial assignment* (*satisfiability modulo theories and assignment* or SMA for short) [42, 40, 39, 86, 77, 38, 7, 36, 6, 35, 31, 1]. An SMA problem is satisfiable, if there exists a satisfying assignment that includes the initial one, and unsatisfiable otherwise. CDSAT generalizes MCSAT to generic combinations of theories, solving the problem of integrating CDCL with multiple conflict-driven theory reasoning procedures. Since CDSAT also accommodates black-box theory reasoning procedures, it also subsumes equality sharing and CDCL(T).

A6.1. The CDSAT Framework. A basic feature of CDSAT is that it works with *both Boolean and first-order assignments.* The initial assignment may contain Boolean (e.g., $L \leftarrow \mathsf{true}$) and first-order (e.g., $x \leftarrow 3$) assignments to terms occurring in the input formula. A formula F is viewed as a Boolean term and abbreviates $F \leftarrow \mathsf{true}$. Propositional logic is one of the theories (the Boolean theory). Assignable values are constants introduced by *conservative theory extensions*, so that terms and values remain separate. The CDSAT transition system orchestrates in a conflict-driven manner *theory inference systems*, called *theory modules* [42, 7]. A theory module is an abstraction of a theory reasoning procedure. Thanks to this abstraction, the distinction between conflict-driven and black-box procedure fades. The theory module for a black-box procedure has only one inference rule that detects unsatisfiability of a Boolean assignment to a set of literals. CDSAT works with a *trail* Γ of assignments, which includes the input, is shared by all theory modules, and represents a satisfying assignment in case of positive answer. The elements of Γ are either *decisions* or *justified assignments*, where the *justification* is a set of prior assignments in Γ. Decisions can be either Boolean or first-order. Input assignments are justified assignments with empty justification. All justified assignments are Boolean except for input first-order assignments.

A6.2. The CDSAT Transition System. The *Decide* rule lets a theory module post on Γ an *acceptable* assignment to a term that is *relevant* for its theory. Acceptability excludes assignments causing conflicts from which nothing can be learned. Relevance ensures that a theory module does not mingle with what belongs to others. *Deduce* adds to Γ a justified assignment derived by a theory inference, provided the term of the derived assignment comes from a *finite global basis.* This is crucial for termination, since theory inferences can generate *new* terms. *Deduce* transitions cover both *propagations* and inferences that *detect* and *explain* theory conflicts, letting them surface in Γ as Boolean conflicts. A *conflict state* is given by the trail Γ and a conflict, which is an unsatisfiable subset of Γ. If the conflict is at level 0, rule *Fail* reports unsatisfiability. Otherwise, *Resolve* unfolds the conflict, replacing a justified assignment by its justification. *Backjump* solves the conflict by flipping a Boolean assignment, so that the procedure will not hit the same conflict. In the Boolean case, these two rules can emulate CDCL conflict solving. As first-order assignments cannot be flipped, *UndoClear* undoes a first-order decision A and clears Γ of A's consequences, when Γ contains a *late propagation* (late w.r.t. A) that makes A unacceptable, so that A will not be repeated. *UndoDecide* undoes A, clears Γ of A's consequences, and flips a Boolean consequence of A, so that A will not be retried [42, 7]. *LearnBackjump* generalizes *Backjump* allowing CDSAT to flip a Boolean subset of the conflict into a learned clause [40, 6].

A6.3. CDSAT Theory Modules. The inference rules of a theory module derive Boolean assignments from assignments, and can generate new terms, provided they come from a finite *local basis*. We defined theory modules and local bases for *propositional logic*, and for the quantifier-free fragments of the theories of *equality*, *linear rational arithmetic* (LRA), and *arrays with extensionality* [42, 7, 6]. We also showed how a finite *global basis* can be built from the local bases [6]. If all modules are black-boxes, CDSAT can emulate equality sharing [6]. However, CDSAT does not require stable infiniteness, provided there is a *leading theory* $\mathcal{T}_1$ that knows all sorts in the union of theories and acts as an aggregator of cardinality requirements by different theories. The theory module of the leading theory enforces the aggregated requirements, such as *at-most cardinality constraints* [6]. For all above mentioned theories $\mathcal{T}$, we proved that the $\mathcal{T}$-module is *leading-theory complete* [6]. This means that if the $\mathcal{T}$-module cannot expand an assignment, for all $\mathcal{T}_1$-models satisfying the assignment there is a satisfying $\mathcal{T}$-model that agrees with the $\mathcal{T}_1$-model on cardinality of shared sorts and equality of shared terms [7]. If the theories are *disjoint*, there is a finite global basis that contains the input, and the theory modules are sound and leading-theory complete, CDSAT is *sound*, *terminating*, and *complete* [42, 7]. We also extended CDSAT with *proof generation* towards different proof formats, including resolution-based proofs [40, 6, 36].

A6.4. CDSAT for Nondisjoint Theories with Shared Predicates. We extended CDSAT to *predicate-sharing unions*, that is, unions of theories that are *either disjoint or share only predicate symbols* (in addition to equality) [35, 1]. Consider a *theory of arrays with length*, where extensionality says that two arrays are equal if they have the same length n and the same values at all indices between 0 and $n-1$. This axiom involves symbols from linear integer arithmetic (LIA), so that the two theories are nondisjoint. Also, such an axiomatization forces the indices to be integers. We proposed a *theory of arrays with abstract length* [35], then renamed *theory of arrays with abstract domain* [1], where the notion of an index being within bounds is abstracted into that of an index being *admissible*. In this theory, extensionality says that two arrays are equal if they have the same length and the same values at all admissible indices. Indices do not have to be integers, neither do they need to form a linear order. This approach covers several instances, including one where length and equality of arrays involve the starting address in memory, as it is common in programming languages. Admissibility is a *shared predicate*: it appears as a free symbol in the theory of arrays, and in another theory (e.g. LIA, but not necessarily), that defines it. This motivates the extension of CDSAT to predicate-sharing theories. The only definitions of the CDSAT framework that had to be generalized to accommodate shared predicates are *relevance* (of a term to a theory for the purpose of decisions) and *leading-theory completeness*, which shows the flexibility of the framework. We gave a theory module for the theory of arrays with abstract domain, we proved that it is leading-theory complete, and that CDSAT is *complete for predicate-sharing unions* [35, 1]. For termination, we generalized the construction of the finite global basis from the local ones to the predicate-sharing case [1]. We are working on *maps with abstract domain* and *vectors with abstract domain* (vectors are *dynamic arrays*), defining their presentations and their modules, and showing that they are leading-theory complete [1].

A7. The QSMA Algorithm for Quantifiers in SMT

Many SMT methods assume that the problem is quantifier-free (QF), because only the QF fragment of most theories is decidable. However, theory symbols and quantifiers occur together in problems from applications. We approached this issue in the case of one theory with unique model (e.g., LIA and LRA). We presented an algorithm named QSMA for *quantified satisfiability modulo theory and assignment* (of values to the free variables in the input formula) [33, 32, 2].

A7.1. The QSMA Framework. QSMA accepts formulæ with arbitrary alternation of quantifiers in arbitrary positions. $\forall$-quantifiers are converted into $\exists$-ones by double negation. The input formula φ is represented as a QSMA-*tree* $\mathcal{G}$, where each node n is labeled by a tuple $n.\bar{x}$ of free variables and a QF formula $n.F$. The variables are free, because their block of $\exists$ is removed, and the formula is QF because additionally its quantified subformulas are replaced by Boolean proxy variables $n.\bar{p}$ that label the arcs to the subformulas subtrees. Thus, $n.\bar{x}$ and $n.\bar{p}$ are n's *assignable variables.* Conversely, the *formula at a node* $n.\psi$ is obtained by putting back the block of $\exists$ and plugging in the formulas at n's children in place of the $n.\bar{p}$, in such a way that $r.\psi = \varphi$ for r the root of $\mathcal{G}$. Satisfying $\mathcal{G}$ satisfies φ, and $\mathcal{G}$ is satisfied by finding an assignment to r's assignable variables that satisfies $r.F$ and satisfies a subtree iff it assigns true to the corresponding Boolean variable. Therefore, the assignment to a Boolean proxy variable defines the mission of the algorithm when it recurses on a subtree. QSMA is designed to be built on top of an underlying solver for the QF fragment of the theory. The solver must provide a *model extension* function SMA (extend the current assignment to a node's assignable variables to satisfy a given formula), a *model-based under-approximation* function MBU (compute an implicant of the given formula that is true in the given assignment), and a *model-based over-approximation* function MBO (compute a formula implied by the given formula that is false in the given assignment) [33, 2].

A7.2. The QSMA Algorithm and its Optimization OptiQSMA. For all nodes n of a QSMA-tree, the QSMA algorithm maintains a QF formula $n.U$ ($n.O$), that is an under-approximation (over-approximation) of $n.\psi$. Picture a class of models as a bubble: the $n.U$ bubble is inside the $n.\psi$ bubble, which is inside the $n.O$ bubble. Therefore, if $n.U$ is true in the current assignment, the algorithm can return that $n.\psi$ is true, and if $\neg n.O$ is true in the current assignment, the algorithm can return that $n.\psi$ is false. Otherwise, the algorithm calls SMA on a formula L that is a necessary condition for $\mathcal{G}$'s satisfaction. If SMA signals that no satisfying assignment exists, it is safe to return false. Prior to that, QSMA calls MBO on L and uses the outcome to strengthen $n.O$. If SMA returns a satisfying assignment, QSMA recurses on n's subtrees. If all recursive calls succeeds, QSMA builds a formula L' that is a sufficient condition for $\mathcal{G}$'s satisfaction, so that it is safe to return true. Prior to that, QSMA calls MBU on L' and uses the outcome to weaken $n.U$. Weakening $n.U$ inflates the $n.U$ bubble and strengthening $n.O$ deflates the $n.O$ bubble, compressing the $n.\psi$ bubble from below and from above. Thus, the QSMA algorithm gradually zooms in on a model of $n.\psi$ or finds that none exists. We proved that QSMA is totally correct, assuming for termination that MBU and MBO have *finite basis* (they generate finitely many formulas for a given formula) [33, 2]. We also presented optiQSMA, an optimized version of QSMA. optiQSMA features a *look-ahead* mechanism that avoids making a recursive call on each subtree. By passing to the SMA function a *look-ahead formula*, it is possible to avoid recursing on *no-alternation-nodes* (descendants on paths where all Boolean proxy variables are assigned true) and limit recursion to *first-alternation-nodes* (the first descendants where the Boolean proxy variables are assigned false). We proved that satisfaction with look-ahead is the same as satisfaction, and the optimization preserves total correctness. optiQSMA also saves memory by storing only $n.U$ formulas, whereas $n.O$ formulas are generated and used, but not stored. optiQSMA is implemented in the YicesQS solver built on top of the Yices 2 solver, and it is a major ingredient of YicesQS excellent performance in the arithmetic division of the SMT-COMP competition [33, 2].

B. Interpolation of Proofs

Given disjoint sets of clauses A and B, such that $A \cup B$ is inconsistent, a (reverse) *interpolant* of (A, B) is a formula implied by A, inconsistent with B, and such that its uninterpreted symbols are common to A and B. If B encodes a partial model, an interpolant is a candidate *explanation* of why A is in conflict with B. Therefore, interpolation is relevant to conflict-driven satisfiability procedures [74, 39, 86, 31] as well as to abstraction refinement and invariant generation [46].

B1. Interpolation of Ground Superposition Proofs. A complete *interpolation system* for an inference system Γ extracts an interpolant of (A, B) from any Γ-refutation of $A \cup B$. It attaches a *partial interpolant* to every clause in the refutation, in such a way that the partial interpolant of $\Box$ is an interpolant of (A, B). For each inference rule the partial interpolant of the conclusion is defined from those of the premises. In a proof by propositional resolution (hence CDCL), all literals are input literals, hence the atom of the literal resolved upon is either an *A-symbol* (occurring only in A), or a *B-symbol* (occurring only in B), or a *common* symbol. The partial interpolant of the resolvent is defined based on this case analysis [45, 10]. A first-order proof with equality, even in the ground case, contains *new* literals, which may mix A-symbols and B-symbols, jeopardizing the case analysis. We gave sufficient conditions to ensure that ground superposition proofs do not contain such literals, without losing refutational completeness [10], and we designed the *first complete interpolation system* for *ground* refutations by superposition [79, 10].

B2. Interpolation of Non-Ground Proofs by Superposition and CDCL(Γ+$\mathcal{T}$). In non-ground proofs, literals mixing A-symbols and B-symbols are unavoidable, because substitutions mix symbols. Therefore, we worked with a *two-stage approach* [89, 11]. A *provisional interpolation system* computes a *provisional interpolant*, that is entailed by A and inconsistent with B. Then *lifting* replaces constants that are either A-symbols or B-symbols with quantified variables. We defined a *complete provisional interpolation system* for superposition, and we proved that the lifting of a provisional interpolant is an interpolant, obtaining the *first complete interpolation system* for non-ground superposition refutations, under the assumption that the only A-symbols or B-symbols in the provisional interpolant are constants [11]. The two-stage approach can interpolate refutations by equality sharing, CDCL($\mathcal{T}$), and CDCL(Γ+$\mathcal{T}$). A provisional interpolation system for CDCL($\Gamma + \mathcal{T}$) is obtained by combining those for superposition and for CDCL($\mathcal{T}$). Then, under the above assumption, lifting yields interpolants for CDCL($\Gamma + \mathcal{T}$)-refutations.

C. Distributed Automated Deduction

I was the first one to investigate *distributed automated deduction* [108, 66, 64, 63, 28, 62, 29, 60, 61, 25, 27, 97, 22, 23, 57, 56, 84, 83, 18, 54, 73]. I analyzed the parallelizability of theorem-proving strategies, classifying types of parallelism based on the granularity of data accessed in parallel: *fine-grain parallelism* is *parallelism at the term/literal level*, *medium-grain parallelism* is *parallelism at the clause level*, and *coarse-grain parallelism* is *parallelism at the search level* [108, 29, 83, 18, 73]. Parallelism at the term/literal level affects operations *below the inference or clause level* (e.g., *parallel rewriting*), but it requires clause preprocessing, which is problematic in theorem proving, where new clauses are generated. Parallelism at the clause level yields *parallel inferences*, but it is at odd with *eager contraction*, as priority to contraction reduces the concurrency of inferences. The possibility of *conflicts* between parallel inferences is an obstacle especially for *contraction-based* strategies, where *backward contraction*, the contraction of pre-existing clauses by new ones, applies to clauses active as premises of expansion inferences. Thus, I proposed *parallelism at the search level* by *Clause-Diffusion* [108, 66, 64, 63, 28, 62, 29, 25, 27].

C1. The Clause-Diffusion Method. In Clause-Diffusion, multiple deductive processes *search in parallel* the space of the problem and cooperate to seek a proof [108, 25, 27, 60, 97, 22, 73]. All processes start with the same input problem, ordering-based inference system, and search plan, although different search plans may be assigned. Every process develops *its own derivation* and builds *its own database of clauses* independently. The processes are *asynchronous*, as the only synchronization occurs when one sends all others a halting message because it found a proof. Clause-Diffusion is a *distributed-search* method, because it *subdivides the search space* by subdividing clauses and inferences. In an ordering-based strategy, a newly generated clause φ is subject to *forward contraction*, the contraction of new clauses by pre-existing ones. If the resulting normal form $\varphi\downarrow$ is not trivial, it is kept. In Clause-Diffusion, whenever a process generates and keeps a clause, it assigns it to a process, possibly to itself, by an *allocation criterion.* Every clause is *owned* by *only one* process (every clause has its own variables, and variants are distinct clauses). I designed several *heuristic allocation criteria* [56, 73]. Then, expansion inferences are subdivided based on ownership of the premises: for example, a process paramodulates only into the clauses it owns. Backward contraction inferences that generate clauses are subdivided without delaying the deletion of redundant clauses: whenever a process detects that a clause φ can be backward-simplified, it deletes it, but generates $\varphi\downarrow$ only if it owns φ. Every kept clause, regardless of whether generated by expansion or backward contraction, is given a *unique global identifier* and is broadcast as an *inference message*, hence the name of the method. I defined *fairness of distributed derivations*, giving sufficient conditions and showing that Clause-Diffusion satisfies them [108, 64, 25]. Thus, if the inference system is refutationally complete and the search plan at each process is fair, parallelization by Clause-Diffusion preserves *completeness.* I discovered that *subsumption in distributed derivations* may violate fairness and the soundness of contraction, and I gave a solution that preserves these properties and allows subsumption [108, 28]. Clause-Diffusion also achieves *distributed global contraction* and *distributed proof reconstruction* [22, 73]. The former property ensures that if φ is globally redundant at some stage of the distributed derivation, φ is recognized redundant eventually by every process. The latter property ensures that the process that generates □ is able to reconstruct the proof from the final state of its database, even if all processes contributed to the proof. I found sufficient conditions for this property, and proved that Clause-Diffusion fulfills them, with neither centralized control nor ad hoc postprocessing [22].

C3. The Clause-Diffusion Provers and Super-Linear Speed-Up. I implemented several *Clause-Diffusion theorem provers*. *Aquarius* parallelized Bill McCune's OTTER 2.2 theorem prover for FOL [66, 108, 63, 27]. *Peers* was built on top of a prototype prover (from Bill's Otter Parts Store) for equational theories modulo associativity and commutativity (AC) [62, 25]. *Peers-mcd* implemented *Modified Clause-Diffusion* [22], the final version of the methodology. *Peers-mcd.a* [22] had the same sequential basis as Peers. All subsequent versions parallelized Bill's EQP prover for equational theories modulo AC. EQP became famous in 1996 for proving that *Robbins algebras are Boolean*, a conjecture open since 1933. In most experiments, at least one allocation criterion allowed *Peers-mcd.b* to speed-up over EQP's best performance, with *super-linear speed-up* in two thirds of the proof of the Robbins theorem [57, 56]. Clause-Diffusion enables super-linear speed-up, because it does not compute in parallel the sequential search, but it uses distributed search to generate different searches. Peers-mcd.b also generated the *first mechanical proof* of the *Levi commutator problem* [84]. The proof of the Robbins theorem by *Peers-mcd.c* was *the fastest* at the time [18]. *Peers-mcd.d* [54] featured both *distributed search* and *multi-search*, where the processes apply different search plans, including *target-oriented heuristics* [106, 69, 65]. Peers-mcd.d offered *distributed-search* strategies (search space subdivided and same search plan for all processes), *multi-search* strategies (no subdivision and different search plans), and *hybrid* ones (subdivision and different search plans). I investigated whether Peers-mcd.d could prove the *Moufang identities* without building cancellation laws in the inference system. With some strategies, EQP could not find a proof while Peers-mcd.d did. With others, EQP succeeded, but Peers-mcd.d was faster, with instances of *super-linear speed-up*. Distributed search behaved better than multi-search, which did not find the proofs, and their hybridization performed even better [54].

C4. PSATO. Clause-Diffusion inspired *PSATO*, the first distributed-search SAT-solver with a *divide-and-conquer* organization [61, 23, 73]. A master process *partitions* the search space, by assigning *disjoint subproblems* to slave processes, each executing the Davis-Putnam-Logemann-Loveland procedure for SAT. Each subproblem is defined by a *guiding path*, which encodes a Boolean assignment. Thus, every subproblem is a Boolean instance of SMA. A guiding path can be read as a conjunction of literals, later called a *cube*, so that PSATO is an ancestor of the *cube-and-conquer* approach to parallel CDCL-based SAT solving. PSATO solved *quasigroup existence problems*, including some that were never conquered before [61, 23].

D. Strategy Analysis

Theorem-proving strategies are evaluated by comparing the performances of their implementations. Worst-case or average-case complexity analyses do not apply, as theorem-proving strategies are only semidecision procedures. The search space is infinite, and the complexity of searching for a proof is proportional to neither input nor output size [96, 95, 105]. Explanations such as "contraction helps by pruning the search space" are only intuitive. Therefore, I investigated formal tools for *strategy analysis* [94, 58, 93, 85, 92, 55, 20, 19, 17].

D1. Analysis of Ordering-Based Strategies. I introduced the *marked search-graph* as a *model of the search space* and *search process* that covers both expansion and contraction inferences [94, 58, 93, 20]. In this model, the *search graph* represents the space of all possible inferences, and the *marking* represents the *search process*, with generations and deletions of clauses. At each stage of a derivation a finite portion of the search space has been generated (the *present*) and an infinite portion remains to be explored (the *future*). While a strategy works with a finite amount of data, capturing the complexity of a search problem requires to measure changes *in both present and future*, since that finite amount of data can generate anything in the future. Since the latter is infinite, one needs to impose a *bound.* I introduced the notion of *bounded search space*, modeling the infinite search space as an *infinite succession* of bounded search spaces. Since they are finite, the bounded search spaces can be compared by a well-founded ordering on multisets of clauses or proofs. I analyzed *contraction-based strategies of different contraction power*, showing that more contraction eventually causes a bigger reduction of the bounded search spaces [58, 20].

D2. Analysis of Distributed Ordering-Based Strategies. In *distributed search* multiple processes are active in parallel. I devised the *parallel marked search-graph* to model also *subdivision of search space*, *communication*, and *overlap* among the processes [85, 92, 55, 19]. I defined bounded search spaces relative to each process, taking the subdivision scheme into account, and then *parallel bounded search spaces*, which capture also the overlap. Subdivision and contraction make the bounded search spaces smaller, but communication undoes in part this impact. I compared a distributed-search contraction-based strategy with its sequential basis, and analyzed the overlaps due to inaccurate subdivision and to communication, giving *sufficient conditions* to avoid the first and minimize the second. Then, I discovered two patterns of worst-case behavior, called *late contraction* and *contraction undone*, where the interaction of asynchronous communication and contraction violates eager contraction. It follows that sufficient conditions for the parallel bounded search spaces to be smaller than the sequential one are minimum overlap and immediate propagation of clauses. Although these conditions cannot be weakened, they are *not necessary*, and hence distributed-search contraction-based strategies may behave well in practice, and even exhibit *super-linear speed-up's* [56], approximating the ideal behavior in the analysis.

D3. Analysis of Subgoal-Reduction Strategies. I defined *marked search-graphs* also for linear resolution, and *clausal normal form tableaux*, including *model-elimination tableaux.* In these marked search-graphs, the marking captures the application of substitutions to *rigid variables*, hence to the whole tableau, the closure of branches, and the effects of backtracking [17]. The *bounded search spaces* are *multisets of partial interpretations*, which can be compared by comparing their *cardinalities*, coherently with strategies that survey and eliminate candidate models. I analyzed tableau-based strategies with and without the *regularity check*, that prevents the repetition of literals on a branch, and with and without *lemmaizing by folding-up*, showing that both refinements reduce the bounded search spaces [17].