Logical Engineering with Instance Based Methods

Peter Baumgartner

Logic and Computation
NICTA

Computer Science Lab
Australian National University

NICTA

ANU
THE AUSTRALIAN NATIONAL UNIVERSITY

Collaborators: Alexander Fuchs, Christoph Sticksel, Cesare Tinelli

An early IM - The DPLL Procedure

	Given		Clause form
Preprocessing	$\forall x\, \exists y\, \mathsf{P}(y, x) \land \forall z\, \neg\mathsf{P}(z, \mathsf{a})$	$\longrightarrow$	$\mathsf{P}(\mathsf{f}(x), x)$ $\neg\mathsf{P}(z, \mathsf{a})$
Outer loop: grounding	$\mathsf{P}(\mathsf{f}(\mathsf{a}), \mathsf{a})$ $\neg\mathsf{P}(\mathsf{a}, \mathsf{a})$	....	$\mathsf{P}(\mathsf{f}(\mathsf{a}), \mathsf{a})$ $\neg\mathsf{P}(\mathsf{a}, \mathsf{a})$ $\neg\mathsf{P}(\mathsf{f}(\mathsf{a}), \mathsf{a})$
Inner loop: propositional DPLL	**Satisfiable**		**Unsatisfiable**

Obvious problem: how to control the grounding?
Modern IMs address this (and other weaknesses)

Why Instance Based Methods?

IMs are different to Resolution, Tableaux, Connection Methods ...

- Conceptually
- Search space
- Decidable classes

Part I

IMs capitalize on advances in SAT solving

- Some IMs include "the best" SAT solvers as subroutines
- Some IMs lift successful SAT techniques to the first-order level
- All IMs apply successful first-order theorem proving techniques

Part II

Logical Engineering

- Exploit strengths of IMs by suitable mapping of application problems
- In particular for SW verification

Why Instance Based Methods?

IMs are different to Resolution, Tableaux, Connection Methods ...

- Conceptually
- Search space
- Decidable classes

Two-level IMs
One-level IMs

IMs capitalize on advances in SAT solving

- Some IMs include "the best" SAT solvers as subroutines
- Some IMs lift successful SAT techniques to the first-order level
- All IMs apply successful first-order theorem proving techniques

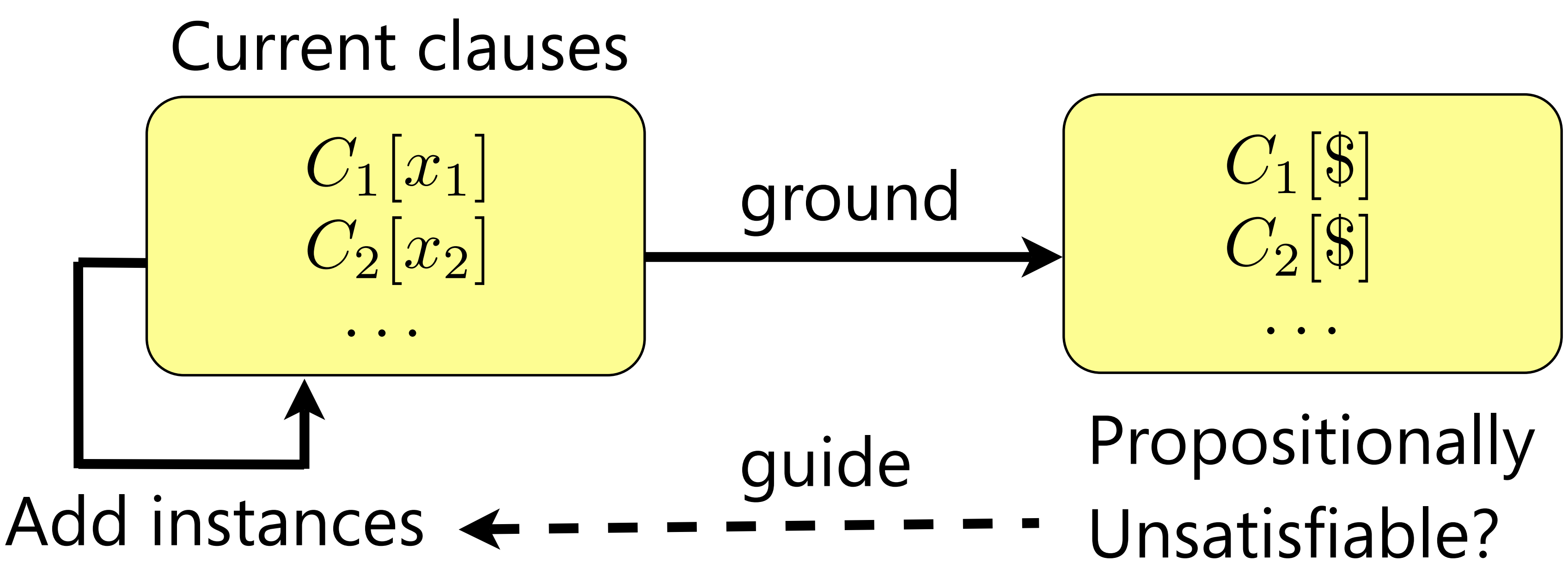
Logical Engineering

- Exploit strengths of IMs by suitable mapping of application problems
- In particular for SW verification

Two-Level vs One-Level IMs

Two-Level IMs

- Strict separation between instance generation and SAT solving phase
- Uses (arbitrary) propositional SAT solver as a subroutine
- DPLL, HL, SHL, OSHL [Plaisted et al], PPI [Hooker], InstGen[Ganzinger& Korovin], Equinox [Claessen] comparison paper [Jacobs&Waldmann]



InstGen: guide adding instances by model of \$-clause set and unification

Inst-Gen [Ganzinger&Korovin]

Current clauses

$$\mathsf{P}(\mathsf{f}(x), x) \lor \mathsf{Q}(x) \qquad \neg\mathsf{P}(z, \mathsf{a}) \lor \neg\mathsf{Q}(z)$$

$$\xrightarrow[x,z \,\to\, \$]{\text{ground}}$$

$$\underline{\mathsf{P}(\mathsf{f}(\$), \$)} \lor \mathsf{Q}(\$) \qquad \underline{\neg\mathsf{P}(\$, \mathsf{a})} \lor \neg\mathsf{Q}(\$)$$

Model: $\{\mathsf{P}(\mathsf{f}(\$), \$), \neg\mathsf{P}(\$, \mathsf{a})\}$

Model determines literals selection in current clauses for InstGen inference:

$$\text{InstGen}\ \frac{\underline{\mathsf{P}(\mathsf{f}(x), x)} \lor \mathsf{Q}(x) \qquad \underline{\neg\mathsf{P}(z, \mathsf{a})} \lor \neg\mathsf{Q}(z)}{\mathsf{P}(\mathsf{f}(\mathsf{a}), \mathsf{a}) \lor \mathsf{Q}(\mathsf{a}) \qquad \neg\mathsf{P}(\mathsf{f}(\mathsf{a}), \mathsf{a}) \lor \neg\mathsf{Q}(\mathsf{f}(\mathsf{a}))}$$

Conclusions are obtained by unifying selected literals

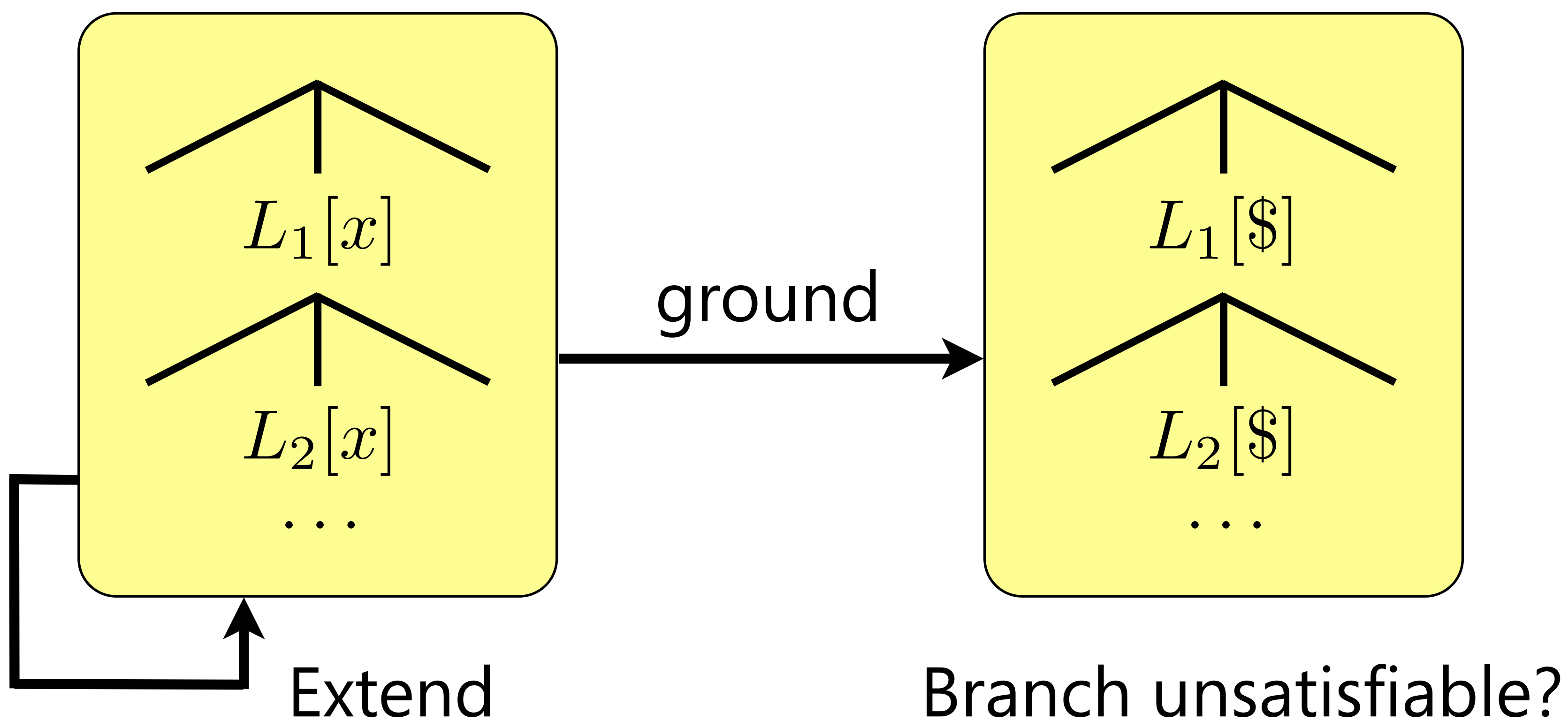
Add conclusions to "current clauses" and start over

This is just the very basic calculus

Two-Level vs One-Level IMs

One-Level IMs

- Monolithic: one single base calculus, two modes of operation
  - First-order mode: first-order calculus
  - Propositional mode: temporarily replace all variables by $
- HyperTableauxNG [B], DCTP[Letz&Stenz], OSHT [Plaisted&Yahya], FDPLL [B], ME [B&Tinelli]



Next: One-level IM FDPLL / Model Evolution

Model Evolution - Motivation

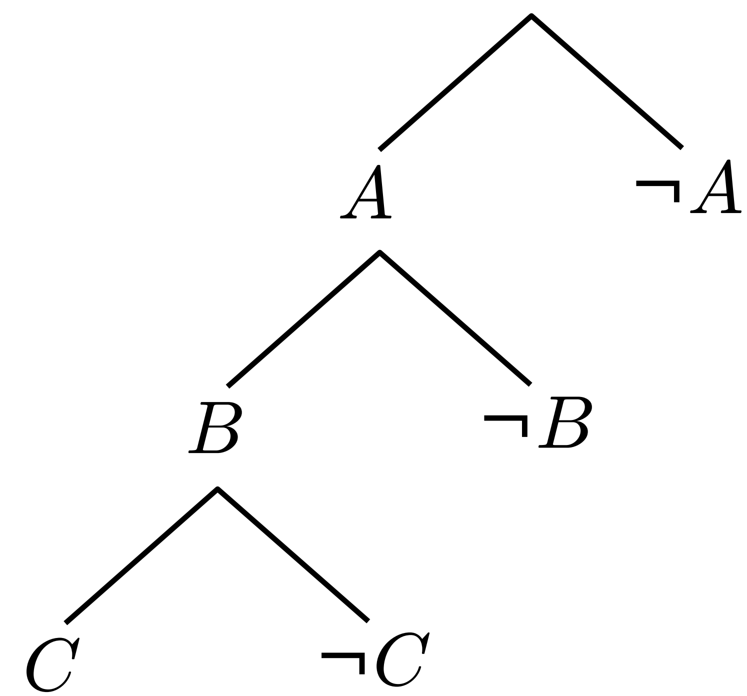
- The best modern SAT solvers (satz, MiniSat, zChaff) are based on the Davis-Putnam-Logemann-Loveland procedure [DPLL 1960-1963]
- **Can DPLL be lifted to the first-order level?**
  How to combine
  - DPLL techniques
    (unit propagation, backjumping, lemma learning,...)
  - first-order techniques?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

DPLL procedure

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

Algorithm components:

- Propositional semantic tree
  enumerates interpretations
- Propagation
- Split
- Backjumping



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

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

ME - lifting this idea to first-order level

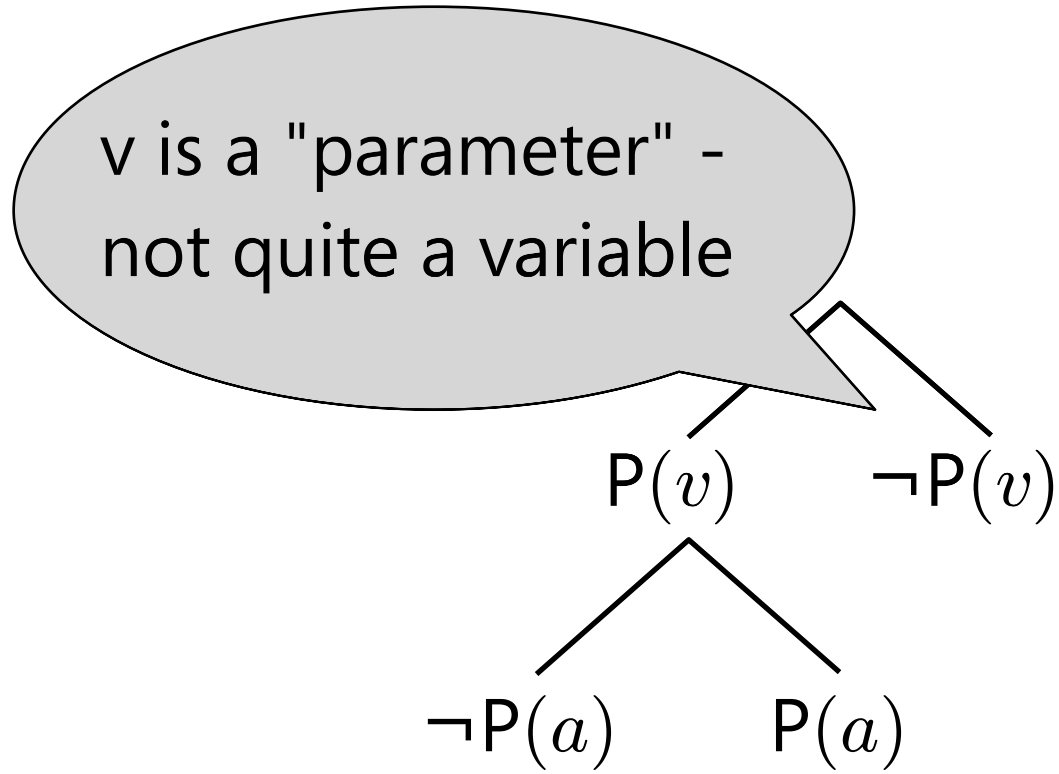
ME as First-Order DPLL

Input: First-order clause set

Output: Model or „unsatisfiable"
if termination

Algorithm components:

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



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

Interpretation induced by a branch?

Interpretation Induced by a Branch

Branch B

$\mathsf{P}(x, y)$

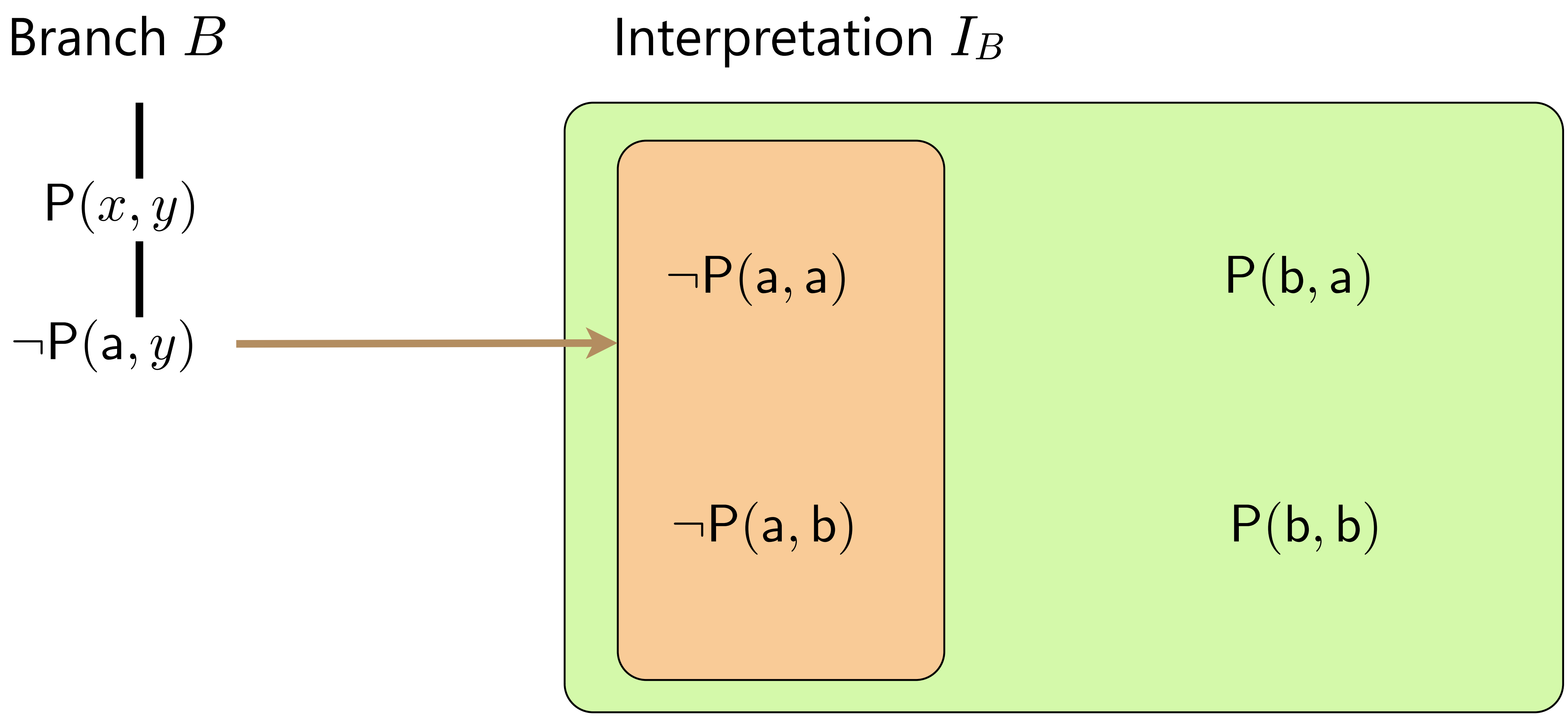
Interpretation I_B

$\mathsf{P}(\mathsf{a}, \mathsf{a})$ $\mathsf{P}(\mathsf{b}, \mathsf{a})$

$\mathsf{P}(\mathsf{a}, \mathsf{b})$ $\mathsf{P}(\mathsf{b}, \mathsf{b})$

- A branch literal specifies a truth value for all its ground instances, unless there is a more specific literal specifying the opposite truth value

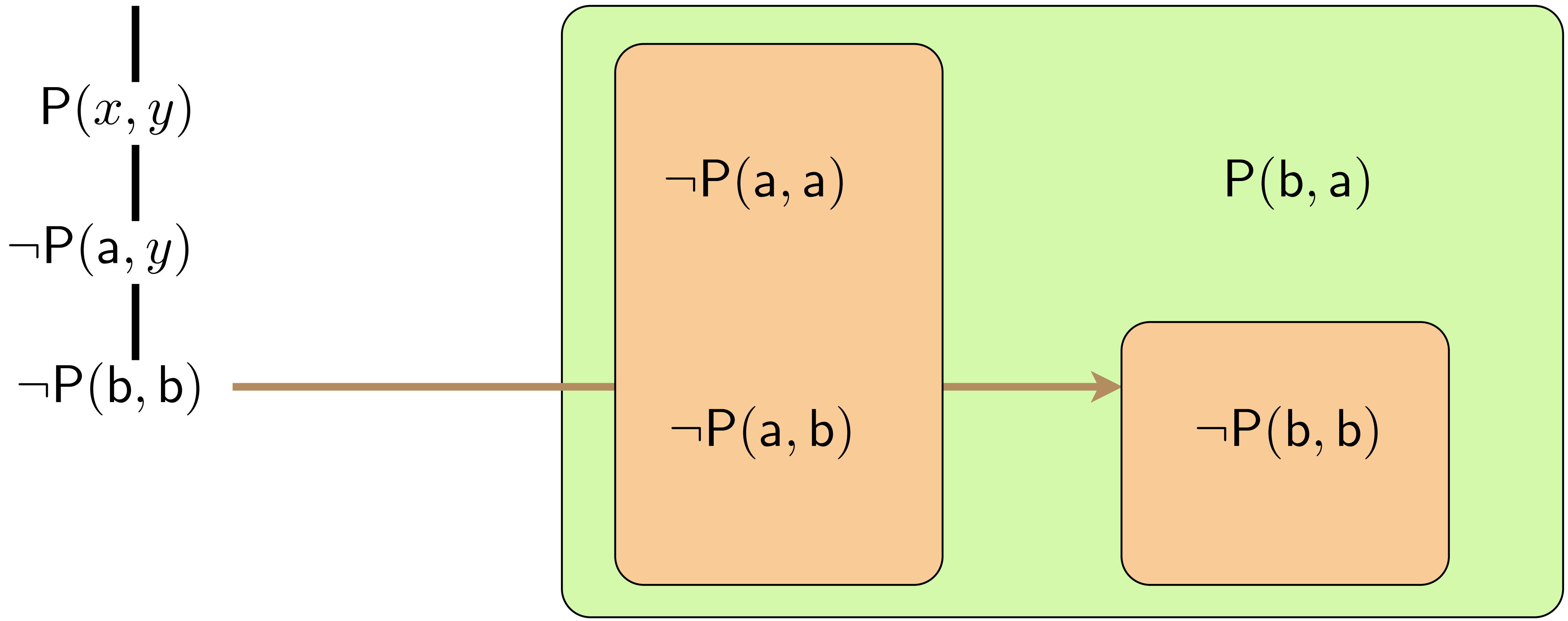
Interpretation Induced by a Branch



- A branch literal specifies a truth value for all its ground instances, unless there is a more specific literal specifying the opposite truth value

Interpretation Induced by a Branch

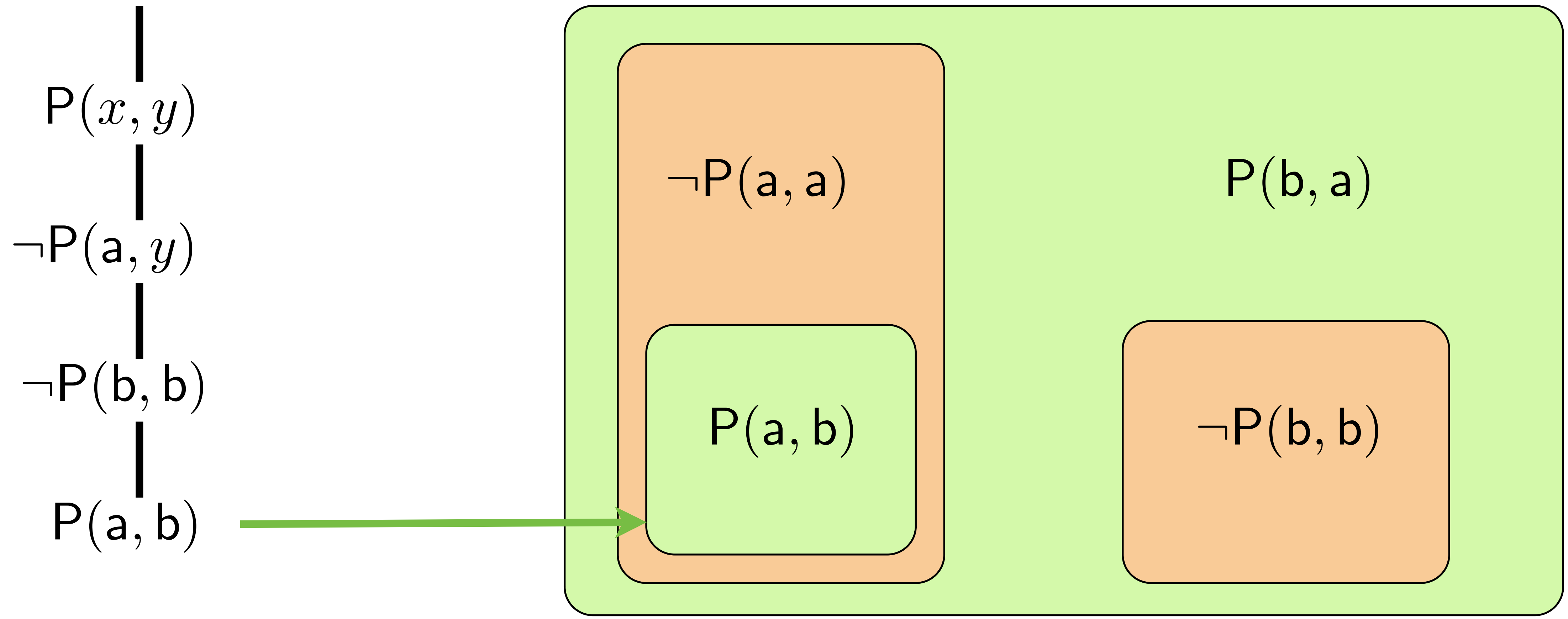
Branch B — Interpretation I_B



- A branch literal specifies a truth value for all its ground instances, unless there is a more specific literal specifying the opposite truth value

Interpretation Induced by a Branch

Branch B Interpretation I_B



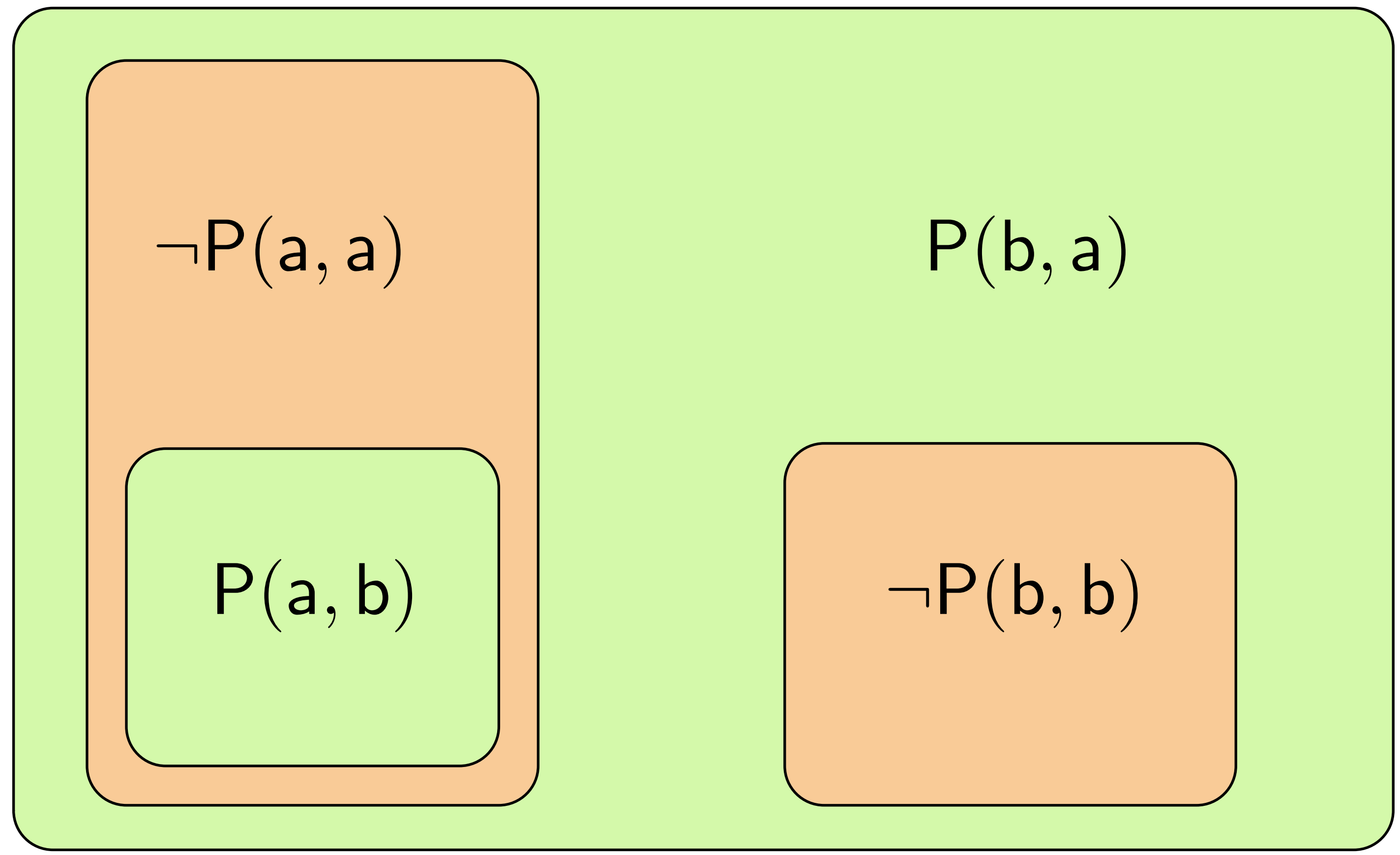
- A branch literal specifies a truth value for all its ground instances, unless there is a more specific literal specifying the opposite truth value

Interpretation Induced by a Branch

Branch B

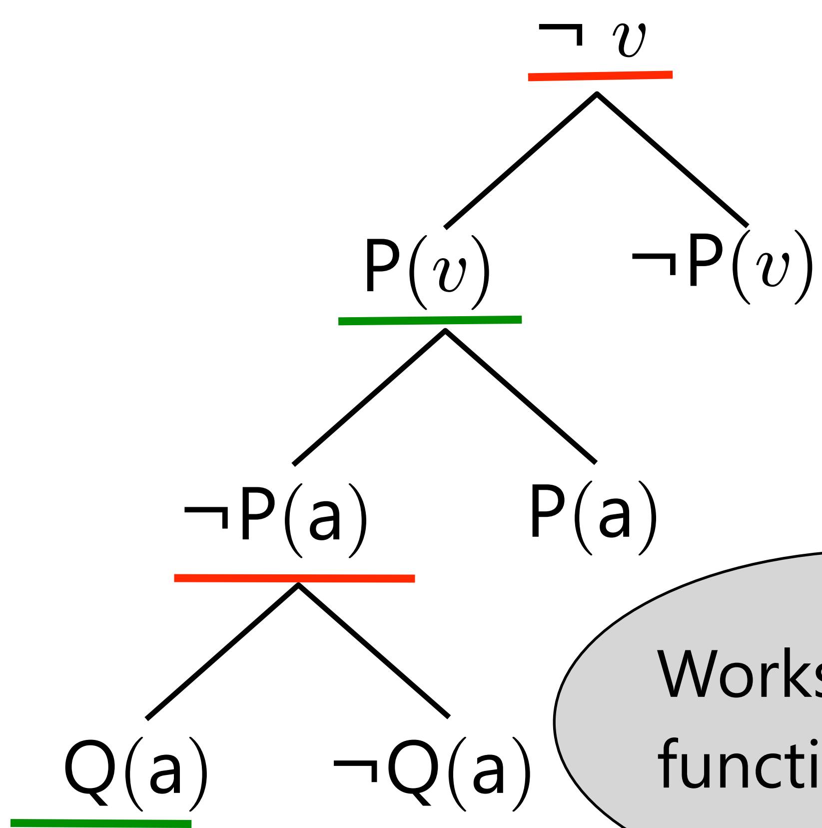
$$\{\ \mathsf{P}(x, y)\ ,\quad \neg\mathsf{P}(\mathsf{a}, y)\ ,\quad \neg\mathsf{P}(\mathsf{b}, \mathsf{b})\ ,\quad \mathsf{P}(\mathsf{a}, \mathsf{b})\ \}$$

Interpretation I_B



- A branch literal specifies a truth value for all its ground instances, unless there is a more specific literal specifying the opposite truth value
- **The order of the literals on the branch is irrelevant**

Inference Rule: Split



Works also with function symbols

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

True: $\mathsf{P}(\mathsf{b})$

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

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

True: $\mathsf{P}(\mathsf{b})$, $\mathsf{Q}(\mathsf{a})$

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

$$\{\neg v, \mathsf{P}(v), \neg\mathsf{P}(\mathsf{a})\} \overset{?}{\models} \mathsf{P}(x) \vee \mathsf{Q}(x) \quad ✗ \quad \xrightarrow{\text{Context Unifier}} \quad \frac{\mathsf{P}(\mathsf{a}) \vee \mathsf{Q}(\mathsf{a})}{\text{Split}}$$

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

Split - detect falsified instances and repair interpretation
Additional rules: Close, Assert, Compact, Resolve, Subsume

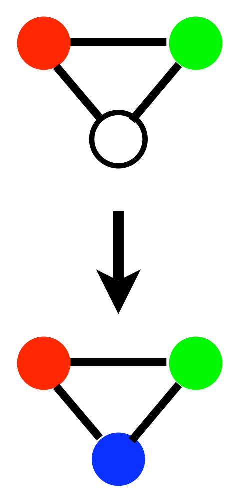
Example - Detecting Functional Dependencies

Graph 3-colorability

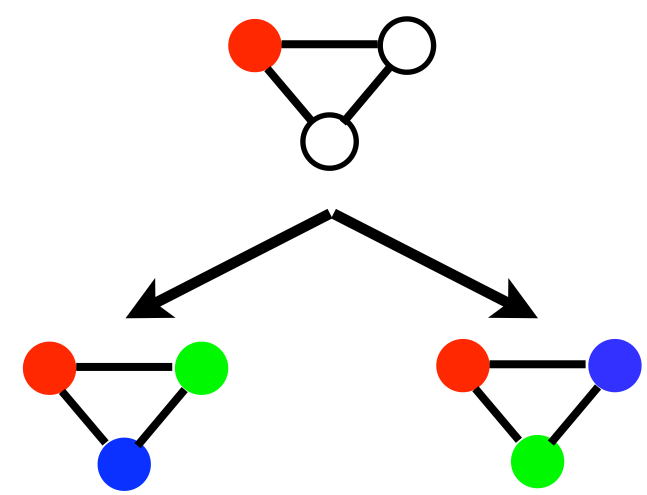
$$\forall n\ \mathsf{R}(n) \lor \mathsf{G}(n) \lor \mathsf{B}(n)$$

$$\forall n\ (\mathsf{R}(n) \rightarrow \neg\mathsf{G}(n)) \land (\mathsf{R}(n) \rightarrow \neg\mathsf{B}(n)) \land (\mathsf{B}(n) \rightarrow \neg\mathsf{G}(n))$$

$$\begin{aligned}\forall m,n\ &(\mathsf{R}(m) \land \mathsf{R}(n) \rightarrow \neg\mathsf{edge}(m,n)) \land \\ &(\mathsf{G}(m) \land \mathsf{G}(n) \rightarrow \neg\mathsf{edge}(m,n)) \land (\mathsf{B}(m) \land \mathsf{B}(n) \rightarrow \neg\mathsf{edge}(m,n))\end{aligned}$$



B depends on R **and** G



B does not depend on R

Application in NICTA's G12 platform

(Dis-)prove functional (non-)dependance

Demo: Darwin theorem prover

ME - Achievements so far

- **FDPLL** [CADE-17]
  - Basic ideas, predecessor of ME
- **ME Calculus** [CADE-19, AI Journal]
  - Proper treatment of universal variables and unit propagation
  - Semantically justified redundancy criteria
- **ME+Equality** [CADE-20]
  - Superposition inference rules, currently being implemented
- **ME+Lemmas** [LPAR 2006]
- **Darwin prover** [JAIT 2006]
  `http://combination.cs.uiowa.edu/Darwin/`
  - Won CASC-J3 and CASC-21 EPR division
- **FM-Darwin**: finite model computation [JAL 2007]

Resolution vs IMs

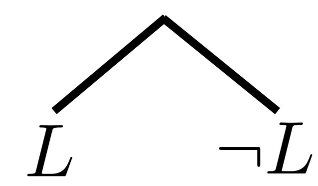
Resolution

$$\text{Res}\ \frac{C \vee L \qquad \overline{L'} \vee D}{(C \vee D)\sigma}$$

- Inefficient in propositional case
- Clauses can grow in length
- Recombination of clauses
- Subsumption deletion
- Selection by A-ordering
- Difficult to extract model
- Decides many classes
- Wins CASC FOF

Instance Based Methods

$$\text{InstGen}\ \frac{C \vee L \qquad \overline{L'} \vee D}{(C \vee L)\sigma \quad (\overline{L'} \vee D)\sigma}$$



- Efficient in propositional case
- Clauses do not grow in length
- No recombination of clauses
- Limited subsumption deletion
- Selection by interpretation
- Easy to extract model
- **Decides Bernays-Schönfinkel Class**
- Does not win CASC FOF

Complementary methods

Why Instance Based Methods?

IMs are different to Resolution, Tableaux, Connection Methods ...

- Conceptually
- Search space
- Decidable classes

IMs capitalize on advances in SAT solving

- Some IMs include "the best" SAT solvers as subroutines
- Some IMs lift successful SAT techniques to the first-order level
- All IMs apply successful first-order theorem proving techniques

Briefly

Ideas

Logical Engineering

- Exploit strengths of IMs by suitable mapping of application problems
- In particular for SW verification

Exploiting Strengths of IMs

... in particular as decision procedures for the Bernays-Schönfinkel class:

- CASC-competition: EPR category
- Optimized functional translation of modal logics [Ohlbach&Schmidt]
- DQBF satisfiability $\forall P_1 \;\; \exists Q_1(P_1) \;\; \forall P_2 \;\; \exists Q_2(P_2) \;\; \cdots$
- LTL model checking [Navarro-Pérez&Voronkov CADE-21]
- Planning [Voronkov et al CP 2007]
- CEGAR [Klaessen]
- Back-end for DL reasoning (SHOIQ), cf [Motik et al])
- Strong equivalence (under answer sets semantics) of logic programs
- Finite model computation (FM-Darwin)
- Within constraint modelling
  - Analysis of constraint models (functional dependencies ...)
  - Model expansion [Ternovska&Mitchell]

Application for SW Verification

Applications of formal methods often rely on proving or disproving first-order logic formulas over a fixed (background) theory $\mathcal{T}$

- E.g. proving properties of programs involving arrays and integers

Core Problem: SMT - Satisfiability Modulo Theories

- Is a given formula satisfiable modulo a given theory $\mathcal{T}$?

One Main Approach: DPLL($\mathcal{T}$)

- Prop. DPLL + solver for conjunctions of ground $\mathcal{T}$-literals ($\mathcal{T}$-solver)
- Issue: works inherently with propositional abstractions
  - DPLL cannot analyze term structure
  - Non-ground formulas grounded by "external" heuristic
    - Still a hot topic (cf. SMT session, R. Leino talk @ CADE-21)
    - Here: contribution from the viewpoint of First-Order ATP

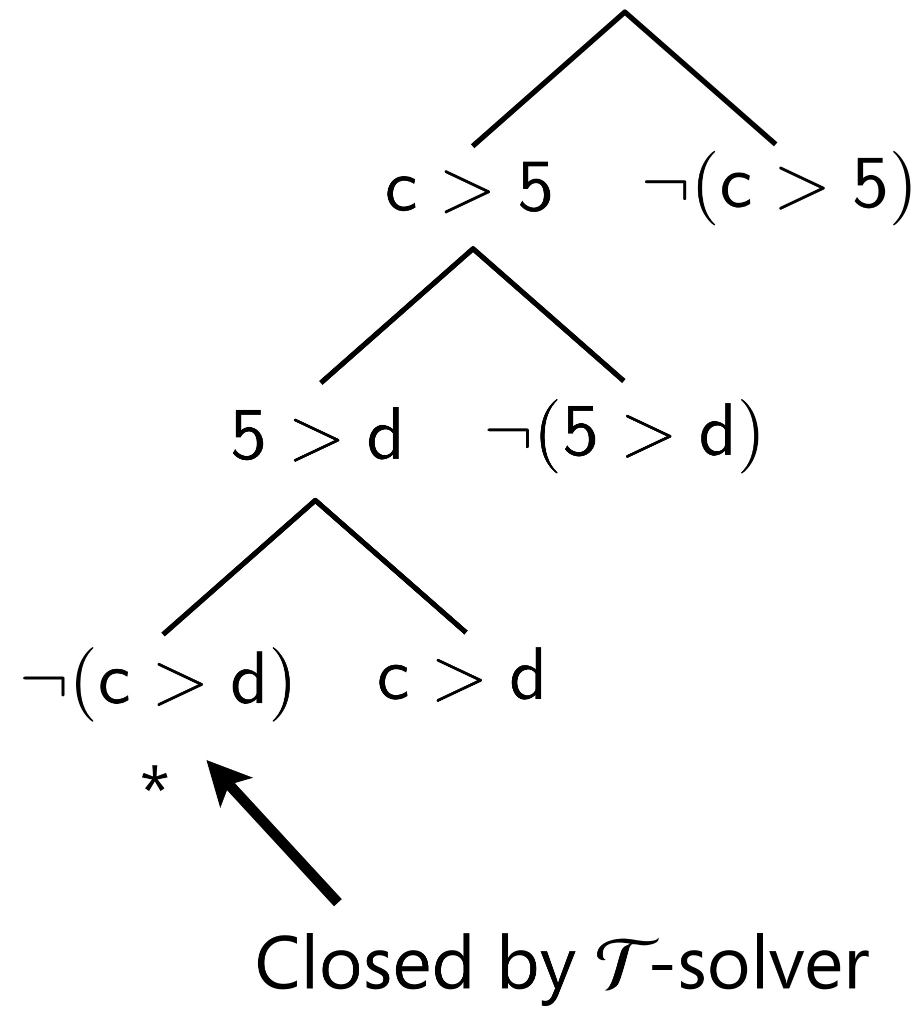
Plan: address issues by using "ME($\mathcal{T}$)" instead of DPLL($\mathcal{T}$)

DPLL($\mathcal{T}$) Approach to SMT

- DPLL computes candidate model of propositional abstraction
- Check candidate model with $\mathcal{T}$-solver

Treated as propositional variables

$$\begin{array}{lcll} & \cdots & & \\ c > 5 & \vee & \cdots & (1) \\ 5 > d & \vee & \cdots & (2) \\ \neg(c > d) & \vee & P(c) & (3) \end{array}$$



Refinements

- Incremental $\mathcal{T}$-solver
- $\mathcal{T}$-solver reports relevant literals
- Theory propagation ($\mathcal{T}$-solver computes unit consequences)

Lifting DPLL($\mathcal{T}$) to ME($\mathcal{T}$) ?

ME($\mathcal{T}$) - Basic Approach

- Replace DPLL by ME
- Rename all theory literals as positive literals
  - $\neg(5 > 3)$ becomes $5 \not> 3$
- Turn args of negative non-theory literals into vars
  - $\neg P(5)$ becomes $x \neq 5 \vee \neg P(x)$

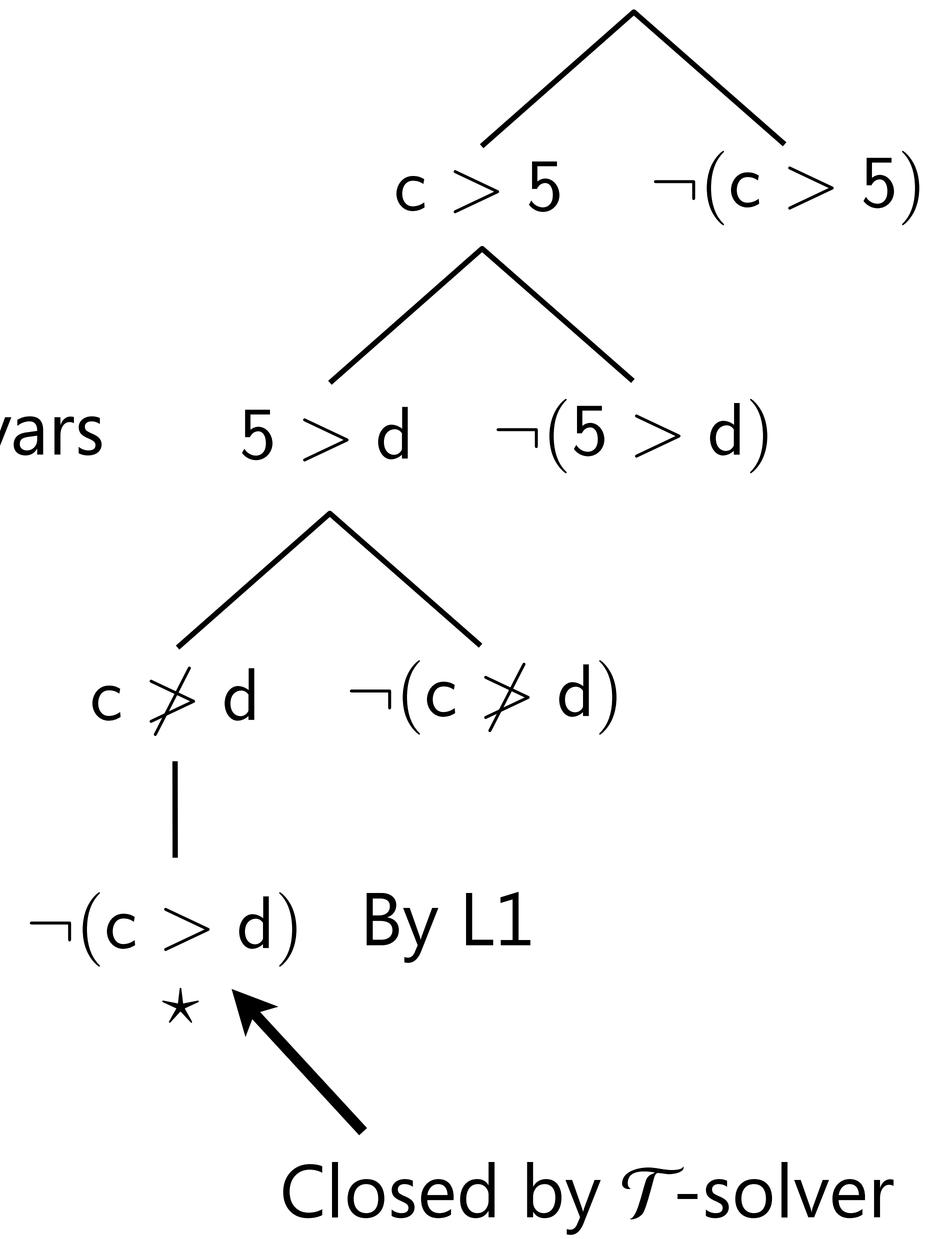
Ground FO-literals

$$\begin{array}{llll} & \cdots & & \\ c > 5 & \vee & \cdots & (1) \\ 5 > d & \vee & \cdots & (2) \\ c \not> d & \vee & P(c) & (3) \end{array}$$

x, y FO variables

"Theory lemma"

$$\neg(x > y) \vee \neg(x \not> y) \qquad \text{(L1)}$$



ME($\mathcal{T}$) proper generalization of DPLL($\mathcal{T}$)

Theory Lemmas Application I: Theory Propagation

- **Theory propagation** - important efficiency improvement for DPLL($\mathcal{T}$)
  - $\mathcal{T}$-solver computes $\mathcal{T}$-implied literals which avoids branching
- Approximated in ME($\mathcal{T}$) by theory lemmas
  - Doesn't rely on $\mathcal{T}$-solver in any way

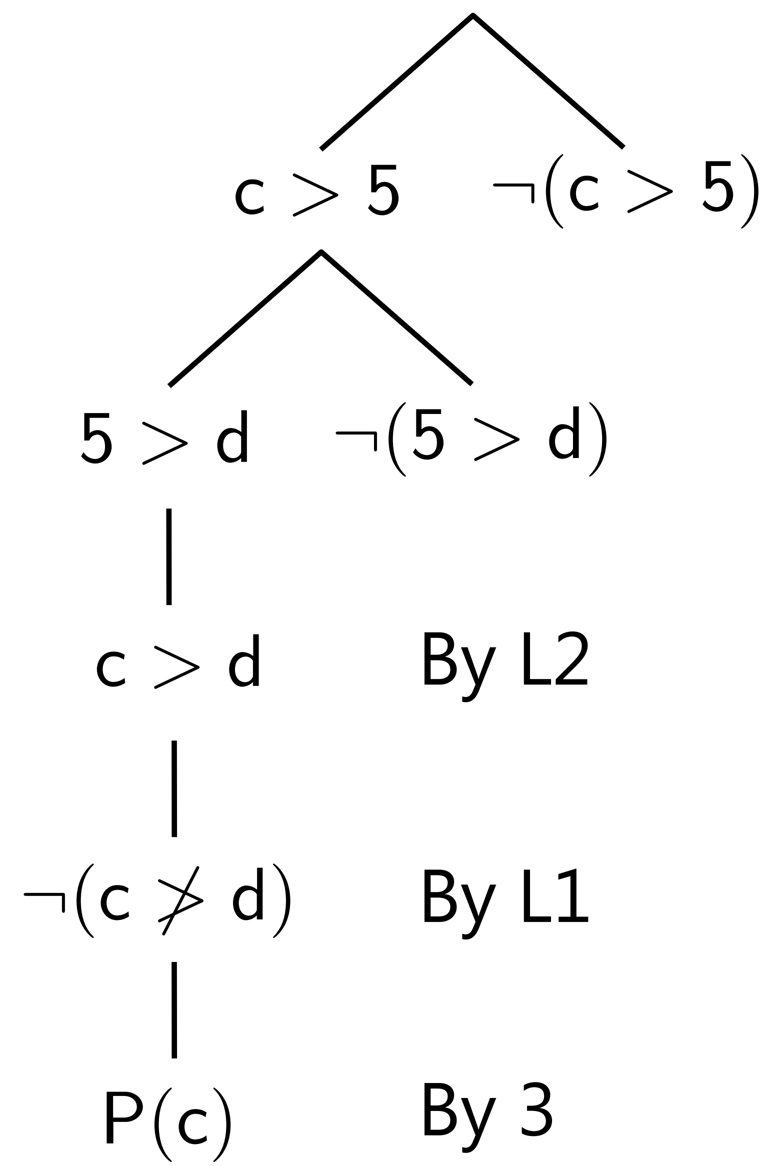
Input clause set

$$\begin{array}{llll} & \cdots & & \\ \mathsf{c} > 5 & \vee & \cdots & (1) \\ 5 > \mathsf{d} & \vee & \cdots & (2) \\ \mathsf{c} \not> \mathsf{d} & \vee & \mathsf{P}(\mathsf{c}) & (3) \end{array}$$

Theory lemmas

$$\begin{array}{ll} \neg(x > y) \vee \neg(x \not> y) & \text{(L1)} \\ \neg(x > y) \vee \neg(y > z) \vee x > z & \text{(L2)} \end{array}$$

Cheap implementation of e.g. "ME(DL)"
Also: avoids learning of subsumed clauses



Theory Lemmas Application II: Problem Reduction

To prove: $(x+y)^2 = x^2 + 2xy + y^2$ (Binom)

Sufficient set of axioms:

$$xy = yx \qquad x + y = y + x \qquad \text{(Comm)}$$
$$x(yz) = (xy)z \qquad x + (y + z) = (x + y) + z \qquad \text{(Assoc)}$$
$$1x = x \qquad 0 + x = x \qquad \text{(Neutral)}$$
$$x(y + z) = xy + xz \qquad 2x = x + x \qquad \text{(Distrib,2)}$$

FO theorem proving, axioms above: very easy e.g. for SPASS, KeY

DPLL(T), T=UFLIA, left column axioms+(2): CVC3 fails

ME(T), T=UFLIA, left column axioms+(2) as theory lemmas:

reduce (Binom) to $(xx + xy) + (xy + yy) = xx + ((xy + xy) + yy)$,
then complete proof with call to UFLIA-solver

Can (E.g.) KeY taclets modeled as clauses, for contextual rewriting?
Related to [Bonacina&Echenim] this CADE

Theory Lemmas Application III: Non-ground Input

Typical scenario

- $\mathcal{T}$ = Linear arithmetic + Arrays + ...
- Uninterpreted function and/or predicate symbols

The theory of arrays

$$\mathsf{select}(\mathsf{store}(a, i, j, e), i, j) = e \qquad \text{(A1)}$$

$$\mathsf{select}(\mathsf{store}(a, i, j, e), i', j') = \mathsf{select}(a, i', j') \leftarrow \neg(i = i') \qquad \text{(A2)}$$

$$\mathsf{select}(\mathsf{store}(a, i, j, e), i', j') = \mathsf{select}(a, i', j') \leftarrow \neg(j = j') \qquad \text{(A3)}$$

Challenging example problem [Ranise]

Define

$$\forall a, n\ \mathsf{symmetric}(a, n) \leftrightarrow (\forall i, j\ 1 \le i, j \le n \rightarrow \mathsf{select}(a, i, j) = \mathsf{select}(a, j, i))$$

Prove $\{\mathsf{symmetric}(\mathsf{a}, \mathsf{n})\}\quad \mathsf{a}[0, 0] := \mathsf{e}_0\,; \ldots ; \mathsf{a}[\mathrm{k}, \mathrm{k}] := \mathsf{e}_\mathrm{k}\quad \{\mathsf{symmetric}(\mathsf{a}, \mathsf{n})\}$

Results in non-ground clause set
Required instances are not obvious

Theory Lemmas = Array Axioms Relational Translation

Array axioms (1-dimensional, for simplicity)

$$\mathsf{select}(\mathsf{store}(a,i,e),i) = e \qquad \text{(A1)}$$

$$\mathsf{select}(\mathsf{store}(a,i,e),j) = \mathsf{select}(a,j) \leftarrow \neg(i=j) \qquad \text{(A2)}$$

Relational translation

$$\mathsf{select}(h,i) = e \leftarrow \mathsf{store}(a,i,e) = h$$

$$\mathsf{select}(h,i,e) \leftarrow \mathsf{store}(a,i,e,h) \qquad \text{(A1)}$$

$$\mathsf{select}(h,j,r) \leftarrow \mathsf{store}(a,i,e,h) \land \mathsf{select}(a,j,r) \land \neg(i=j) \qquad \text{(A2)}$$

$$r1 = r2 \leftarrow \mathsf{select}(a,i,r1) \land \mathsf{select}(a,i,r2) \qquad \text{(Func-1)}$$

$$r1 = r2 \leftarrow \mathsf{store}(a,i,e,r1) \land \mathsf{store}(a,i,e,r2) \qquad \text{(Func-2)}$$

$$\mathsf{select}(a,i,\mathsf{skf}(a,i)) \leftarrow \qquad \text{(Totality)}$$

(Totality) is problematic

- Generates a huge search space
  - Without it all function symbols have gone (good for ME)
- Approximate (Totality) by

$$\mathsf{select}(a,i,\mathsf{skf}(a,i)) \leftarrow \mathsf{index}(i) \qquad \text{(Definedness)}$$

index ?

Controlling the Search Space with the index Predicate

Relational translation of array axioms

$$\text{select}(h,i,e) \leftarrow \text{store}(a,i,e,h) \qquad \text{(A1)}$$

$$\text{select}(h,j,r) \leftarrow \text{store}(a,i,e,h) \wedge \text{select}(a,j,r) \wedge \neg(i=j) \qquad \text{(A2)}$$

$$r1 = r2 \leftarrow \text{select}(a,i,r1) \wedge \text{select}(a,i,r2) \qquad \text{(Func-1)}$$

$$r1 = r2 \leftarrow \text{store}(a,i,e,r1) \wedge \text{store}(a,i,e,r2) \qquad \text{(Func-2)}$$

$$\text{select}(a,i,\text{skf}(a,i)) \leftarrow \text{index}(i) \qquad \text{(Definedness)}$$

Options for defining the index predicate

(1) add a clause "index(i)" - select is total

(2) add a clause "¬index(i)" - select is partial

(3) add clauses "index(t)" for all input ground terms t

(4) add clauses "index(i) ← P(...,i,...)" for all/some predicate symbols P

Options (2) - (4) are incomplete

But target logic LIA + free predicate symbols is incomplete anyways

Experiments with Symmetric Array Problem

Definition of "symmetric array":

$$\forall a, n \ \mathsf{symmetric}(a, n) \leftrightarrow (\forall i, j \ 1 \le i, j \le n \to \mathsf{select}(a, i, j) = \mathsf{select}(a, j, i))$$

Prove $\{\mathsf{symmetric}(\mathsf{a}, \mathsf{n})\} \ \ \mathsf{a}[0,0] := \mathsf{e}_0 ; \ldots ; \mathsf{a}[\mathrm{k},\mathrm{k}] := \mathsf{e}_\mathrm{k} \ \ \{\mathsf{symmetric}(\mathsf{a}, \mathsf{n})\}$

Systems tried

CVC3: DPLL($\mathcal{T}$) prover (with instantiation heuristics) - cannot solve

KeY: Interactive verification system, "taclets" - cannot solve

SPASS: Hyper-resolution setting, equality array axioms (performed best)

Darwin: Relational array axioms, heuristics (4)

k	SPASS	Darwin
2	< 1	< 1
3	142	3
4	$>$ 5h	7
5	$>$ 5h	20
6	$>$ 5h	63

To be fair:
no arithmetic in this example:
SPASS is a complete prover, whereas
Darwin setup is incomplete
but allows good control of search space

ME($\mathcal{T}$)- Conclusion (1)

- **View from DPLL($\mathcal{T}$)**
  - Proper extension of DPLL($\mathcal{T}$) by integrating FO reasoning
    - Advantages derive from being able to analyze term structure
  - New way to handle non-ground formulas
    - Implemented by theory lemmas instead of meta-logical: "Points of definedness" (cf. "select" above) computed by calculus itself, by first-order reasoning, in a by need fashion
- **View from First-Order Theorem Proving**
  - This is "total theory reasoning" + "partial theory reasoning" ($\mathcal{T}$-propagation by theory lemmas)
  - Goal: better functionality of ATP systems
    - Useful explanation for failure, e.g. a model
    - Reasoning with integers

Message of the day

Conclusion (2)

- **Related Work**
  - Big engines approach [Armando&Bonacina&Ranise&Schulz]:
    E.g. DPLL($\mathcal{T}$) where $\mathcal{T}$ is implemented by a first-order theorem prover
  - SPASS+ $\mathcal{T}$ [Prevosto&Waldmann]:
    two-level architecture with SMT-solver as black box
- **Future**
  - Implement the coupling ME + CVC3
  - Experiments
    - In particular proof obligations from KeY
  - $\mathrm{ME}_{\mathcal{T}}$ - non-ground $\mathcal{T}$-interpretations

$$P(v) \mid v < 5 \quad — \quad \neg P(v) \mid v < 5$$