Beagle -
A Hierarchic Superposition Theorem Prover

Peter Baumgartner
Joshua Bax
Uwe Waldmann

Introduction

Goal

Automated deduction in hierarchic combinations of specifications

Previous work: calculus

Hierarchic superposition [BachmairGanzingerWaldmann94]

Hierarchic superposition with weak abstraction [BW14]

This work: implementation

Beagle theorem prover

This talk

HSPWA summary

Beagle design and features

Experiments

Hierarchic Specifications

Background (BG) specification consists of

Sorts, e.g., $\{$ int $\}$

Operators, e.g., $\{$ 0, 1, -1, 2, -2, ..., α_1, α_2, ..., -, $+$, $>$, $\approx$ $\}$

Models, e.g., linear integer arithmetic (LIA)

Foreground (FG) specification extends BG specification by

New sorts, e.g., $\{$ list $\}$

New operators, e.g.,

$\{$ cons: int $\times$ list $\mapsto$ list, empty: list, length: list $\mapsto$ int, a: list $\}$

First-order clauses, e.g.,

$\{$ length(a) $\geq$ 1, length(cons(x, y)) $\approx$ length(y) $+$ 1 $\}$

Deduction problem

Check whether a given clause set N has a hierarchic model, i.e., a model that extends one of the models of the BG specification

Hierarchic Superposition

Superposition

Abstraction for pulling out certain BG terms t: $C[t] \rightarrow C[x] \vee x \not\approx t$

Superposition inference rules on FG literals of abstracted clauses

$$\text{Sup}\ \frac{l \approx r \vee C \qquad s[u] \not\approx t \vee D}{(s[r] \not\approx t \vee C \vee D)\sigma}$$

Interface to BG reasoner

$$\text{Close}\ \frac{C_1 \cdots C_n}{\square}$$

E.g., $\text{Close}\ \frac{\alpha<0 \quad \alpha\approx 5}{\square}$

if C1,...,Cn are BG clauses and { C1,...,Cn } is BG-unsatisfiable

Simplification

Tautologies, subsumption, demodulation

Specific BG simplification see below

Hierarchic Superposition

Refutational completeness

Hierarchic superposition is refutationally complete for clauses sets N s.th.

N is (weakly) abstracted

N is sufficiently complete

BG specification is compact

Hierarchic Superposition

Two kinds of BG variables

Abstraction variables X: mapped only to BG terms

Ordinary variables x: mapped to BG terms or BG-sorted FG terms

Tradeoff sufficient completeness

$\{ \mathsf{length}(a) \not\approx X \}$ not sufficiently complete, no refutation

$\{ \mathsf{length}(a) \not\approx x \}$ sufficiently complete, refutation

Tradeoff search space

$\mathsf{length}(a) \approx X$ is ordered from left to right

$\mathsf{length}(a) \approx x$ is not ordered

Lemmas

$X + 0 \approx X$ is redundant

$x + 0 \approx x$ can be useful

Hierarchic Superposition

Define inference rule

Replaces a ground BG-sorted FG term by a fresh BG constant α

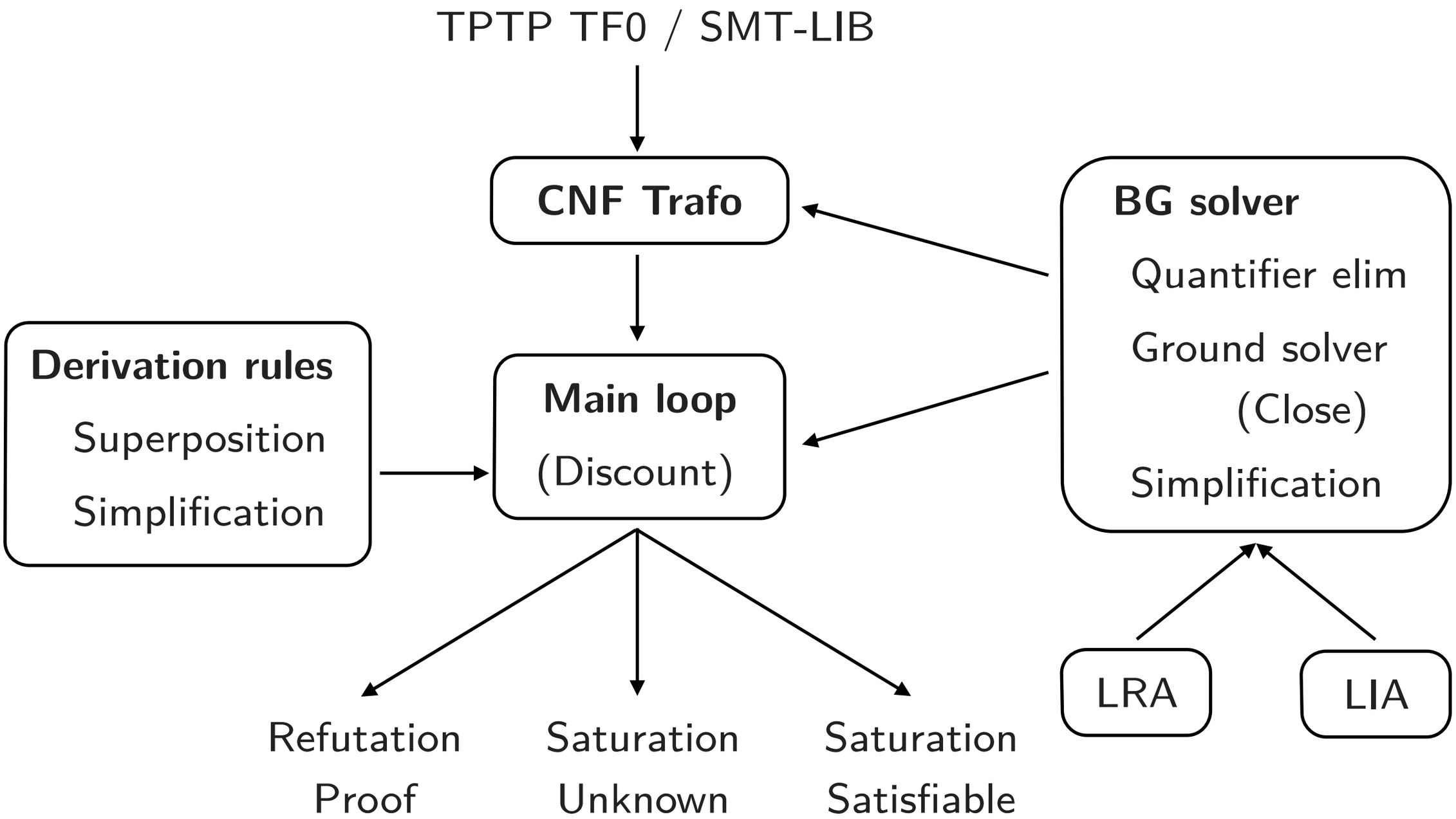
$$\text{Define}\ \frac{\text{length(a)} > 5}{\alpha > 5 \qquad \text{length(a)} \approx \alpha}$$

Purpose: establish sufficient completeness during derivations

Similar to preprocessing steps in [NelsonOppen79] and [KruglovWeidenbach12]

However in hierarchic superposition ground terms can show up in the middle of derivations, hence an inference rule

Beagle Structure



BG Solver

Quantifier elimination

During CNF transformation

$\forall\, x\ (P(x) \vee \exists\, y\ (x < y \wedge y < 3)) \quad \rightsquigarrow \quad \forall\, x\ (P(x) \vee x < 2)$

(better than $\forall\, x\ (P(x) \vee (x < f(x) \wedge f(x) < 3))$ by Skolemization)

During derivations

$\alpha < x \vee x < \beta \quad \rightsquigarrow \quad \alpha < \beta$ cached for BG ground solver calls

LIA: Cooper's algorithm

+ subsumption: $\{\ \alpha < 5,\ \alpha < 3,\ \ldots\ \} \rightsquigarrow \{\ \alpha < 3,\ \ldots\ \}$

+ resolution: $\{\ \ldots,\ s_i < \alpha,\ \ldots,\ \ldots,\ \alpha < t_j,\ \ldots,\ \ldots\ \} \rightsquigarrow \{\ \ldots,\ s_i + 1 < t_j,\ \ldots,\ \ldots\ \}$

LRA: Fourier-Motzkin

BG Solver

Ground solver

Implements the Close inference rule

Called whenever a new BG clause is derived

Primitive algorithm around it for determining minimal unsat core

LIA

Cooper's algorithm

OR

Z3 or CVC4

via SMT-LIB interface

Z3 provides unsat core natively

LRA

Simplex

BG Solver

BG Simplification

Two options: “cautious” or “aggressive”

Cautious BG simplification

Evaluation of arithmetic subterms

$$f(x)+(1+1) > f(x)+2 \quad \rightsquigarrow$$

$$f(x)+2 \quad > f(x)+2 \quad \rightsquigarrow$$

$$\mathsf{false}$$

Unabstraction of BG domain elements

$$C \vee x \not\approx 5 \quad \rightsquigarrow \quad C\{x \mapsto 5\}$$

Preserves sufficient completeness

However, for many problems “aggressive” simplification fares better

BG Solver

Aggressive BG simplification

Eliminate operators $>$, $\geq$ and $\leq$ in terms of $<$

BG-sorted subterms are brought into a polynomial-like form

$5 \cdot \alpha + f(3+6,\ \alpha \cdot 4) - \alpha \cdot 3 \quad \rightsquigarrow \quad 2 \cdot \alpha + f(9,\ 4 \cdot \alpha)$

Unique for pure BG formulas (modulo associativity of $+$)

Move around polynomials between lhs and rhs of (dis/in)equations

$s - t \approx u \ \rightsquigarrow\ s \approx u + t$ (eliminate -)

$\text{length}(a) + -5 \approx 0 \ \rightsquigarrow\ \text{length}(a) \approx 5$ (eliminate number)

Aggressive BG simplification may destroy sufficient completeness

$\{\ P(1 + (2 + f(x))), \neg P(1 + (x + f(x)))\ \}$ is sufficiently complete

$\{\ P(\quad 3 + f(x)\ ), \neg P(1 + (x + f(x)))\}$ is not sufficiently complete

However may also install sufficient completeness

Main Loop

Discount loop

I.e., set of unprocessed clauses is not interreduced

Split rule

Split clause into variable disjoint subsets

Alternatives e.g. never/only split BG subclauses

Dependency-directed backtracking

Fairness

weight-age-ratio n: select n lightest clauses, then an oldest one

Can also emphasise use of clauses derived from the conjecture

Auto mode

Aggressive simplification

Exhaustive splitting

First 50% of available time use abstraction variables, then ordinary variables

Implementation

Implementation language: Scala

Class hierarchy for terms and formulas, most data structures immutable

Parser library for TPTP TF0 input, SMTtoTPTP for SMT-LIB input

Primitive term indexing

Mapping $\{$ op $\mapsto$ pos, ... $\}$ for every op-subterm at every position pos

Used for superposition inferences and for demodulation

Scala specific features

Libraries: `List`, `Vector`, `Map`, `Set`, ...

Extensive use of very effective `lazy val` (deferred computation of values)

E.g. `lazy val maximalLits =` "some costly computation"

Often clause is deleted before `maximalLits` is accessed, so don't compute

Availability

GPL'ed source/jar at https://bitbucket.org/peba123/beagle

Experiments

TPTP

TPTP Version 6.1.0, MacBook Pro 2.3GHz Core i7, 16GB

Time limit 180 sec, auto strategy

"Theorem" problems by category

Category	ARI	DAT	GEG	HWV	MSC	NUM	PUZ	SEV	SWV	SWW	SYN	SYO
Total	539	103	5	88	2	43	1	6	2	177	1	3
Solved	531	98	5	0	2	41	1	2	2	97	0	2

HWV: too much combinatorial search - currently out of reach

DAT: many problems require ordinary variables (s.c. issue otherwise)

SWW: very sensitive to parameter settings, e.g., weight-age-ratio

Cooper vs Z3

Four configs: splitting BG subclauses on/off vs BG solver Cooper/Z3

Result: splitting BG subclauses on is almost always better

Result: Z3 or Cooper makes no difference (BG proof tasks too easy?)

Experiments

SMT-LIB

SMT-lib 2014, Difficulty ratings from SMT-comp 2014

Time limit 120 sec, auto strategy

Results

Logic	ALIA	QF	AUFLIA	QF	UFLIA	QF	UFIDL	QF	QF_IDL	QF_LIA
Total	41	72	4	516	6602	195	62	335	694	2610
Solved	31	40	4	205	1736	155	42	29	24	28

QF means QF_(previous category)

Skipped LIA as it only had TPTP problems

89 UFLIA/sledgehammer problems solved by Beagle, not by any SMT solver

1391 'trivial' rated problems not solved by Beagle

Hierarchic Superposition

(Weak) abstraction

Removes certain BG subterms from FG terms

$C[t] \quad \rightsquigarrow \quad C[X] \vee X \not\approx t$ if t is a pure BG term (only "abstraction" variables) and ...

$C[t] \quad \rightsquigarrow \quad C[x] \vee x \not\approx t$ if t is an impure BG term and ...

Goal is to remove as few BG subterms as possible, yet preserve s.c.

Weak abstraction examples

$\mathsf{cons}(2, \mathsf{empty})) \not\approx \mathsf{cons}(x + y, \mathsf{empty}) \rightsquigarrow$

$\mathsf{cons}(2, \mathsf{empty})) \not\approx \mathsf{cons}(z, \mathsf{empty}) \vee z \not\approx x + y$

$\mathsf{length}(\mathsf{cons}(x, y)) \approx \mathsf{length}(y) + 1$ is already weakly abstracted

(Inference rule conclusions may require weak abstraction)