Instance Based Methods

Peter Baumgartner

NICTA, Logic and Computation Program, Canberra
Peter.Baumgartner@nicta.com.au

Theorem proving is about . . .

Logics: Propositional, First-Order, Higher-Order, Modal, Description, . . .

Calculi and proof procedures: Resolution, DPLL, Tableaux, . . .

Systems: Interactive, Automated

Applications: Knowledge Representation, Verification, . . .

Milestones

60s: Calculi: DPLL, Resolution, Model Elimination

70s: Logic Programming

80s: Logic Based Knowledge Representation

90s: Modern Theory and Implementations, "A Basis for Applications"

2000s: Ontological Engineering, Verification

IMs: first-order automated calculi and proof procedures

Two Separated Worlds

	First-Order Reasoning	Propositional Reasoning
Techniques	Resolution	DPLL
	Model Elimination	OBDD
	Hyper Linking	Stalmarck's Method
		Tableaux
		Stochastic (GSAT)
Systems	E, Otter, Setheo, SNARK, Spass, Vampire, FACT	Chaff, SMV, Heerhugo, Walk-Sat, Minisat
Applications	SW-Verification	Symbolic Model Checking
	Mathematics	Mathematics
	Description Logics	Planning
	TPTP	Nonmonotonic Reasoning

IM's combine techniques from propositional and first-order reasoning

Talk Overview

Talk provides overview about the following

- Common principles behind IMs, Some calculi
- Comparison among IMs, difference to tableaux and resolution
- Ranges of applicability/non-applicability

Topics left out

- Improvements: universal variables, lemma learning
- Extensions: equality, finite domain reasoning
- Implementations and implementation techniques

Setting the Stage

Skolem-Herbrand-Löwenheim Theorem

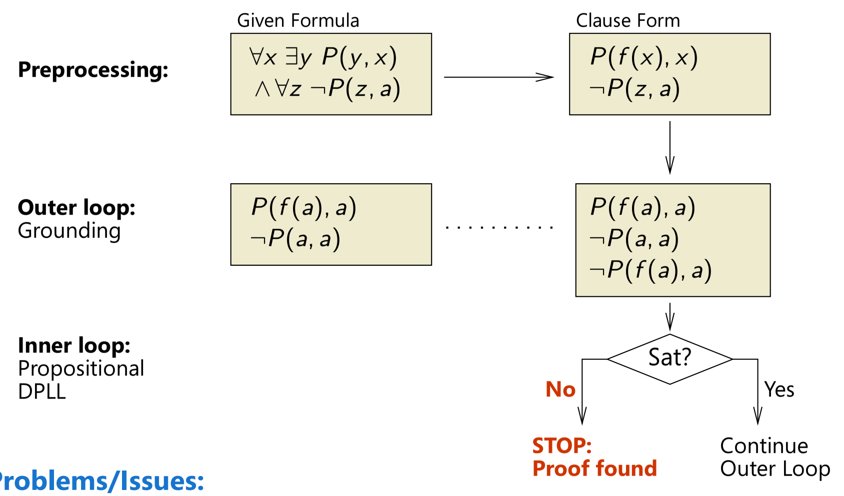
$\forall\phi$ is unsatisfiable iff some finite set of ground instances $\{\phi\gamma_1, \ldots, \phi\gamma_n\}$ is unsatisfiable

For refutational theorem proving (i.e. start with negated conjecture) it thus suffices to

- enumerate growing finite sets of such ground instances, and
- test each for propositional unsatisfiability. Stop with "unsatisfiable" when the first propositionally unsatisfiability set arrives

This has been known for a long time: Gilmore's algorithm, DPLL
It is also a common principle behind IMs
So what's special about IMs? Do this in a clever way!

An early IM: the DPLL Procedure



Problems/Issues:

- Controlling the grounding process in **outer loop** (irrelevant instances)
- Repeat work **across** inner loops
- Weak redundancy criterion **within** inner loop

Development of IMs (I)

Purpose of this slide

- List existing methods (apologies for "forgotten" ones ...)
- Define abbreviations used later on
- Provide pointer to literature
- Itemize structure indicates reference relation (when obvious)
- **Not:** table of contents of what follows (presentation is systematic instead of historical)

DPLL – Davis-Putnam-Logemann-Loveland procedure [DP60], [DLL62b], [DLL62a], [Dav63], [CDHM64]

- FDPLL – First-Order DPLL [Bau00]
  - ME – Model Evolution Calculus [BT03]
  - ME with Equality [BT05]

Development of IMs (II)

HL – Hyperlinking [LP92]

- SHL – Semantic Hyper Linking [CP94]
  - OSHL – Ordered Semantic Hyper Linking [PZ97]

PPI – Primal Partial Instantiation (1994) [HRCS02]

- "Inst-Gen" [GK03]

MACE-Style Finite Model Buiding [McC94],..., [CS03]

DC – Disconnection Method [Bil96]

- HTNG - Hyper Tableaux Next Generation [Bau98]
- DCTP – Disconnection Tableaux [LS01]

Ginsberg & Parkes method [GP00]

OSHT – Ordered Semantic Hyper Tableaux [YP02]

Classification of Instance Based Methods

Two Families of IMs

- Two-Level Calculi
- One-Level Calculi

Next

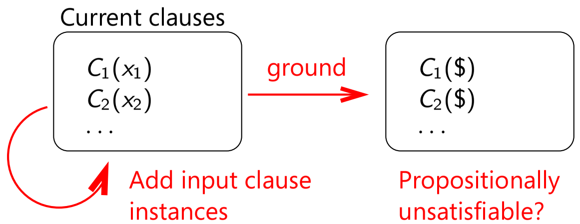
(1) Describe some members of each family, and

(2) do some comparison

Two-Level vs. One-Level Calculi

Two-Level Calculi

- Separation between instance generation and SAT solving phase
- Uses (arbitrary) propositional SAT solver as a subroutine
- DPLL, HL, SHL, OSHL, PPI, Inst-Gen
- **Problem:** how to tell SAT solver to take advantage of an input clause like $\forall x P(x)$?

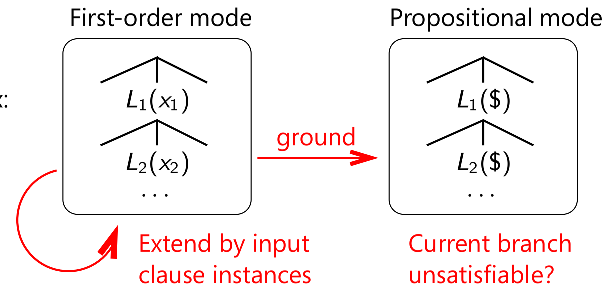


Two-Level vs. One-Level Calculi

One-Level Calculi

- Monolithic: one single base calculus, two modes of operation
- First-order mode: builds base calculus data structure from input clause instances
- Propositional mode: \$-instance of data structures drives first-order mode
- HyperTableaux NG, DCTP, OSHT, FDPLL, ME

E.g. Tableaux:



Next: two-level calculus "Inst-Gen" [GK03]

Inst-Gen - Idea (I)

Important notation: $\perp$ denotes both a unique constant and a substitution that maps every variable to $\perp$.

Example, S is "current clause set" :

$$S: \quad P(x,y) \lor P(y,x) \qquad\qquad S\perp: \quad P(\perp,\perp) \lor P(\perp,\perp)$$
$$\neg P(x,x) \qquad\qquad\qquad \neg P(\perp,\perp)$$

Analyze $S\perp$:

- In this case: SAT detects **unsatisfiability** of $S\perp$
- Stop and return "S is unsatisfiable"

But what if $S\perp$ is satisfied by some model, denoted by $I_\perp$?

Inst-Gen - Idea (II)

Important notation: $\perp$ denotes both a unique constant and a substitution that maps every variable to $\perp$.

Example, S is "current clause set" :

$$S: \quad \underline{P(x)} \vee Q(x) \qquad\qquad S\perp: \quad \underline{P(\perp)} \vee Q(\perp)$$
$$\underline{\neg P(a)} \qquad\qquad\qquad\qquad \underline{\neg P(a)}$$

Analyze $S\perp$:

- In this case: SAT detects satisfiability of $S\perp$
- Selected literals $\neg P(a)$ and $P(\perp)$ of $S\perp$ specify a model of $S\perp$
- But selected literals $\neg P(a)$ and $P(x)$ of S don't specify a model of S
- Use selected literals $\neg P(a)$ and $P(x)$ to derive new clause instance $P(a) \vee Q(a)$

Inst-Gen - Idea (III)

Add the new instance $P(a) \vee Q(a)$ to S gives:

$$S: \quad \underline{P(x)} \vee Q(x) \qquad\qquad S\perp: \quad \underline{P(\perp)} \vee Q(\perp)$$
$$P(a) \vee \underline{Q(a)} \qquad\qquad\qquad P(a) \vee \underline{Q(a)}$$
$$\underline{\neg P(a)} \qquad\qquad\qquad\qquad \underline{\neg P(a)}$$

Analyze $S\perp$: ...

Summary

- Propositional model (of $S\perp$, via selection) conceived as approximation of first-order model (of S)
- **Calculus goal:** refine approximation until model of S is found or refining the model is impossible
- Refinement is done by adding instances of clauses of S
- The **Inst-Gen inference rule** does this

Inst-Gen Inference Rule

$$\text{Inst-Gen} \quad \frac{C \vee L \qquad \overline{L'} \vee D}{(C \vee L)\theta \qquad (\overline{L'} \vee D)\theta} \quad \text{where}$$

(i) $\theta = \mathrm{mgu}(L, L')$, and

(ii) θ is a **proper instantiator:** maps some variables to nonvariable terms

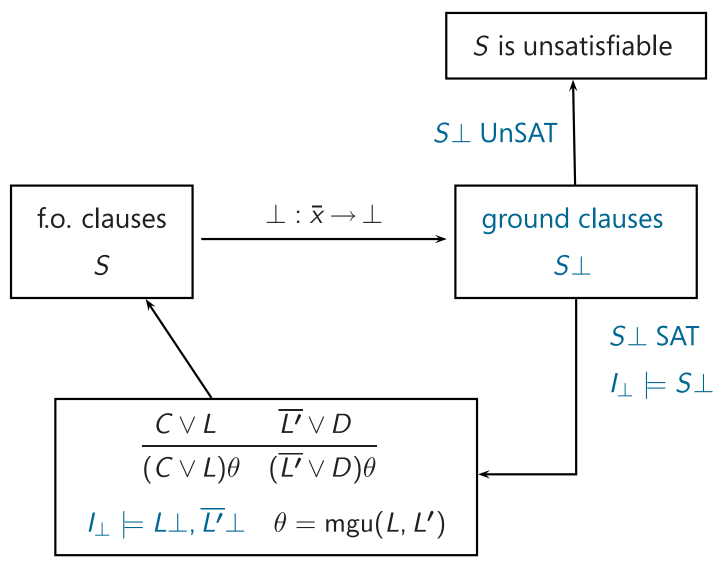
Example

$$\text{Inst-Gen} \quad \frac{Q(x) \vee P(x,b) \qquad \neg P(a,y) \vee R(y)}{Q(a) \vee P(a,b) \qquad \neg P(a,b) \vee R(b)} \quad \text{where}$$

(i) $\theta = \mathrm{mgu}(P(x,b), \neg P(a,y)) = \{x \to a, y \to b\}$, and

(ii) θ is a proper instantiator

Inst-Gen - Outer Loop



Properties and Improvements

- As efficient as possible in propositional case
- Literal selection **in the calculus**
  - Require "back channel" from SAT solver (output of models) to select literals in S (as obtained in $I_\perp$)
  - Restrict inference rule application to selected literals
  - Need only consider instances falsified in I_S
  - Allows to extract model if S is finitely saturated
  - Flexibility: may change models $I_\perp$ arbitrarily during derivation
- Hyper-type inference rule, similar to Hyper Linking [LP92]
- Subsumption deletion by proper subclauses
- Special variables: allows to replace SAT solver by solver for richer fragment (guarded fragment, two-variable fragment)

Resolution vs. Inst-Gen

Resolution	Inst-Gen
$$\frac{(C \vee L) \qquad (\overline{L'} \vee D)}{(C \vee D)\theta} \quad \theta = \mathrm{mgu}(L, L')$$	$$\frac{C \vee L \qquad \overline{L'} \vee D}{(C \vee L)\theta \quad (\overline{L'} \vee D)\theta} \quad \theta = \mathrm{mgu}(L, L')$$
Inefficient in propositional case	Efficient in propositional case
Length of clauses can grow fast	Length of clauses fixed
Recombination of clauses	No recombination of clauses
Subsumption deletion	Subsumption deletion limited
A-Ordered resolution: selection based on term orderings	Selection based on propositional model
Difficult to extract model	Easy to extract model
Decides many fragments, not Bernays-Schönfinkel class	Decides (only?) Bernays-Schönfinkel class
	Doesn't win CASC

Other Two-Level Calculi (I)

DPLL - Davis-Putnam-Logemann-Loveland Procedure

- Weak concept of redundancy already present (purity deletion)

PPI – Primal Partial Instantiation

- Comparable to Inst-Gen, but see [JW05]
- With fixed iterative deepening over term-depth bound

MACE-Style Finite Model Buiding (Different Focus)

- Enumerate finite domains $\{0\}$, $\{0,1\}$, $\{0,1,2\}$, …
- Transform clause set to encode search for model with finite domain
- Apply (incremental) SAT solver
- Complete for finite models, not refutationally complete

Other Two-Level Calculi (II) - HL and SHL

HL - Hyper Linking (Clause Linking)

- Uses hyper type of inference rule, based on simultaneous mgu of nucleus and electrons
- Doesn't use selection (no guidance from propositional model)

SHL - Semantic Hyper Linking

- Uses "back channel" from SAT solver to guide search: find **single** ground clause $C\gamma$ so that $I_\perp \not\models C\gamma$ and add it
- Doesn't use unification; basically guess ground instance, but …
- Practical effectiveness achieved by other devices:
  - Start with "natural" initial interpretation
  - "Rough resolution" to eliminate "large" literals
  - Predicate replacement to unfold definitions [LP89]
- See also important paper [Pla94]

Other Two-Level Calculi (III) - OSHL

OSHL - Ordered Semantic Hyper Linking [PZ97], [PZ00]

- Goal-orientation by chosing "natural" initial interpretation I_0 that falsifies (negated) theorem clause, but satisfies most of the theory clauses
- Stepwisely modify I_0
  Modified interpretation represented as $I_0(L_1, \ldots, L_m)$
  (which is like I_0 except for ground literals $L_1, \ldots, L_m$)
- Completeness via fair enumeration of modifications
- Special treatment of unit clauses
- Subsumption by proper subclauses
- Uses A-ordered resolution as propositional decision procedure

IMs - Classification

Recall:

- Two-level calculi: instance generation separated from SAT solving – may use any SAT solver
- One-level calculi: monolithic, with two modes of operation: First-order mode and propositional mode
  Developed so far:

IM	Extended Calculus
DC	Connection Method, Tableaux
DCTP	Tableaux
OSHT	Hyper Tableaux
Hyper Tableaux NG	Hyper Tableaux
FDPLL	DPLL
ME (successor of FDPLL)	DPLL

Next: one-level calculus: FDPLL (it is simpler than ME)

Motivation for FDPLL (and ME)

DPLL: Successfully used for propositional logic
FDPLL: New lifting of DPLL to **f**irst-order logic

Why?

- **Migrate to the first-order level** those very effective techniques developed for propositional DPLL
- Combine with **successful first-order techniques** (unification, redundancy tests)
- **Theorem Proving**: Alternative to Resolution, Model Elimination
- **Model computation**: Counterexamples, diagnosis, abduction, planning, nonmonotonic reasoning,...

Unification:

$$unify\{P(a,y),\ P(x,f(x))\}$$
$$= \{P(a,f(a))\}$$

Theorem Proving:

$$Axioms \overset{?}{\models} Conjecture$$

Model Computation: Is

$$Axioms \wedge \neg Conjecture$$

satisfiable? I.e.

$$Axioms \overset{?}{\not\models} Conjecture$$

Contents FDPLL Part

- Propositional DPLL as a semantic tree method
- FDPLL calculus
- FDPLL vs. Inst-Gen

Propositional DPLL as a Semantic Tree Method

(1) $A \vee B$ (2) $C \vee \neg A$ (3) $D \vee \neg C \vee \neg A$ (4) $\neg D \vee \neg B$

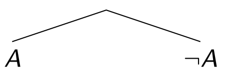
⟨empty tree⟩

$\{\} \not\models A \vee B$
$\{\} \models C \vee \neg A$
$\{\} \models D \vee \neg C \vee \neg A$
$\{\} \models \neg D \vee \neg B$

- A Branch stands for an interpretation
- **Purpose of splitting:** satisfy a clause that is currently falsified
- Close branch if some clause is plainly falsified by it ($\star$)

Propositional DPLL as a Semantic Tree Method

(1) $A \vee B$ (2) $C \vee \neg A$ (3) $D \vee \neg C \vee \neg A$ (4) $\neg D \vee \neg B$

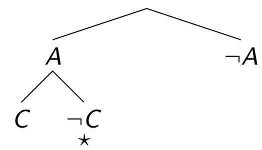


$\{A\} \models A \vee B$
$\{A\} \not\models C \vee \neg A$
$\{A\} \models D \vee \neg C \vee \neg A$
$\{A\} \models \neg D \vee \neg B$

- A Branch stands for an interpretation
- **Purpose of splitting:** satisfy a clause that is currently falsified
- Close branch if some clause is plainly falsified by it ($\star$)

Propositional DPLL as a Semantic Tree Method

(1) $A \vee B$ (2) $C \vee \neg A$ (3) $D \vee \neg C \vee \neg A$ (4) $\neg D \vee \neg B$

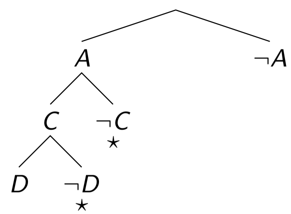


$\{A, C\} \models A \vee B$
$\{A, C\} \models C \vee \neg A$
$\{A, C\} \not\models D \vee \neg C \vee \neg A$
$\{A, C\} \models \neg D \vee \neg B$

- A Branch stands for an interpretation
- **Purpose of splitting:** satisfy a clause that is currently falsified
- Close branch if some clause is plainly falsified by it ($\star$)

Propositional DPLL as a Semantic Tree Method

(1) $A \vee B$ (2) $C \vee \neg A$ (3) $D \vee \neg C \vee \neg A$ (4) $\neg D \vee \neg B$



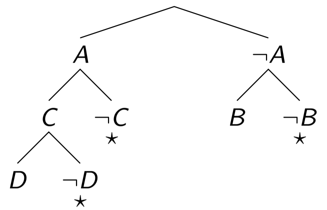
$\{A, C, D\} \models A \vee B$
$\{A, C, D\} \models C \vee \neg A$
$\{A, C, D\} \models D \vee \neg C \vee \neg A$
$\{A, C, D\} \models \neg D \vee \neg B$

Model $\{A, C, D\}$ found.

- A Branch stands for an interpretation
- **Purpose of splitting:** satisfy a clause that is currently falsified
- Close branch if some clause is plainly falsified by it ($\star$)

Propositional DPLL as a Semantic Tree Method

(1) $A \vee B$ (2) $C \vee \neg A$ (3) $D \vee \neg C \vee \neg A$ (4) $\neg D \vee \neg B$

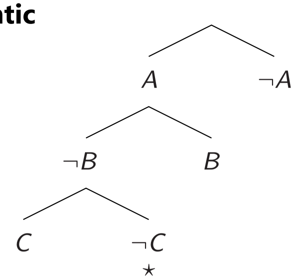
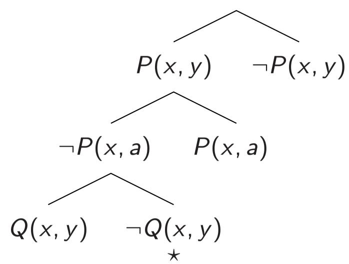


$\{B\} \models A \vee B$
$\{B\} \models C \vee \neg A$
$\{B\} \models D \vee \neg C \vee \neg A$
$\{B\} \models \neg D \vee \neg B$

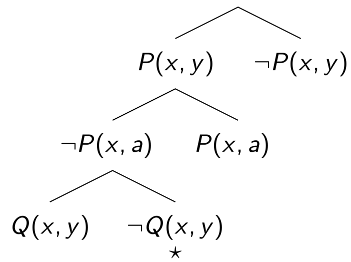
Model $\{B\}$ found.

- A Branch stands for an interpretation
- **Purpose of splitting:** satisfy a clause that is currently falsified
- Close branch if some clause is plainly falsified by it ($\star$)

FDPLL Data Structure: First-Order Semantic Trees

	DPLL	**FDPLL**
Clauses	$B \vee C$	$P(x,y) \vee Q(x,x)$
Semantic Trees		

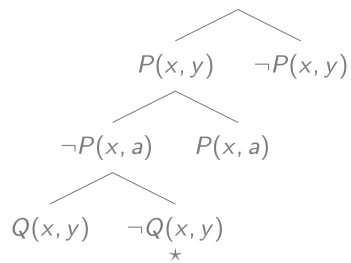
First-Order Semantic Trees



Issues:

- How are variables treated?
  (a) **Universal**, as in Resolution?, (b) **Rigid**, as in Tableaux?
  (c) **Schematic!**
- How to extract an interpretation from a branch?
- When is a branch closed?
- How to construct such trees (calculus)?

First-Order Semantic Trees



Issues:

- How are variables treated?
  (a) Universal, as in Resolution?, (b) Rigid, as in Tableaux?
  (c) Schematic!
- **How to extract an interpretation from a branch?**
- When is a branch closed?
- How to construct such trees (calculus)?

Interpretation Represented by a Branch

Branch $\mathcal{B}$: Interpretation $I_{\mathcal{B}} = \{\ldots\}$:

$P(x, y)$

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

Interpretation Represented by a Branch

Branch $\mathcal{B}$: Interpretation $I_{\mathcal{B}} = \{\ldots\}$:

$P(x, y)$

$\neg P(a, y)$

$P(a, a)$	$P(b, a)$
$P(a, b)$	$P(b, b)$

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

Interpretation Represented by a Branch

Branch $\mathcal{B}$: Interpretation $I_{\mathcal{B}} = \{\ldots\}$:

$P(x, y) \longrightarrow$

$P(a, a)$	$P(b, a)$
$P(a, b)$	$P(b, b)$

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

Interpretation Represented by a Branch

Branch $\mathcal{B}$: Interpretation $I_{\mathcal{B}} = \{\ldots\}$:

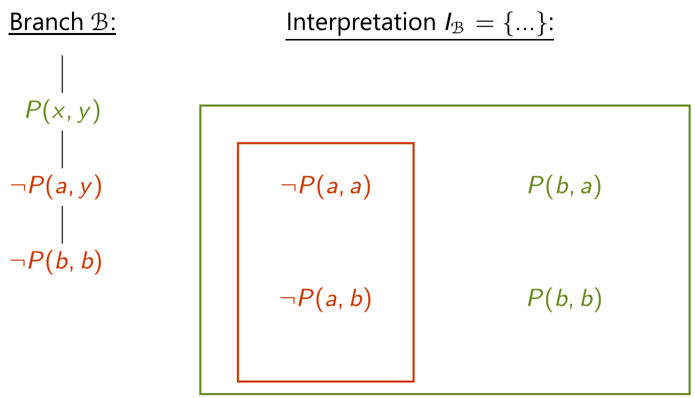
$P(x, y)$

$\neg P(a, y) \longrightarrow$

$\neg P(a, a)$	$P(b, a)$
$\neg P(a, b)$	$P(b, b)$

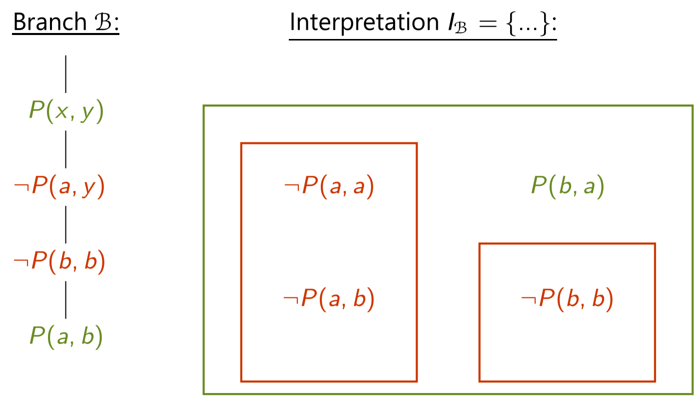
- A branch literal specifies the truth values for all its ground instances, unless there is a more specific literal specifying the opposite truth value

Interpretation Represented by a Branch



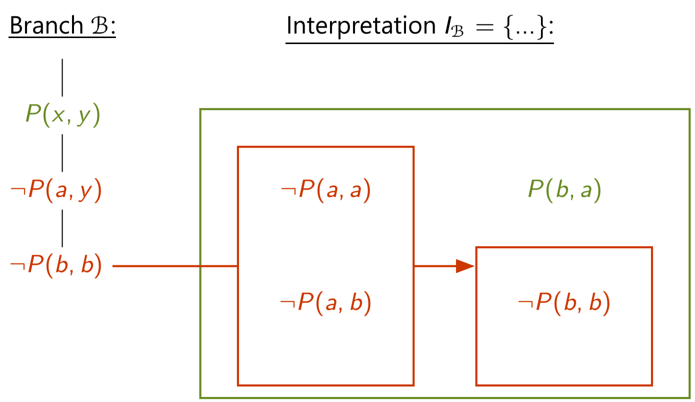
- A branch literal specifies the truth values for all its ground instances, unless there is a more specific literal specifying the opposite truth value

Interpretation Represented by a Branch



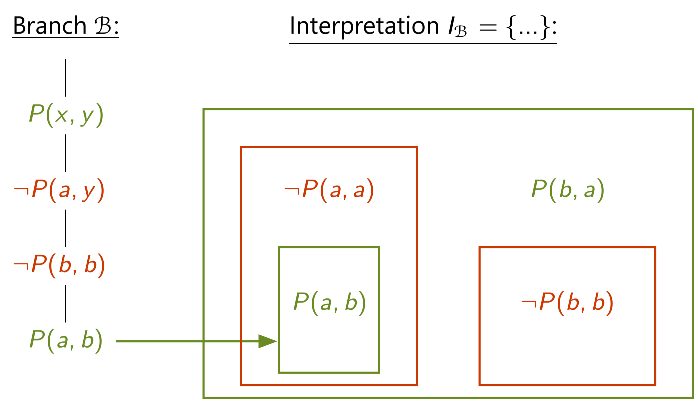
- A branch literal specifies the truth values for all its ground instances, unless there is a more specific literal specifying the opposite truth value

Interpretation Represented by a Branch



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

Interpretation Represented by a Branch



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

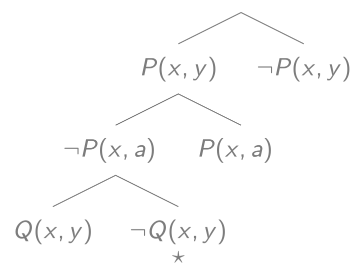
Interpretation Represented by a Branch

Branch $\mathcal{B}$: Interpretation $I_{\mathcal{B}} = \{\ldots\}$:

Branch $\mathcal{B}$	Interpretation $I_{\mathcal{B}}$
$P(x,y)$	
$\neg P(a,y)$	$\{ \neg P(a,a), \quad P(b,a),$
$\neg P(b,b)$	
$P(a,b)$	$P(a,b), \quad \neg P(b,b) \}$

- A branch literal specifies the truth values for all its ground instances, unless there is a more specific literal specifying the opposite truth value
- The order of literals does not matter

First-Order Semantic Trees



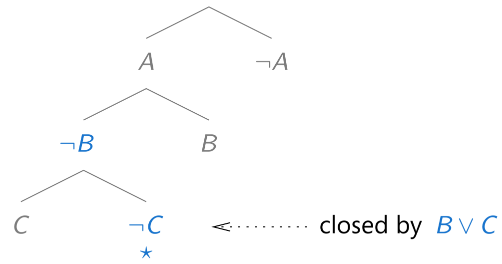
Issues:

- How are variables treated?
  (a) Universal, as in Resolution?, (b) Rigid, as in Tableaux?
  (c) Schema!
- How to extract an interpretation from a branch? ✔
- **When is a branch closed?**
- How to construct such trees (calculus)?

Calculus: Branch Closure

Purpose: Determine if branch elementary contradicts an input clause.

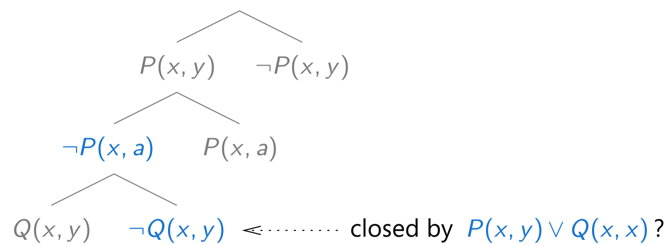
Propositional case:



Calculus: Branch Closure

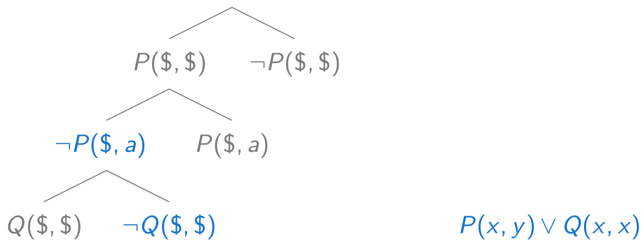
Purpose: Determine if branch elementary contradicts an input clause.

FDPLL case:



Calculus: Branch Closure

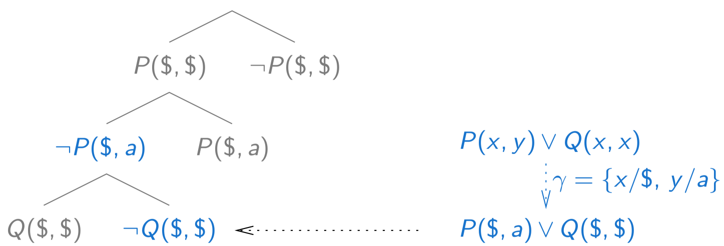
Purpose: Determine if branch elementary contradicts an input clause.
FDPLL case:



1. Replace **all** variables in tree by a constant \$. Gives propositional tree
2.
3.

Calculus: Branch Closure

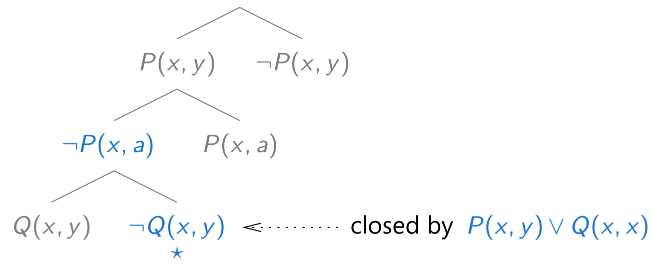
Purpose: Determine if branch elementary contradicts an input clause.
FDPLL case:



1. Replace all variables in tree by a constant \$. Gives propositional tree
2. Compute matcher γ to propositionally close branch
3.

Calculus: Branch Closure

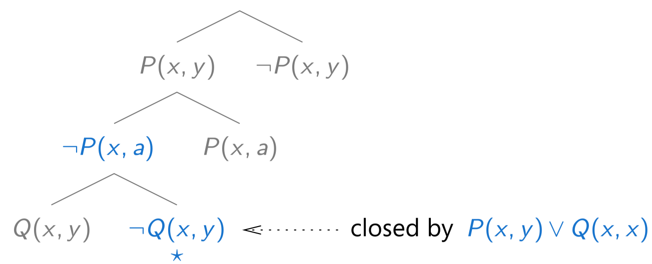
Purpose: Determine if branch elementary contradicts an input clause.
FDPLL case:



1. Replace all variables in tree by a constant \$. Gives propositional tree
2. Compute matcher γ to propositionally close branch
3. Mark branch as closed ($\star$)

Calculus: Branch Closure

Purpose: Determine if branch elementary contradicts an input clause.
FDPLL case:

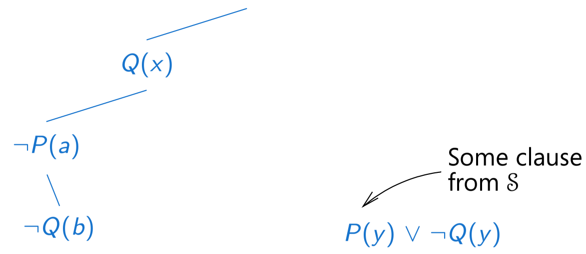


1. Replace all variables in tree by a constant \$. Gives propositional tree
2. Compute matcher γ to propositionally close branch
3. Mark branch as closed ($\star$)

Theorem: FDPLL is sound (because propositional DPLL is sound)

Calculus: The Splitting Rule

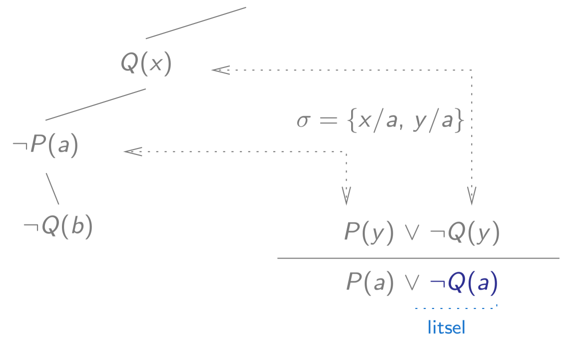
Purpose: Satisfy a clause that is currently "false"



1.
2.
3.

Calculus: The Splitting Rule

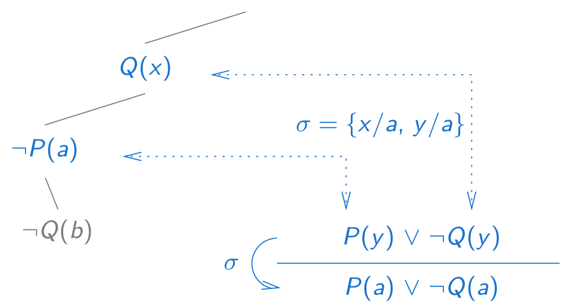
Purpose: Satisfy a clause that is currently "false"



1. Compute simultaneous most general unifier σ
2. Select from clause instance a literal that is not on branch
3.

Calculus: The Splitting Rule

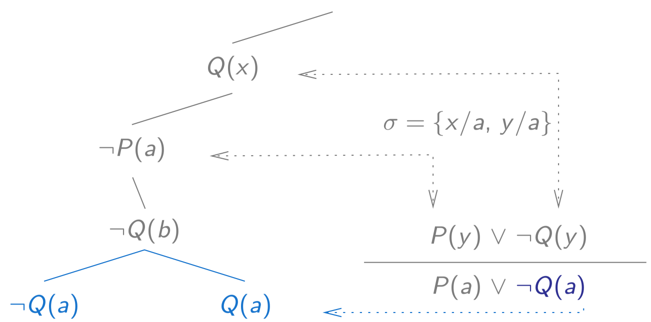
Purpose: Satisfy a clause that is currently "false"



1. Compute simultaneous most general unifier σ
2.
3.

Calculus: The Splitting Rule

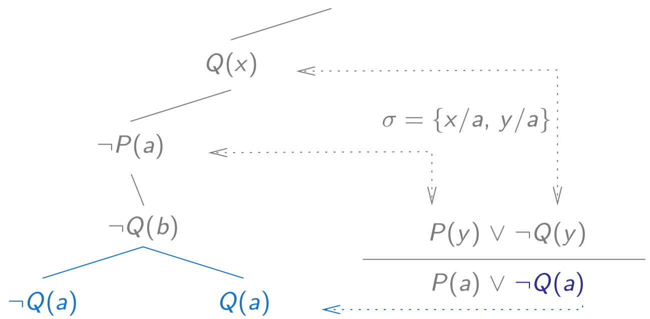
Purpose: Satisfy a clause that is currently "false"



1. Compute simultaneous most general unifier σ
2. Select from clause instance a literal that is not on branch
3. Split with this literal

Calculus: The Splitting Rule

Purpose: Satisfy a clause that is currently "false"

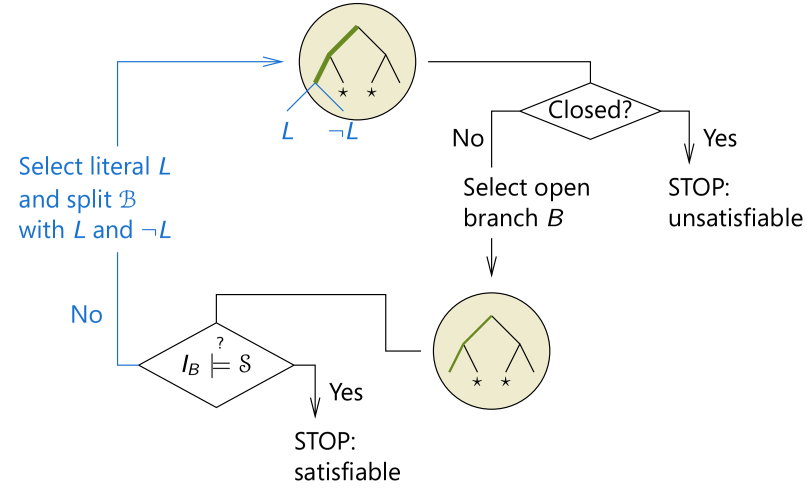


1. Compute simultaneous most general unifier σ
2. Select from clause instance a literal that is not on branch
3. Split with this literal

Proposition: If $I_{\mathcal{B}} \not\models \mathcal{S}$, then split is applicable to some clause from $\mathcal{S}$

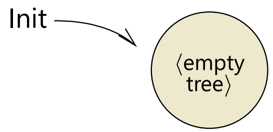
FDPLL Calculus - Main Loop

Input: a clause set S
Output: "unsatisfiable" or "satisfiable" (if it terminates)



FDPLL Calculus - Main Loop

Input: a clause set S
Output: "unsatisfiable" or "satisfiable" (if it terminates)



FDPLL – Model Computation Example

```
(1)  train(X,Y) ; flight(X,Y).      %% train from X to Y or flight from X to
(2)  -flight(sb,X).                 %% no flight from sb to anywhere.
(3)  flight(X,Y) :- flight(Y,X).    %% flight is symmetric.
(4)  connect(X,Y) :- flight(X,Y).   %% a flight is a connection.
(5)  connect(X,Y) :- train(X,Y).    %% a train is a connection.
(6)  connect(X,Z) :- connect(X,Y),  %% connection is a transitive relation.
                     connect(Y,Z).
```

Computed Model (as output by Darwin implementation)

```
+ flight(X, Y)
- flight(sb, X)
- flight(X, sb)
+ train(sb, Y)
+ train(Y, sb)
+ connect(X, Y)
```

Model Evolution Calculus

- Same motivation as for FDPLL: lift propositional DPLL to first-order
- Loosely based on FDPLL, but wouldn't call it "extension"
- Extension of Tinelli's sequent-style DPLL [Tin02]
- See [BT03] for calculus, [BFT05] for implementation "Darwin"

Difference to FDPLL

- Systematic treatment of universal and schematic variables
- Includes first-order versions of unit simplification rules
- Presentation as a sequent-style calculus, to cope with dynamically changing branches and clause sets due to simplification

Model Evolution Implementation: Darwin

- See `http://goedel.cs.uiowa.edu/Darwin/`
- Participated in recent prover competition at CADE-20. Results:

MIX: unsatisfiable problems:

MIX	Vampire 8.0	Vampire 7.0	E 0.9pre3	EP 0.9pre3	Prover9 0705	THEO JN05	Darwin 1.2	Otter 3.3
Attempted	150	150	150	150	150	150	150	150
Solved	137	133	117	117	100	52	32	27
Av. Time	40.64	45.54	24.06	29.97	45.59	48.67	13.14	46.37
Solutions	137	133	0	108	100	52	0	27

EPR: unsat. and satisfiable problems, no function symbols:

EPR	DCTP 10.21p	Darwin 1.2	Paradox 1.3	E 0.9pre3	Vampire 8.0
Attempted	120	120	120	120	120
Solved	117	113	111	96	91
Av. Time	14.94	1.28	7.60	1.52	1.08
Solutions	0	54	55	0	59

Comparison: FDPLL vs. Inst-Gen

FDPLL/ME and Inst-Gen temporarily switch to propositional reasoning. But:

Inst-Gen (and other two-level calculi)

- Use the $\perp$-version $S_\perp$ of the **current clause set** S

$\Rightarrow$ Works **globally**, on clause sets

- Flexible: may switch focus all the time – but memory problem (?)

FDPLL (and other one-level calculi)

- Use the \$-version of the **current branch**

$\Rightarrow$ Works **locally** in context of current branch

- Not so flexible – but don't expect memory problems:
  FDPLL/ME needs not keep **any** input clause instance
  Needs to keeps only instances of input literals (grows slower)

Comparison: Resolution vs. Tableaux vs. IMs

Consider a transitivity clause $P(x,z) \leftarrow P(x,y) \wedge P(y,z)$

Resolution

- Resolution may generate clauses of unbounded length:

$$P(x,z') \leftarrow P(x,y) \wedge P(y,z) \wedge P(z,z')$$
$$P(x,z'') \leftarrow P(x,y) \wedge P(y,z) \wedge P(z,z') \wedge P(z',z'')$$

- Does not decide function-free clause sets
- Complicated to extract model

\+ (Ordered) Resolution very good on some classes, Equality

Comparison: Resolution vs. Tableaux vs. IMs

Consider a transitivity clause $P(x,z) \leftarrow P(x,y) \wedge P(y,z)$

Rigid Variables Approaches (Tableaux, Connection Methods)

- Have to use unbounded number of variants per clause:

$$P(x',z') \leftarrow P(x',y') \wedge P(y',z')$$
$$P(x'',z'') \leftarrow P(x'',y'') \wedge P(y'',z'')$$

- Weak redundancy criteria
- Difficult to exploit proof confluence
  Usual calculi backtrack more than theoretically necessary
  But see [Gie01], [BEF99], [Bec03]

- Model Elimination: **goal-orientedness** compensates drawback

Comparison: Resolution vs. Tableaux vs. IMs

Consider a transitivity clause $P(x,z) \leftarrow P(x,y) \wedge P(y,z)$

Instance Based Methods

- May need to generate and keep **proper** instances of clauses:

$$P(x,z) \leftarrow P(x,y) \wedge P(y,z)$$
$$P(a,z) \leftarrow P(a,y) \wedge P(y,b)$$

- Cannot use subsumption: weaker than Resolution
- Clauses do not grow in length, no recombination of clauses: better than Resolution, same as in rigid variables approaches
+ Need not keep variants: better than rigid variables approaches

Applicability/Non-Applicability of IMs

Other methods (currently?) better at:

- Goal orientation
- Equality, theory reasoning
- Many decidable fragments (Guarded fragment, two-variable fragment)

Suggested applicability for IMs:

- Clause sets without function symbols (except constants)
  All IM's decide this class, unlike e.g. resolution
- Applications:
  - Translation from basic modal logics
  - Reasoning on Logic Programs
  - Finite model generation for arbitrary frst-order formulas (with appropriate transformation)

Instead of Conclusions: An Open Research Problem

- ARM (atomic representation of models) [GP98]
  ARM: set of atoms. Set of all ground instances is an interpretation
- Branches are stronger than ARMs wrt. interpretations that can be finitely represented .
  E.g., for $\Lambda = \{P(u,v), \neg P(u,u)\}$ and $\Sigma_F = \{a/0, f/1\}$ there is no equivalent ARM
- Branches are equivalent to DIGs (Disjunctions of Implicit Generalizations) [FP05]
- Branches cannot represent certain infinite interpretations, e.g. models of the clause set

$$P(x) \vee P(f(x)), \;\; \neg P(x) \vee \neg P(f(x))$$

Thus, FDPLL (and the other IMs) do not terminate on this example

Instead of Conclusions: An Open Research Problem

- ARM (atomic representation of models) [GP98]
  ARM: set of atoms. Set of all ground instances is an interpretation
- Branches are stronger than ARMs wrt. interpretations that can be finitely represented .
  E.g., for $\Lambda = \{P(u,v), \neg P(u,u)\}$ and $\Sigma_F = \{a/0, f/1\}$ there is no equivalent ARM
- Branches are equivalent to DIGs (Disjunctions of Implicit Generalizations) [FP05]
- Branches cannot represent certain infinite interpretations, e.g. models of the clause set

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

Thus, FDPLL (and the other IMs) do not terminate on this example

IM based on more powerful representation formalism?

ture Notes in Artificial Intelligence, pages 60–76. Springer, 1998.

[Bau00] Peter Baumgartner. FDPLL – A First-Order Davis-Putnam-Logeman-Loveland Procedure. In David McAllester, editor, **CADE-17 – The 17th International Conference on Automated Deduction**, volume 1831 of **Lecture Notes in Artificial Intelligence**, pages 200–219. Springer, 2000.

[Bec03] Bernhard Beckert. Depth-first proof search without backtracking for free-variable clausal tableaux. **Journal of Symbolic Computation**, 36:117–138, 2003.

343, Trento, Italy, 1999. Springer.

[BFT05] Peter Baumgartner, Alexander Fuchs, and Cesare Tinelli. Implementing the Model Evolution Calculus. In Stephan Schulz, Geoff Sutcliffe, and Tanel Tammet, editors, **Special Issue of the International Journal of Artificial Intelligence Tools (IJAIT)**, International Journal of Artificial Intelligence Tools, 2005. To appear.

[Bil96] Jean-Paul Billon. The Disconnection Method. In P. Miglioli, U. Moscato, D. Mundici, and M. Ornaghi, editors, **Theorem Proving with Analytic Tableaux and Related Methods**, number 1071 in Lecture Notes in Artificial

mated Deduction, volume 2741 of **Lecture Notes in Artificial Intelligence**, pages 350–364. Springer, 2003.

[BT05] Peter Baumgartner and Cesare Tinelli. The model evolution calculus with equality. In Nieuwenhuis [Nie05], pages 392–408.

[Bun94] Alan Bundy, editor. **Automated Deduction — CADE 12**, LNAI 814, Nancy, France, June 1994. Springer-Verlag.

[CDHM64] T.J. Chinlund, M. Davis, P.G. Hinman, and M.D. McIlroy. Theorem-Proving by Matching. Technical report, Bell Laboratories, 1964.

[CP94] Heng Chu and David A. Plaisted. Semantically Guided First-Order Theorem editors, **CADE-19 Workshop: Model Computation – Principles, Algorithms, Applications**, 2003.

[Dav63] Martin Davis. Eliminating the irrelevant from mechanical proofs. In **Proceedings of Symposia in Applied Amthematics – Experimental Arithmetic, High Speed Computing and Mathematics**, volume XV, pages 15–30. American Mathematical Society, 1963.

[DLL62a] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. **Communications of the ACM**, 5(7), 1962.

[DLL62b] Martin Davis, George Logemann, and Donald Loveland. A machine program

[FP05] Christian G. Fermüller and Reinhard Pichler. Model representation via contexts and implicit generalizations. In Nieuwenhuis [Nie05], pages 409–423.

[Gie01] Martin Giese. Incremental closure of free variable tableaux. In **Proc. International Joint Conference on Automated Reasoning**, volume 2083 of **Lecture Notes in Artificial Intelligence**. Springer Verlag, Berlin, Heidelberg, New-York, 2001.

[GK03] Harald Ganzinger and Konstantin Korovin. New directions in instance-based theorem proving. In **LICS - Logics in Computer Science**, 2003.

[GP00] Matthew L. Ginsberg and Andrew J. Parkes. Satisfiability algorithms and finite quantification. In Anthony G. Cohn, Fausto Giunchiglia, and Bart Selman, editors, **Principles of Knowledge Representation and Reasoning: Proceedings of the Seventh International Conference (KR'2000)**, pages 690–701. Morgan Kauffman, 2000.

[HRCS02] J.N. Hooker, G. Rago, V. Chandru, and A. Shrivastava. Partial Instantiation Methods for Inference in First Order Logic. **Journal of Automated Reasoning**, 28(4):371–396, 2002.

[JW05] Swen Jacobs and Uwe Waldmann. Comparing Instance Generation Methods for Automated Reasoning. In Bernhard

Duplicates with the Hyper-Linking Strategy. **Journal of Automated Reasoning**, 9:25–42, 1992.

[LS01] Reinhold Letz and Gernot Stenz. Proof and Model Generation with Disconnection Tableaux. In Robert Nieuwenhuis and Andrei Voronkov, editors, **LPAR**, volume 2250 of **Lecture Notes in Computer Science**. Springer, 2001.

[McC94] William McCune. A davis-putnam program and its application to finite first-order model search: Qusigroup existence problems. Technical report, Argonne National Laboratory, 1994.

[Nie05] Robert Nieuwenhuis, editor. **Automated Deduction – CADE-20**, volume

Ordered Semantic Hyper Linking. In **Proceedings of Fourteenth National Conference on Artificial Intelligence (AAAI-97)**, 1997.

[PZ00] David A. Plaisted and Yunshan Zhu. Ordered Semantic Hyper Linking. **Journal of Automated Reasoning**, 25(3):167–217, 2000.

[Tin02] Cesare Tinelli. A DPLL-based calculus for ground satisfiability modulo theories. In Giovambattista Ianni and Sergio Flesca, editors, **Proceedings of the 8th European Conference on Logics in Artificial Intelligence (Cosenza, Italy)**, volume 2424 of **Lecture Notes in Artificial Intelligence**. Springer, 2002.