From parallel theorem proving to parallel SAT-solving and back

Maria Paola Bonacina

Dipartimento di Informatica
Università degli Studi di Verona
Verona, Italy, EU

6 August 2017

Overview of this talk

- Parallel constraint reasoning (PCR)
- Learning from past work: selected key ideas in parallel automated theorem proving (ATP)
- Drawing connections between parallel ATP and parallel SAT-solving
- Abstracting common concepts: e.g., proof reconstruction

Theorem-proving strategies

A classical taxonomy:

- Ordering-based strategies
  ordered resolution, subsumption, superposition, simplification, ...
- Subgoal-reduction strategies
  e.g., linear resolution, model elimination (ME), ME-tableaux
- Instance-based strategies
  e.g., hyperlinking, inst-gen

In this talk: only ordering-based strategies, see the survey for the others

Expansion and contraction

Like many search procedures, most reasoning methods combine various forms of growing and shrinking:

- Recall CDCL in SAT/SMT: decisions and propagations grow the trail while backjumps shrink it
- Ordering-based strategies: expansion and contraction of a set of clauses
- Well-founded ordering $\succ$ to restrict expansion and define contraction: redundancy

Expansion

An inference

$$\frac{A}{B}$$

where A and B are sets of clauses is an expansion inference if

- $A \subset B$: something is added
- Hence $A \prec B$
- Soundness of expansion: what is added is a logical consequence of what was already there
  $B \setminus A \subseteq Th(A)$ hence $B \subseteq Th(A)$ hence $Th(B) \subseteq Th(A)$

Contraction

An inference

$$\frac{A}{B}$$

where A and B are sets of clauses is a contraction inference if

- $A \not\subseteq B$: something is deleted or replaced
- $B \prec A$: if replaced, replaced by something smaller
- Soundness of contraction, called adequacy: what is gone is logical consequence of what is kept
  $A \setminus B \subseteq Th(B)$ hence $A \subseteq Th(B)$ hence $Th(A) \subseteq Th(B)$ (monotonicity)

Derivation

- Input set S
- Inference system: a set of inference rules
- Derivation: $S = S_0 \vdash S_1 \vdash \ldots S_i \vdash S_{i+1} \vdash \ldots$
  $\forall i$ S_{i+1} is derived from S_i by an inference
- Refutation: a derivation such that $\square \in S_k$ for some k
- Refutational completeness: for all unsatisfiable S there is a refutation
- Persistent clauses: $S_\infty = \bigcup_{i \geq 0} \bigcap_{j \geq i} S_j$
- Once redundant always redundant

Ordering-based inference system

- Expansion rules: ordered resolution, ordered factoring, superposition/ordered paramodulation, equational factoring, reflection (resolution with $x \simeq x$)
- Contraction rules: subsumption, simplification, tautology deletion, clausal simplification (unit resolution + subsumption)
- Refutationally complete

Search

- An inference system is non-deterministic
- Given input problem and inference system, many derivations are possible
- Which derivation will be built? Search problem

Theorem-proving strategy

- Theorem-proving strategy: inference system + search plan
- The search plan picks at every stage of the derivation which inference to do next
- A theorem-proving strategy is a deterministic procedure
- Refutationally complete inference system + fair search plan = complete theorem-proving strategy

Forward and backward contraction

- Eager-contraction search plan: contract before expanding
- Forward contraction:
  reduce new clause C by older clauses
  find all clauses D that can reduce C
- Backward contraction:
  reduce older clause D by new clause C
  find all clauses D that C can reduce

Search plans for ordering-based strategies

- Lists `To-Be-Selected` and `Already-Selected`
- Given-clause algorithm: select a given-clause C from `To-Be-Selected`, do all expansion inferences between C and all D in `Already-Selected`, move C to `Already-Selected`
- Apply forward contraction to each new clause
- Two versions for backward contraction:
  - Apply to both lists
  - Apply only to `Already-Selected`

Parallel inferences

- ROO: a parallel version of Otter [Lusk, McCune, Slaney 1992]
- `To-Be-Selected` and `Already-Selected` in shared memory
- Processes $p_0, \ldots, p_{n-1}$ select given-clauses and do expansion (including forward contraction) in parallel
- Conflicts arise if they try backward contraction in parallel
- Only one process for backward contraction
- Backward-contraction bottleneck

Problems with parallel inferences

- No read-only data: any clause can be contracted
- Highly dynamic set of generated and kept clauses
- Conflicts between parallel inferences:
  e.g., p_1 rewrites D by backward contraction
  p_2 reads D as expansion premise
- All due to backward-contraction
- Backward contraction indispensable to counter search space growth by expansion

From parallel inferences to parallel search

[Bonacina 1992] [Bonacina, Hsiang: JAR 1994]

Parallel search

- Parallel processes $p_0, \ldots, p_{n-1}$
- Each builds its own derivation and its own set of generated and kept clauses
- Success when one p_i finds a proof
- Communication
- Separate databases: no conflicts, no backward-contraction bottleneck
- Duplication harmless for soundness if inferences are sound

Parallel search

How to differentiate the searches of $p_0, \ldots, p_n$?

- Distributed search: subdivide the search space among the processes
- Multi-search: let the processes use different search plans
- The two may be combined
- General idea: Seek a proof by different searches hopefully faster than a sequential one

Distributed search: Clause Diffusion

- Subdivide the search space by subdividing clauses
- Distributed derivation:

$$(O_0; NO_0)^j \vdash (O_1; NO_1)^j \vdash \ldots (O_i; NO_i)^j \vdash \ldots$$

- $\forall p_j,\ 0 \leq j \leq n-1,\ \forall i,\ i \geq 0$:
  - O_i^j is the set of clauses owned by p_j
  - NO_i^j is the set of clauses not owned by p_j
  - $S_i^j = O_i^j \uplus NO_i^j$ is the local database of clauses at p_j
  - $\bigcup_{j=0}^{n-1} S_i^j$ is the global database at stage i
  - $S_0^0 = S_0^1 = \ldots = S_0^{n-1} = S$ is the input set of clauses
  - Every clause is owned by a process: $\bigcup_{j=0}^{n-1} O_i^j = \bigcup_{j=0}^{n-1} S_i^j$
    And only one: $O_i^j \cap O_i^k = \emptyset$ (exceptions in practice)

[Bonacina 1992] [Bonacina, Hsiang: FI 1995] [Bonacina: JSC 1996]

Subdivision and diffusion of clauses I

- p_j generates C by expansion or backward contraction
- Forward contraction: $D = C\downarrow$
- p_j determines owner p_k of D by an allocation criterion
- D's id: $\langle k, m, j\rangle$ globally unique
- $k = j$: D enters O^j
- $k \neq j$: D enters NO^j
- p_j applies D to backward-contract clauses in S^j
- p_j broadcasts inference message $\langle D, k, m, j\rangle$

Subdivision and diffusion of clauses II

- p_q, $q \neq j$, receives $\langle D, k, m, j \rangle$
- Forward contraction: $E = D\downarrow$
- $k = q$: E enters O^q
- $k \neq q$: E enters NO^q
- p_q applies E to backward-contract clauses in S^q

Clause Diffusion: allocation criteria

- Round-robin
- Work-load based
- Syntax-based: weight-based
- Ancestor-graph oriented (AGO): heuristics to try to minimize search overlap, e.g.:
  - Assign C to the process that owns the most of its ancestors

Clause Diffusion: subdivision of inferences

- No subdivision of forward-contraction inferences
- No subdivision of backward-contraction inferences that delete clauses (e.g., subsumption)
- Subdivision of expansion inferences:
  p_j performs the inference if it owns the clause paramodulated or superposed into or the negative-literal parent in resolution
- Subdivision of backward-contraction inferences that simplify clauses: $C \in S^j$ can backward-simplify $D \in S^j$:
  p_j generates $D\downarrow$ if it owns D, only deletes D otherwise

Properties

- Distributed fairness: local fairness +
  broadcast eventually all persistent irredundant clauses
- Distributed proof reconstruction:
  - Proof reconstruction: save the clauses deleted by backward contraction
  - Broadcast eventually all clauses ever used as premises
- Distributed global contraction: if C redundant in $\bigcup_{j=0}^{n-1} S_i^j$,
  $\forall p_j \; \exists l \geq i$ such that C redundant in S_l^j
  All delete C and one generates $C\downarrow$

[Bonacina, Hsiang: STACS 1993] [Bonacina: JSC 1996]

The Clause-Diffusion provers I

- Aquarius: parallelization of McCune's Otter 2.2, PCN for message passing, also multi-search
  [Bonacina 1992] [Bonacina, Hsiang: DISCO 1993]
  [Bonacina, Hsiang: JSC 1995]
- Peers: parallelization of code from the Otter Parts Store, equational theories possibly with AC function symbols, p4 for message passing, the pairs algorithm
  [Bonacina, McCune: CADE 1994] [Bonacina, Hsiang: FI 1995]

The Clause-Diffusion provers II

- Peers-mcd.a [Bonacina: JSC 1996]
- Peers-mcd.b: parallelization of McCune's EQP 0.9, equational theories possibly with AC function symbols, also blocking and basic paramodulation, MPI for message passing, AGO allocation criteria both given-clause and pairs algorithms
  [Bonacina: CADE 1997] [Bonacina: PASCO 1997]
  [Bonacina: CADE ws 1998]
- Peers-mcd.c: parallelization of EQP 0.9d [Bonacina: AMAI 2000]

The first big proof: the Robbins theorem

- The Robbins conjecture: Robbins algebra are Boolean
  open in mathematics since 1933
  a challenge for theorem provers since 1990
- EQP 0.9 proved the Robbins conjecture in 1996
  [McCune: JAR 1997]
- Peers-mcd.b proved it with super-linear speedup in two out of three parts of the proof [Bonacina: PASCO 1997]
- Peers-mcd.c proved it with super-linear speedup in two out of three parts of the proof and almost super-linear speedup in the third [Bonacina: AMAI 2000]

The Clause-Diffusion provers III

- Peers-mcd.d: both distributed search and multi-search, distributed mode, multi-search mode, hybrid mode
- Different search plans: given-clause and pairs, different heuristic evaluation functions, different `pick-given-ratio`
- Moufang identities in alternative rings with cancellation laws built-in [Anantharaman, Hsiang: JSC 1990]
- Peers-mcd.d proved them without cancellation laws, with super-linear speedup (w.r.t. EQP0.9d) in distributed and hybrid mode with hybrid doing best

[Bonacina: IJCAR 2001]

Clause Diffusion: Summary

- Pioneered distributed search for ATP
- Inspired PSATO for the idea of subdividing work; master-slave organization in place of peers, guiding-paths [Zhang, Bonacina: PASCO 1994] [Zhang, Bonacina, Hsiang: JSC 1996] [Zhang: CADE 1997] [Zhang, Stickel: JAR 2000]
- Ancestor of divide-and-conquer in parallel constraint reasoning

Multi-search: Team-Work

- Interleave search plans
- Combine search plans by communicating good clauses
- Pioneered multi-search for ATP
- Ancestor of the portfolio approach in parallel constraint reasoning
- Ancestor of machine-learning-inspired approaches to ATP

[Denzinger 1993] [Denzinger, Schulz: JSC 1996]
[Denzinger, Fuchs, Fuchs: IJCAI 1997]

Parallel ATP and parallel SAT-solving

- Parallel search
- Distributed search
  - SAT: partition of the search space
  - ATP: subdivision with heuristics to limit overlap
- From DPLL to CDCL: clause learning, communicating good learned clauses, no analogue of backward contraction
- Cube-and-conquer as an instance of satisfiability modulo assignment

Future: parallelism and model-based ATP?

- Strategies that hybridize tableaux and instance-generation
- Semantically-guided strategies
- Model-based strategies
- Conflict-driven strategies

Current work: proof reconstruction in CDSAT

- Multiple reasoning engines appear also in sequential contexts
- CDSAT is a new method for theory combination
- Key abstraction: CDSAT combines inference systems called theory modules $\mathcal{I}_1, \ldots, \mathcal{I}_n$ for disjoint theories $\mathcal{T}_1, \ldots, \mathcal{T}_n$
- CDSAT solves the problem of combining multiple conflict-driven $\mathcal{T}_k$-satisfiability procedures into a conflict-driven $\mathcal{T}$-satisfiability procedure for $\mathcal{T} = \bigcup_{k=1}^{n} \mathcal{T}_k$
- CDSAT generalizes conflict-driven reasoning to generic combinations

Reference

Maria Paola Bonacina. Parallel theorem proving.
In Youssef Hamadi and Lakhdar Sais (Editors)
Handbook of Parallel Constraint Reasoning
Springer, Lecture Notes in Computer Science, volume in press,
Chapter 6, pages 177–233, 2017 [providing 230 references]