Lemma Learning in the Model Evolution Calculus

Peter Baumgartner

NATIONAL ICT AUSTRALIA LIMITED

Alexander Fuchs

Cesare Tinelli

THE UNIVERSITY OF IOWA

Background – Instance Based Methods

- **Model Evolution** is a sound and complete calculus for first-order clausal logic
- Different to Resolution, Model Elimination,...
  (Pro's and Con's)
- Related to Instance Based Methods
  - Reduction of first-order (clausal) logic to propositional logic in an „intelligent“ way
  - [Ordered] [Semantic] Hyper Linking [Plaisted et al]
  - Inst-Gen [Ganzinger&Korovin]
  - Primal Partial Instantiation [Hooker et al]
  - Disconnection Method [Billon]
  - DCTP [Letz&Stenz]
  - Successor of First-Order DPLL [B.]

Model Evolution - Motivation

- The best modern SAT solvers (satz, MiniSat, zChaff) are based on the Davis-Putnam-Logemann-Loveland procedure [DPLL 1960-1963]
- **Can DPLL be lifted to the first-order level?**
  How to combine
  - successful DPLL techniques
    (unit propagation, backjumping, lemma learning,...)
  - successful first-order techniques?
    (unification, subsumption, ...)?
- Our approach: Model Evolution
  - Directly lifts DPLL. Not: DPLL as a subroutine
  - Satisfies additional desirable properties
    (proof confluence, model computation, ...)

Model Evolution - Achievements

- FDPLL [CADE-17]
  - Basic ideas, predecessor of ME
- ME Calculus [CADE-19]
  - Proper treatment of unit propagation
  - Semantically justified redundancy criteria
- ME+Equality [CADE-20]
  - Superposition inference rules
- Darwin prover [JAIT 2006]
  - Won CASC-21 EPR division
- FM-Darwin: finite model computation [DISPROVING-06]

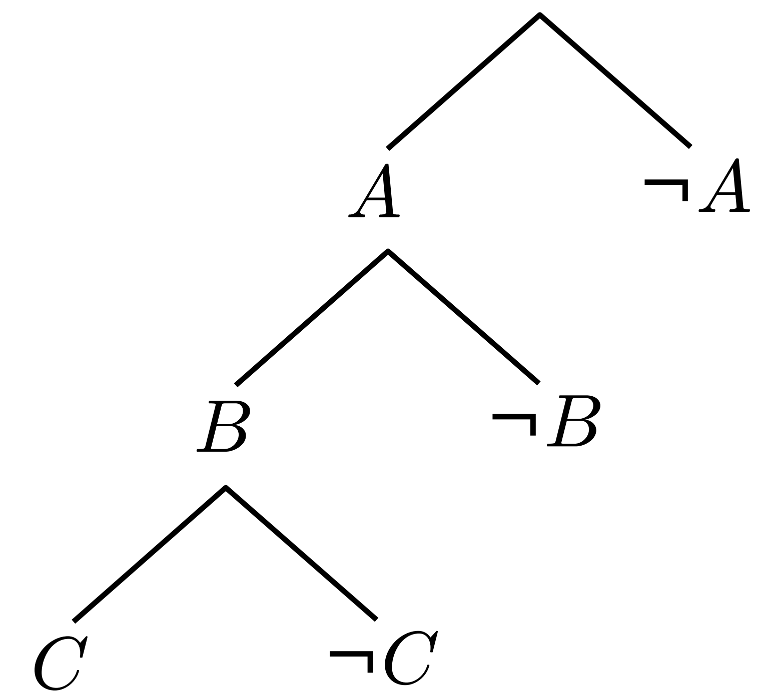
This work: extend ME and Darwin by "lemma learning"

Contents

- DPLL as a starting point for ME
- ME calculus idea
  - Model representation
- Lemma learning
  - Lemma learning in DPLL
  - Grounded lemma learning
  - Lifted lemma learning
  - Experiments

DPLL procedure

Input: Propositional clause set
Output: Model or „unsatisfiable"



Algorithm components:
- Propositional semantic tree enumerates interpretations
- Propagation
- Split
- Backjumping

$$\{A, B\} \overset{?}{\models} \cancel{\neg A} \lor \cancel{\neg B} \lor C \lor D \quad ✗$$

$$\{A, B, C\} \overset{?}{\models} \cancel{\neg A} \lor \cancel{\neg B} \lor C \lor D \quad ✓$$

ME - lifting this idea to first-order level

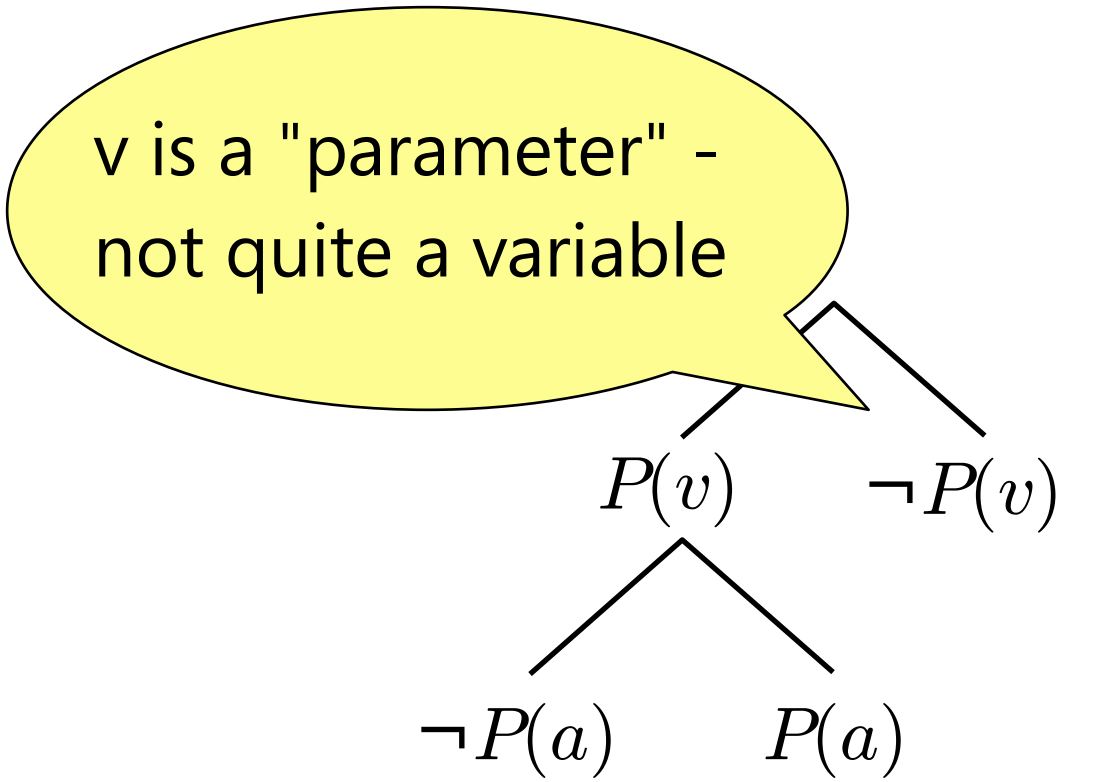
ME as First-Order DPLL

Input: First-order clause set

Output: Model or „unsatisfiable" if termination

Algorithm components:

- First-order semantic tree enumerates interpretations
- Propagation
- Split
- Backjumping

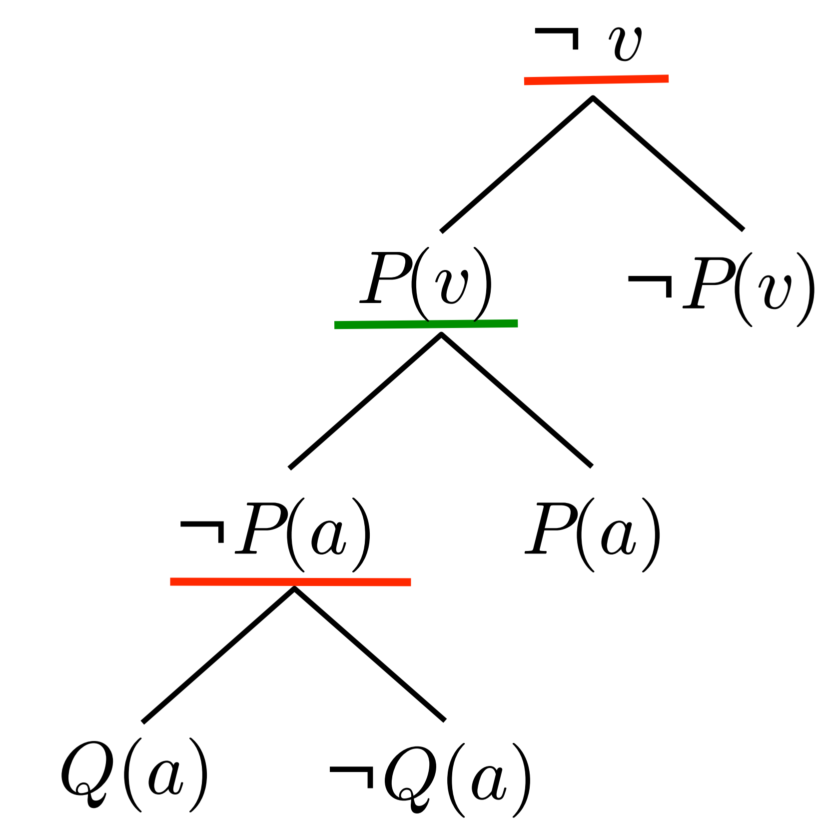


$$\{P(v), \neg P(a)\} \stackrel{?}{\models} P(x) \vee Q(x)$$

Interpretation induced by a branch?

Interpretation Induced by a Branch

A branch literal specifies the truth value of its ground instances unless a more specific branch literal specifies the opposite truth value



Branch: $\{\neg v, P(v), \neg P(a)\}$

True: $P(b)$

False: $\neg P(a)$, $\neg Q(a)$, $\neg Q(b)$

Branch: $\{\neg v, P(v), \neg P(a), Q(a)\}$

True: $P(b)$, $Q(a)$

False: $\neg P(a)$, $\neg Q(b)$

$\{\neg v, P(v), \neg P(a)\} \overset{?}{\models} P(x) \vee Q(x)$ ✗ —Context Unifier→ $P(a) \vee Q(a)$ (Split)

$\{\neg v, P(v), \neg P(a), Q(a)\} \overset{?}{\models} P(x) \vee Q(x)$ ✓

Lemma Learning in DPLL

"Avoid making the same mistake twice"

$\dots$

$B \vee \neg A$ (1)

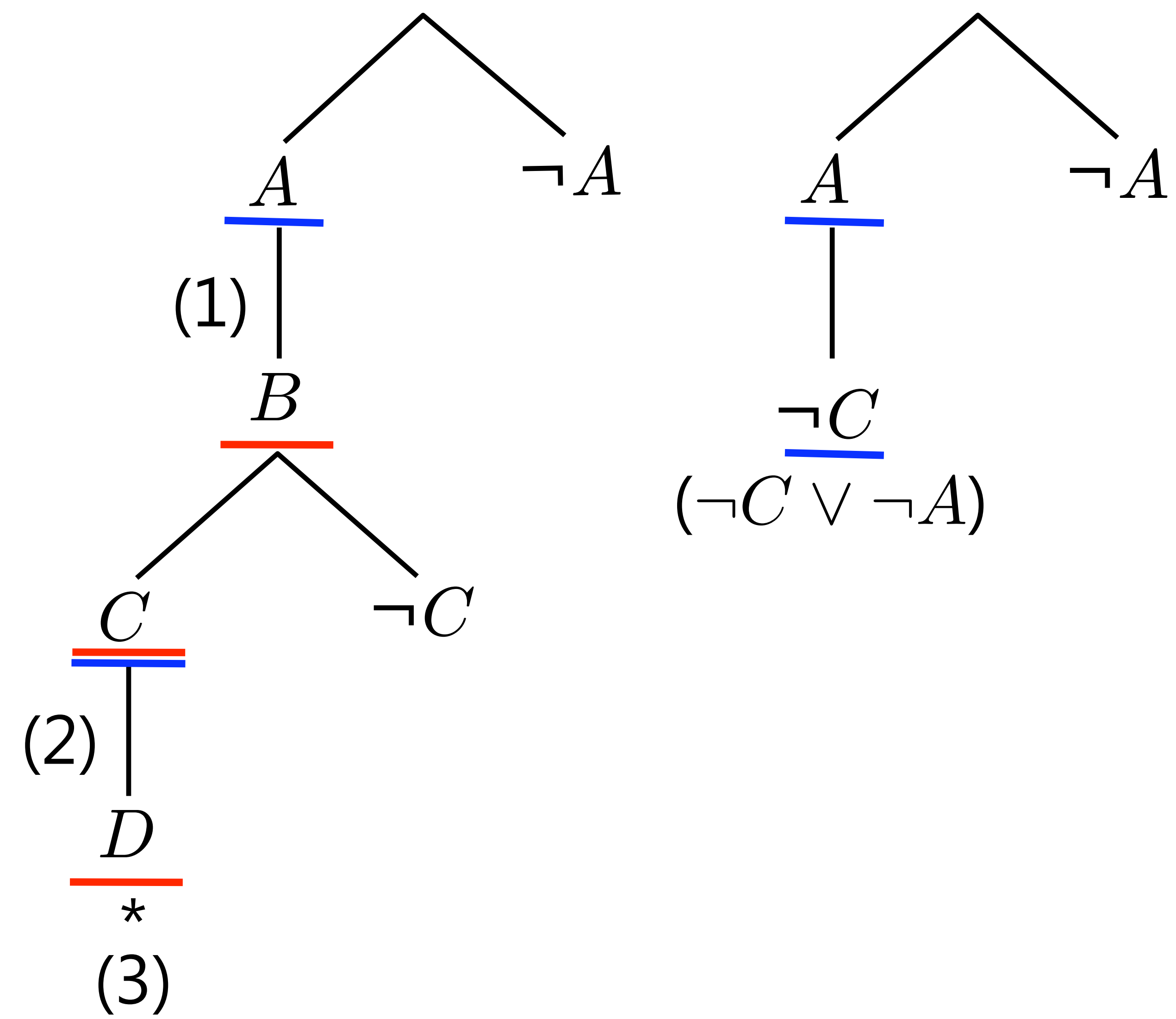
$D \vee \neg C$ (2)

$\underline{\neg D} \vee \underline{\neg B} \vee \underline{\neg C}$ (3)

Lemma Candidates by Resolution:

$$\frac{\dfrac{\underline{\neg D} \vee \neg B \vee \neg C \qquad \underline{D} \vee \neg C}{\underline{\neg B} \vee \neg C} \qquad \underline{B} \vee \neg A}{\underline{\neg C \vee \neg A}}$$

w/o Lemma **With Lemma**



Lemma Learning in DPLL

- **Soundness**
  - Can add **any** clause, provided it is entailed by input clause set
  - Example on previous slide indicates just one strategy
- **Benefits**
  - Can close branches earlier
  - Replace (nondeterministic) search by (deterministic) computation
- **Problem: (too) many redundant clauses**
  - Heuristics to delete lemma clauses
  - In practice regress only up to last split

Lifting to lemma learning in ME?

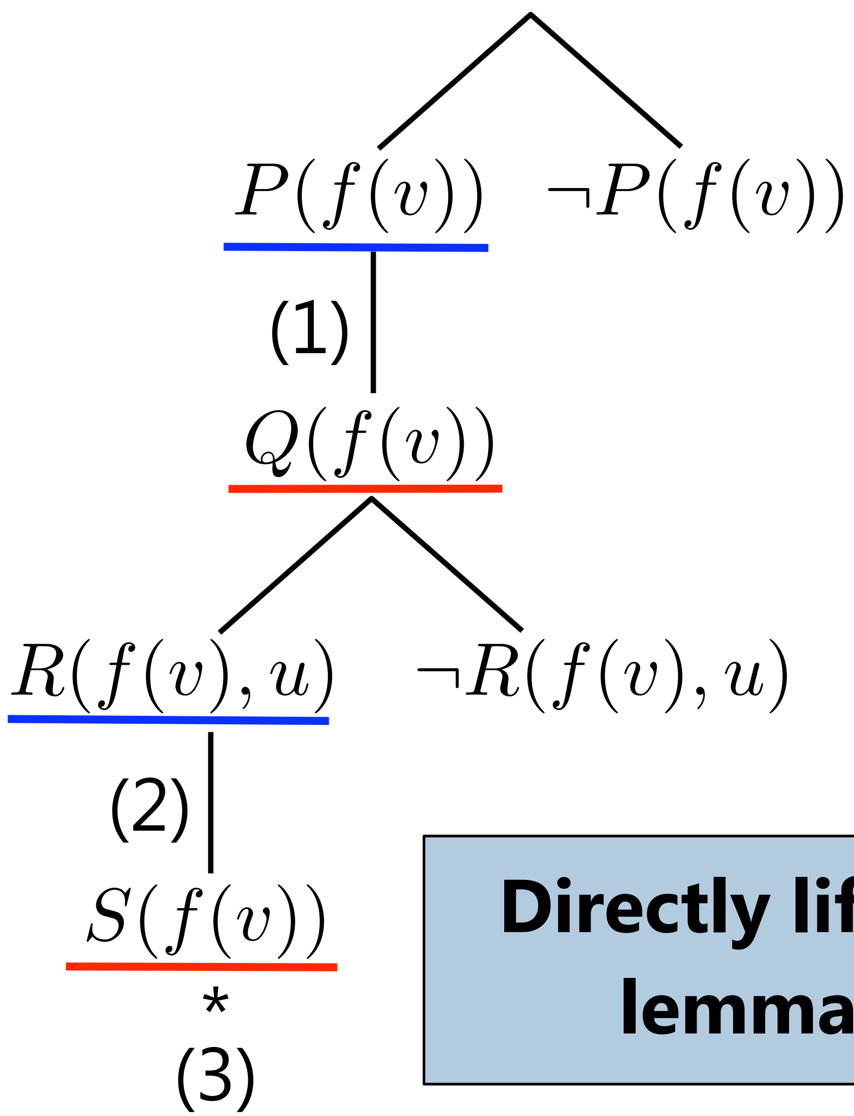
Lemma Learning in ME - Grounded Version

$\ldots$

$Q(x) \vee \neg P(x)$ (1)

$S(x) \vee \neg R(x, y)$ (2)

$\neg S(x) \vee \neg Q(x)$ (3)



"Avoid making the same mistake twice"

Lemma Candidates by Resolution:

$\neg S(f(v)) \vee \neg Q(f(v))$

↓ Skolemize

$$\frac{\neg S(f(c)) \vee \neg Q(f(c)) \qquad S(x) \vee \neg R(x, y)}{\neg Q(f(c)) \vee \neg R(f(c), y)}$$

↓ Skolemize

$$\frac{\neg Q(f(c)) \vee \neg R(f(c), d) \qquad Q(x) \vee \neg P(x)}{\neg P(f(c)) \vee \neg R(f(c), d)}$$

↓ De-Skolemize

$\neg P(f(x)) \vee \neg R(f(x), y)$

Directly lifts DPLL-syle lemma learning

Lemma Learning in ME - Lifted Version

Grounded Version

$$\neg S(f(v)) \vee \neg Q(f(v))$$

↓ Skolemize

$$\frac{\underline{\neg S(f(c))} \vee \neg Q(f(c)) \quad \underline{S(x)} \vee \neg R(x,y)}{\neg Q(f(c)) \vee \neg R(f(c),y)}$$

⋮

$$\neg P(f(x)) \vee \neg R(f(x),y)$$

Based on Skolemization/Matching

Less general/propagations/splits

Lifted Version

$$\frac{\neg S(x) \vee \neg Q(x) \quad \underline{S(x)} \vee \neg R(x,y)}{\neg Q(x) \vee \neg R(x,y)}$$

⋮

$$\neg P(x) \vee \neg R(x,y)$$

Based on Unification

More general/propagations/splits

Proposition: Regression of propagated literals is always possible

Does the lifted method perform better than the grounded one?

Experimental Evaluation

- Extended the Darwin prover by lemma learning
  - Grounded method
  - Lifted method
  - (And one more - see long version of paper)
- Experiments with TPTP (V. 3.1.1)
  - all non-Horn (clausal) problems without equality
- Setting
  - Xeon 2.4 GHz machines, 1 GB main memory, Linux
  - Timeout 300s
- Lemma learning can give spectacular speedups for propositional SAT

Does it work equally well in our case?
What method is better?

Darwin - TPTP Problems (1)

Method	Solved Probls	Avg Time	Total Time	Speed up	Failure Steps	Propag. Steps	Split Steps	Splits per Problem
no lemmas	896	2.7	2397.0	1.00	24991	597286	45074	
grounded	895	2.4	2135.6	1.12	9476	391189	18935	≥ 0
lifted	898	2.4	2173.4	1.10	9796	399525	19367	
no lemmas	244	3.0	713.9	1.00	24481	480046	40766	
grounded	243	1.8	445.1	1.60	8966	273849	14627	≥ 3
lifted	246	2.0	493.7	1.45	9286	282600	15059	
no lemmas	108	5.2	555.7	1.00	23553	435219	38079	
grounded	108	2.2	228.5	2.43	8231	228437	12279	≥ 20
lifted	111	2.6	274.4	2.02	8535	238103	12688	
no lemmas	66	5.0	323.9	1.00	21555	371145	34288	
grounded	67	1.7	111.4	2.91	6973	183292	9879	≥ 100
lifted	70	2.3	151.4	2.14	7275	193097	10294	

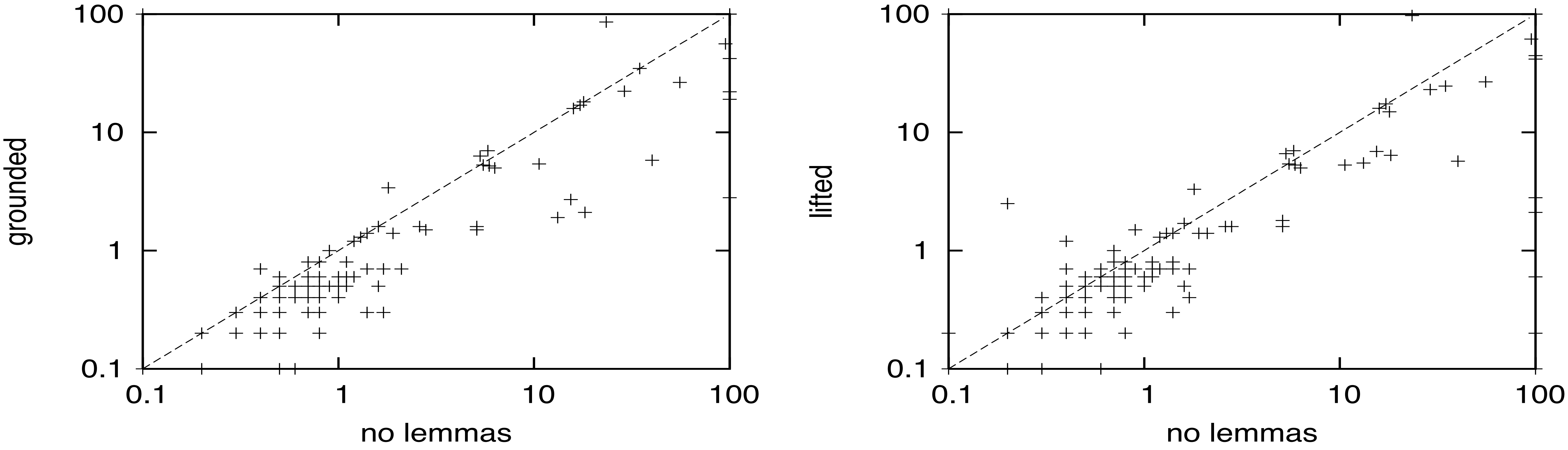
The more splits per problem, the more effective lemma learning is

Darwin - TPTP Problems (2)

Method	Solved Probls	Avg Time	Total Time	Speed up	Failure Steps	Propag. Steps	Split Steps	Splits per Problem
no lemmas	896	2.7	2397.0	1.00	24991	597286	45074	
grounded	895	2.4	2135.6	1.12	9476	391189	18935	≥ 0
lifted	898	2.4	2173.4	1.10	9796	399525	19367	
no lemmas	244	3.0	713.9	1.00	24481	480046	40766	
grounded	243	1.8	445.1	1.60	8966	273849	14627	≥ 3
lifted	246	2.0	493.7	1.45	9286	282600	15059	
no lemmas	108	5.2	555.7	1.00	23553	435219	38079	
grounded	108	2.2	228.5	2.43	8231	228437	12279	≥ 20
lifted	111	2.6	274.4	2.02	8535	238103	12688	
no lemmas	66	5.0	323.9	1.00	21555	371145	34288	
grounded	67	1.7	111.4	2.91	6973	183292	9879	≥ 100
lifted	70	2.3	151.4	2.14	7275	193097	10294	

The lifted method is more effective than the grounded method wrt. the number of solved problems, but worse wrt. the other measures

Darwin - Individual Runtimes



- Lemma learning is a win on most problems
- No surprises (loss of problems solved) with grounded method

Issues with the TPTP Problems

- TPTP includes both satisfiable and unsatisfiable one
  - Prover behaviour is better predictable for unsatisfiable ones
- Many problems are solvable with little splits
  - Lemma learning is not effective then
- Experiments with a second problem set
  - Basis: all **satisfiable** clausal TPTP problems
  - FM-Darwin: MACE-style model finder
  - Model search requires lots of splits, typically

FM-Darwin

- FM-Darwin: MACE-style model finder
- **Procedure**
  - Input: clause set S
  - Output: finite model of size n or non-termination
  - Transformation into function-free clause set FM(S,n)
    FM(S,n) is satisfiable ⇔ S has a finite model of size n
  - For n=1,2,...:
    - Call Darwin to decide satisfiability of FM(S,n)
    - Return model for S when FM(S,n) is satisfiable
- When model is at size n, this gives lots of backjumps:
  - For 1,2,...,n-1 all clause sets unsatisfiable
  - Axioms like x=1 ∨ ... ∨ x=n-1 introduce lot of branching

FM-Darwin, Satisfiable Problems

Method	Solved Probls	Average Time	Total Time	Speed up	Failure Steps	Propagate Steps	Split Steps	Splits per Problem
no lemmas	657	5.6	3601.3	1.00	404237	16122392	628731	≥ 0
grounded	669	3.3	2106.3	1.71	74559	4014058	99865	
lifted	657	4.7	3043.9	1.18	41579	1175468	68235	
no lemmas	162	17.8	2708.6	1.00	398865	15911006	614572	≥ 100
grounded	174	7.9	1203.1	2.25	70525	3833986	87834	
lifted	162	14.0	2126.2	1.27	38157	1023589	57070	
no lemmas	52	36.2	1702.9	1.00	357663	14580056	555015	≥ 1000
grounded	64	10.5	495.3	3.44	53486	3100339	64845	
lifted	57	11.5	538.7	3.16	26154	678319	39873	

- Considerable gain wrt. Propagate and Split steps
- Lifted method better wrt. reducing number of steps
- The grounded method is overall more effective

Conclusions

- Presented two methods of adding lemma learning to Model Evolution
  - Grounded Method
  - Lifted Method
- Both methods are "proper" learning
  - Unlike as in SAT, lemma can apply to infinitely many instances
  - The lifted method gives more general lemmas
- Grounded method seems to be best in average
  - Almost always a win
  - Could solve some problems previously unsolvable for Darwin
- Obtained speed-ups up to factor 3.44
  - But no "exponential" improvement
  - Similar observation made before in other EBL work