Adaptive *DPLL-Fwd Calculi* for *Knowledge Compilation*, *Model Counting* & *Projection/Forgetting*

M.Fareed Arif, Christoph Wernhard

Knowledge Representation and Reasoning Group,
Technische Universität Dresden;

Feb 11, 2010

Outline

- Preliminaries/Overview/Motivation.
- Model Counting ($\#SAT$).
- Knowledge Compilation ($\mathcal{K}$).
- *Projection* / *Forgetting*.
- LP Calculi.
- Adaptive DPLL-Fwd framework.
- Conclusion.
- Appendix

Boolean Satisfiability Problem

Propositional Satisfiability Problem (*SAT*)

Propositional Satisfiability Problem (*SAT*) is certainly the most studied one since it proved to be *NP-Complete* [Stephen Cook 1971].

The Satisfiability Problem (*SAT*):

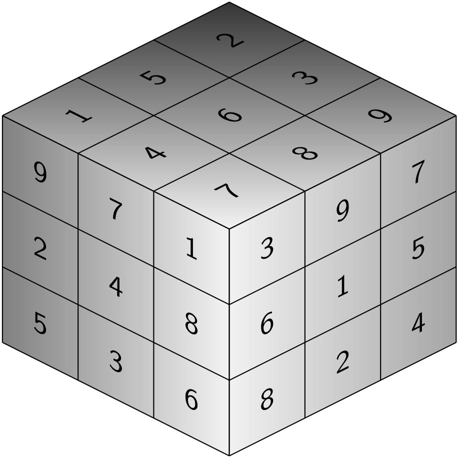
- *Input:*
  - A set of clauses $\mathcal{C}$ over Boolean variables $\mathcal{V}$.
- *Output:*
  - *Yes* (gives a satisfying truth assignment τ for $\mathcal{C}$ if it exists) or *No*.

SAT Applications

List of Applications [Marques 2008]:

- Planning in Artificial Intelligence.
- Bounded Model Checking.
- Software Verification.
- Haplotyping in Bio-informatics.
- Combinational Equivalance Checking.
- Automatic Test-Pattern Generation.
- Proving Automated Termination.
- . . .

Satisfiability Testing Or How to Solve Sudoku Puzzles



Knowledge Representation and Reasoning Group

Activities:

- The Development an efficient C++ DPLL solver.
- The Development of component-based SAT-solver.
- Logical formalization of SAT-solver using:
  - Prolog.
  - Haskell.
  - Rewrite Rules.
- Component-based abstracted Logical formalism for SAT-solver.

Propositional Model Counting

Propositional Model Counting (#*SAT*):

#*SAT* is the problem of computing the number of models for a given propositional formula, i.e., the number of distinct truth assignments to variables for which the formula evaluates to true.

#*SAT* Complexity:

It generalizes *SAT* and is the canonical *#P-Complete Problem*.

For worst-case complexity of (#*SAT*), *for some results it is being hard even for some polynomial-time solvable problems like 2-SAT*.

Model Counting Applications:

$\#SAT$ impact many application areas that are beyond SAT(Under standard complexity theoretic assumption).

$\#SAT$ Applications:

The problem of counting the number of propositional models has number of applications:

- Probabilistic Reasoning.
- Contingency Planning.
- Minimal Model Computation.
- ...

Not surprisingly, the largest formulas we can solve for the model counting problem with state-of-the-art model counters are orders of magnitude smaller than the formulas we can solve with the best SAT solvers [Carla P. Gomes 2008].

Knowledge-based Languages

Knowledge Compilation is the preprocessing underlying propositional theory, it is in some sense harder problem than *SAT*.

Knowledge Compilation (KC):

Transformation of formulas such that they meet a syntactic criteria which permit to execute certain operations like Satisfiability, clausal entailment, model counting etc. in linear time.

Preserving Property:

To compile a propositional theory, one often requires to preserve all satisfying or all consequences of theory.

Testing For Entailment:

A knowledge based $\mathcal{K}$ logically entails a clause C. If so, then of course $(\mathcal{K} \wedge \neg C)$ is *unsatisfiable*. The entailment test amounts to Satisfiability testing.

Negation Normal Form (NNF)

NNF:

A logical formula is said to be in *NNF* if $\wedge$ and $\vee$ are only binary connectives and if all negations are at the atomic level.

$$((\bar{C} \wedge A) \vee D) \wedge (\bar{A} \vee (B \wedge C)) \rightsquigarrow \begin{array}{ccc} \bar{C} & & \\ \wedge & \vee & \mathrm{D} \\ \mathrm{A} & & \\ & \wedge & \\ & & \mathrm{B} \\ \bar{A} & \vee & \wedge \\ & \mathrm{C} & \end{array}$$

Joint Path:

All connected paths in NNF are:
$\{\bar{C}, A, \bar{A}\}, \{\bar{C}, A, B, C\}\}, \{D, \bar{A}\}, \{D, B, C\}$ where $\{\bar{C}, A, \bar{A}\}, \{\bar{C}, A, B, C\}$ are unsatisfiable.

Negation Normal Form (NNF)

$$
\begin{array}{cccccccccccc}
\bar{C} & & & & \bar{C} & & & & & D & & D \\
\wedge & \vee & D & & \wedge & \vee & D & & & \wedge & \vee & \wedge \\
A & & & & A & & & D & & B & & \bar{A} \\
 & \wedge & & \rightsquigarrow & & \wedge & \vee & \wedge & \rightsquigarrow & \wedge & & \\
 & & B & & & B & & \bar{A} & & C & & \\
\bar{A} & \vee & \wedge & & & \wedge & & & & & & \\
 & & C & & & C & & & & & &
\end{array}
$$

CNF vs NNF:

CNF can be factored, i.e., put into more compact *NNF* with application of distributive laws. The time savings is could be significant but the save savings can be dramatic [Panagiotis 2000].

Although most research has restricted attention to CNF may be because the structure of NNF formula can be surprisingly complex.

Knowledge Compilation

DNNF was developed primarily for knowledge compilation.

Decomposable Negation Normal Form (DNNF):

DNNF is a class of formula are in Negation Normal Form (*NNF*) and have the property that atoms are not shared across conjunctives.

Properties of DNNF:

Every *DNNF* formula is automatically a full dissolvent.

- Fully linkless.
- May or may not contain disjoint atoms across conjunctives.

In [Panagiotis 2000] Regular Tableaux and semantic factoring are described as methods for conversion to *DNNF*. Disjoint partitions rule is introduced along with classical DPLL on an input *CNF* formula whose traces record are a *DNNF* [Roberto J. Bayardo 2000].

Linkless Formulas

A pair of formulas are linkless if there is no pair of complementary literal exist.

Linkless Formulas

A pair of formulas are linkless if there is no pair of complementary literal exist.

$$\mathcal{F}_1 = (p \vee q)$$

$$\mathcal{F}_2 = (\neg p \vee q)$$

Linkless out side literal scope $\{p, \neg p\}$.

Linkless Formulas

A pair of formulas are linkless if there is no pair of complementary literal exist.

$$\mathcal{F}_1 = (p \vee q)$$

$$\mathcal{F}_2 = (\neg p \vee q)$$

Linkless out side literal scope $\{p, \neg p\}$.

$$\mathcal{F}_1 = (p \vee q)$$

$$\mathcal{F}_2 = (q \vee r)$$

Fully linkless $\mathcal{S} = \emptyset$.

KB Languages

	ClauseEnt.	*Equiv.*	*#SAT*	*SAT*
DNNF	$\checkmark$	$\times$	$\circ$	$\checkmark$
BBD/FBDD	$\checkmark$	$\checkmark$	$\circ$	$\checkmark$
OBDD	$\checkmark$	$\checkmark$	$\checkmark$	$\checkmark$

Knowledge Base Hierarchy:

$$OBDD \subseteq BDD/FBDD \subset DNNF \subseteq NNF$$

.

Theorem(Linkless + Projection):

If $\mathcal{F}$ is a linkless formula, if $\Gamma = project(\mathcal{F}, \mathcal{S})$ then $\Gamma \models \mathcal{S}$ iff $\mathcal{F} \models \mathcal{S}$ [Panagiotis 2000].

Projection/Forgetting

Projection Operation:

Extraction of knowledge about an arbitrary set of literals within a logical formula [*knowledge-base*].

Projection/Forgetting

Projection Operation:

Extraction of knowledge about an arbitrary set of literals within a logical formula [*knowledge-base*].

$$\mathcal{F} = (p \rightarrow q) \wedge (q \rightarrow r).$$

$$\mathcal{L}(\mathcal{F}) = \{\neg p, q, \neg q, r\}.$$

$$\mathcal{S}(\mathcal{F}) = \{\neg p, r\}.$$

$$\bar{\mathcal{S}}(\mathcal{F}) = \{q, \neg q\}.$$

Projection/Forgetting

Projection Operation:

Extraction of knowledge about an arbitrary set of literals within a logical formula [*knowledge-base*].

$$\mathcal{F} = (p \rightarrow q) \wedge (q \rightarrow r).$$

$$\mathcal{L}(\mathcal{F}) = \{\neg p, q, \neg q, r\}.$$

$$\mathcal{S}(\mathcal{F}) = \{\neg p, r\}.$$

$$\bar{\mathcal{S}}(\mathcal{F}) = \{q, \neg q\}.$$

$$project(\mathcal{F}, \mathcal{S}) \equiv (p \rightarrow r).$$

Projection/Forgetting

Projection Operation:

Extraction of knowledge about an arbitrary set of literals within a logical formula [*knowledge-base*].

$\mathcal{F} = (p \rightarrow q) \wedge (q \rightarrow r).$

$\mathcal{L}(\mathcal{F}) = \{\neg p, q, \neg q, r\}.$

$\mathcal{S}(\mathcal{F}) = \{\neg p, r\}.$

$\bar{\mathcal{S}}(\mathcal{F}) = \{q, \neg q\}.$

$project(\mathcal{F}, \mathcal{S}) \equiv (p \rightarrow r).$

$forget(\mathcal{F}, \bar{\mathcal{S}}) \equiv (p \rightarrow r).$

Projection/Forgetting

Projection Operation:

Extraction of knowledge about an arbitrary set of literals within a logical formula [*knowledge-base*].

$$\mathcal{F} = (p \rightarrow q) \wedge (q \rightarrow r).$$

$$\mathcal{L}(\mathcal{F}) = \{\neg p, q, \neg q, r\}.$$

$$\mathcal{S}(\mathcal{F}) = \{\neg p, r\}.$$

$$\bar{\mathcal{S}}(\mathcal{F}) = \{q, \neg q\}.$$

$$project(\mathcal{F}, \mathcal{S}) \equiv (p \rightarrow r).$$

$$forget(\mathcal{F}, \bar{\mathcal{S}}) \equiv (p \rightarrow r).$$

$$project(\mathcal{F}, \mathcal{S}) \equiv forget(\mathcal{F}, \bar{\mathcal{S}}).$$

Projection/Forgetting

Projection as generalization of second-order quantifier elimination.

Permitting to quantify upon an arbitrary set of ground literals.

Projection/Forgetting

Projection as generalization of second-order quantifier elimination.

Permitting to quantify upon an arbitrary set of ground literals.

$$\mathcal{F} = (\neg p \lor q) \land (\neg q \lor r).$$

Projection/Forgetting

Projection as generalization of second-order quantifier elimination.

Permitting to quantify upon an arbitrary set of ground literals.

$$\mathcal{F} = (\neg p \lor q) \land (\neg q \lor r).$$

$$\exists q.(\mathcal{F}) \equiv project(\mathcal{F}, \{\neg p, r\}) \equiv \neg p \lor r.$$

Projection Syntax

Permitting to quantify upon an arbitrary set of ground literals.
$project(\mathcal{F}, \mathcal{S})$, $forget(\mathcal{F}, \bar{\mathcal{S}})$.

Projection Semantic

$\mathcal{I} \models \textit{project}(\mathcal{F}, \mathcal{S})$ iff there exist an interpretation $\mathcal{J}$ such that $\mathcal{J} \models \mathcal{F}$ and $\mathcal{J} \cap \mathcal{S} \subseteq \mathcal{I}$.

Projection Semantic

$\mathcal{I} \models project(\mathcal{F}, \mathcal{S})$ iff there exist an interpretation $\mathcal{J}$ such that $\mathcal{J} \models \mathcal{F}$ and $\mathcal{J} \cap \mathcal{S} \subseteq \mathcal{I}$.

$$\mathcal{F} = (p \rightarrow q) \wedge (q \rightarrow r).$$

$$\mathcal{S} = \{p, r\}.$$

$$project(\mathcal{F}, \mathcal{S}) = (p \rightarrow r).$$

Projection Semantic

$\mathcal{I} \models project(\mathcal{F}, \mathcal{S})$ iff there exist an interpretation $\mathcal{J}$ such that $\mathcal{J} \models \mathcal{F}$ and $\mathcal{J} \cap \mathcal{S} \subseteq \mathcal{I}$.

$$\mathcal{F} = (p \rightarrow q) \wedge (q \rightarrow r).$$

$$\mathcal{S} = \{p, r\}.$$

$$project(\mathcal{F}, \mathcal{S}) = (p \rightarrow r).$$

$$M_1 = \{p, q, r\}$$

$$M_2 = \{q, r\}$$

$$M_3 = \{r\}$$

$$M_4 = \{\}$$

Projection Semantic

$\mathcal{I} \models project(\mathcal{F}, \mathcal{S})$ iff there exist an interpretation $\mathcal{J}$ such that $\mathcal{J} \models \mathcal{F}$ and $\mathcal{J} \cap \mathcal{S} \subseteq \mathcal{I}$.

$$\mathcal{F} = (p \rightarrow q) \wedge (q \rightarrow r).$$

$$\mathcal{S} = \{p, r\}.$$

$$project(\mathcal{F}, \mathcal{S}) = (p \rightarrow r).$$

$$M_1 = \{p, q, r\}$$

$$M_2 = \{q, r\}$$

$$M_3 = \{r\}$$

$$M_4 = \{\}$$

$$M_1 \cap \mathcal{S} = \{p, r\}$$

$$M_2 \cap \mathcal{S} = \{r\}$$

$$M_3 \cap \mathcal{S} = \{r\}$$

$$M_4 \cap \mathcal{S} = \{\}$$

Properties of Projection

$\mathcal{F}_1 \models_{\mathcal{S}} \mathcal{F}_2 \equiv \mathit{project}(\mathcal{F}_1, \mathcal{S}) \models \mathit{project}(\mathcal{F}_2, \mathcal{S})$.

- $\mathit{project}(\mathcal{F}, \mathcal{S})$ is satisfiable iff $\mathcal{F}$ is satisfiable.
- $\mathcal{F} \models \mathit{project}(\mathcal{F}, \mathcal{S})$.
- If $\mathcal{F}_1 \models \mathcal{F}_2$ then $\mathit{project}(\mathcal{F}_1, \mathcal{S}) \models \mathit{project}(\mathcal{F}_2, \mathcal{S})$.

Proof: [Christoph 2009]

Properties of Projection

- $project(\mathcal{F}, \mathcal{S}) = L$ if $\mathcal{L}(L) \subseteq \mathcal{S}$.
- $project(\mathcal{F}, \mathcal{S}) = \top$ if $\mathcal{L}(L) \cap \mathcal{S} = \emptyset$.
- $project(\mathcal{F}_1 \vee \mathcal{F}_2, \mathcal{S}) \equiv project(\mathcal{F}_1, \mathcal{S}) \vee project(\mathcal{F}_2, \mathcal{S})$.
- $project(\mathcal{F}_1 \wedge \mathcal{F}_2, \mathcal{S}) \not\equiv project(\mathcal{F}_1, \mathcal{S}) \wedge project(\mathcal{F}_2, \mathcal{S})$.

If $< \mathcal{F}_1, \mathcal{F}_2 >$ are linkless out side scope $\mathcal{S}$ then $project(\mathcal{F}_1 \wedge \mathcal{F}_2, \mathcal{S}) \equiv project(\mathcal{F}_1, \mathcal{S}) \wedge project(\mathcal{F}_2, \mathcal{S})$.

Linkless Formulas

A pair of formulas are linkless if there is no pair of complementary literal exist.

Linkless Formulas

A pair of formulas are linkless if there is no pair of complementary literal exist.

$$\mathcal{F}_1 = (p \vee q)$$

$$\mathcal{F}_2 = (\neg p \vee q)$$

Linkless out side literal scope $\{p, \neg p\}$.

Linkless Formulas

A pair of formulas are linkless if there is no pair of complementary literal exist.

$$\mathcal{F}_1 = (p \vee q)$$

$$\mathcal{F}_2 = (\neg p \vee q)$$

Linkless out side literal scope $\{p, \neg p\}$.

$$\mathcal{F}_1 = (p \vee q)$$

$$\mathcal{F}_2 = (q \vee r)$$

Fully linkless $\mathcal{S} = \emptyset$.

LP Relation

[Christoph 2009]

$LP(\mathcal{F}, \mathcal{S}_l, \mathcal{S}_p, \mathcal{F}')$ where

- $\mathcal{F}$ is provided formula.
- $\mathcal{S}_l$ is linkless scope.
- $\mathcal{S}_p$ is projection scope.
- $\mathcal{F}'$ is output formula.
  - $LP(\mathcal{F}, \mathcal{S}_l, \mathcal{S}_p, \mathcal{F}' =_{def}$
  - $\mathcal{F}'$ is linkless outside $\mathcal{S}_l$ (LP-L).
  - $project(\mathcal{F}, \mathcal{S}_p) \equiv project(\mathcal{F}, \mathcal{S}_p)$ (LP-P).

LP Relation Properties

[Christoph 2009]

- Projection Computation: $LP(\mathcal{F}, \mathcal{S}_l, \mathcal{S}_p, \mathcal{F}')$
- Compilation to an equivalent linkless formula: $LP(\mathcal{F}, \mathcal{S}_l, \mathcal{S}_{ALL}, \mathcal{F}')$
- Compute Fully linkless formula: $LP(\mathcal{F}, \emptyset, \mathcal{S}_{ALL}, \mathcal{F}')$
- Satisfiability Checking: $LP(\mathcal{F}, \emptyset, \emptyset, \mathcal{F}')$

LP Calculi

Init: $\top /\!\!/ F_0^*$

Extend:

$$L /\!\!/ F \longrightarrow L/F \wedge \bigvee_{i\in\{1,\ldots,n\}} L_i /\!\!/ F|^*_{L_i} \quad \text{if } \begin{cases} F \models \bigvee_{i\in\{1,\ldots,n\}} L_i, \text{ where } n \geq 1 \\ \text{For } i \in \{1,\ldots,n\}: \\ \quad L_i \in \mathsf{LIT} \text{ and } \mathcal{L}(L_i) \subseteq \mathcal{L}(F) \end{cases}$$

And-Separate:

$$L /\!\!/ \bigwedge_{i\in\{1,\ldots,n\}} F_i \longrightarrow L/\bigwedge_{i\in\{1,\ldots,n\}} F_i \wedge \bigwedge_{i\in\{1,\ldots,n\}} \top /\!\!/ F_i^* \quad \text{if } \begin{cases} n \geq 2 \\ \text{For } i,j \in \{1,\ldots,n\} \text{ s.th. } i \neq j: \\ \quad \langle F_i, F_j \rangle \text{ is linkless outside } S_l \end{cases}$$

True-Up:

$$L/F \wedge (K /\!\!/ \top \vee \bigvee_{i\in\{1,\ldots,n\}} T_i) \longrightarrow L /\!\!/ \top \quad \text{if } \begin{cases} n \geq 0 \\ \mathcal{L}(K) \cap S_p = \emptyset \end{cases}$$

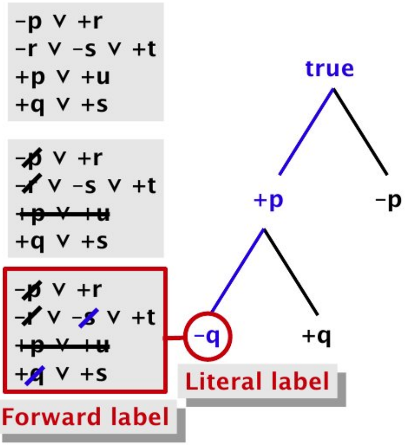
And-True-Up:

$$L/F \wedge \bigwedge_{i\in\{1,\ldots,n\}} \top /\!\!/ \top \longrightarrow L /\!\!/ \top \quad \text{if } n \geq 2$$

True-Below-Cut-Up:

$$L/F \wedge (K /\!\!/ \top \vee \widetilde{K} /\!\!/ \top) \longrightarrow L /\!\!/ \top \quad \text{if } K \text{ is ground}$$

LP Calculi running Example:



Adaptive DPLL-Fwd for Rewrite System

Init:	$\top//\mathcal{F}$		
Split:	$L//\mathcal{F}$	$\Rightarrow$	$L/\mathcal{F} \wedge (\mathcal{K}//\mathcal{F}^*\|_K \vee \bar{K}//\mathcal{F}\|^*_{\bar{\mathcal{K}}})$ if $\mathcal{K}, \bar{\mathcal{K}} \in \mathcal{L}(\mathcal{F})$
Unit:	$L//\mathcal{F}$	$\Rightarrow$	$L/\mathcal{F} \wedge \mathcal{K}//\mathcal{F}^*\|_K$ if $\begin{cases} \mathcal{L}(\mathcal{K}) \subseteq \mathcal{L}(\mathcal{F}) \\ \mathcal{F} \models \mathcal{K} \end{cases}$
True-Up:	$L/\mathcal{F} \wedge (\mathcal{K}//\top \vee \bigvee_{i=1}^{n} \top_i)$	$\Rightarrow$	$L//\top$
Backjump:	$L/\mathcal{F} \wedge \bigvee_{i=1}^{n} \top_i$	$\Rightarrow$	$L//\mathcal{F}$

[Christoph 2009]

Maude Implementation

The Maude system is an implementation of rewriting logic developed at SRI International.

Maude Logical Foundations:

Maude is a declarative language, and a Maude program is a logical theory, and Maude computation is a logical deduction using the axioms specified in the theory/Program. It's modules (rewrite theories) consists of a term-language plus sets of equations and rewrite-rules.

Core Feature:

At the Mathematical level membership logic equational logic is embedded in rewriting logic [Manuel Clavel 2007] $(\Sigma, E \cup A) \hookrightarrow (\Sigma, E \cup A, \phi, R)$.

Quick Overview

Take Home Messages:

- Knowledge Compilation has role to play in SAT domain along with Projection Computation.
- Projection Computation is a generalization of second-order quantifier elimination.
- The LP relation generalizes projection computation and knowledge compilation to linkless formulas.
- The LP Tableau framework abstracted the adapted method for the computation of LP relation.
- Rewrite rules conveniently implemented using Maude system.
- DPLL-Fwd can model classical DPLL like calculi.

For Further Reading

Christoph Wernhard. 2002
Automated Reasoning with Analytic Tableaux and Related Methods: Lecture Notes in Computer Science, "Tableaux for Projection Computation and Knowledge Compilation", pg:325-340.

Christoph Wernhard. 2009
Automated Deduction for Projection Elimination.

Neil V. Murray and Erik Rosenthal. 2003
Tableaux, Path Dissolution, and Decomposable Negation Normal Form for Knowledge Compilation: Lecture Notes in Computer Science "Tableaux, Path Dissolution, and Decomposable Negation Normal Form for Knowledge Compilation", pg:165-180.

Joao Marques-Silva. 2008
Discrete Event Systems, 2008. WODES 2008: "Practical applications of Boolean Satisfiability", pg:74-80.

For Further Reading

Carla P. Gomes, Ashish and Brat Selman. 2009
Handbook of Satisfiability 2009: "Model Counting", pg:633-654.

Manuel Clavel, Francisco Duran and Steven Eker
Lecture Notes in Computer Science: "All About Maude - A High-Performance Logical Framework, How to Specify, Program and Verify Systems in Rewriting Logic".

Roberto J. Bayardo and J. D. Pehoushek. 2000
In AAAI: "Counting Models using Connected Components", pg:157-162.

For Further Reading

Stephen Cook. 1971
In Conference Record of Third Annual ACM Symposium on Theory of Computing: "The complexity of theorem-proving procedures" pg:s 151–158.

Martin Davis and GeorgeLogemann and Donald Loveland. 1962
Journal of the ACM: "A machine program for theorem-proving" 7(5):394-397, July 1962.