Introduction to Knowledge Compilation

Jinbo Huang

NICTA and Australian National University

(based in part on Adnan Darwiche’s tutorial)
Propositional Logic

$\textit{Burglary} \lor \textit{Earthquake} \rightarrow \textit{Alarm}$

Variables

- $A, B, C, \ldots$
- $\textit{Burglary}, \textit{Earthquake}, \textit{Alarm}, \ldots$

Connectives

- $\neg$ (not, negation)
- $\land$ (and, conjunction)
- $\lor$ (or, disjunction)
- $\rightarrow$ (implication)
- $\leftrightarrow$ (biconditional)
Truth of Sentences

\[ \text{Burglary} \lor \text{Earthquake} \rightarrow \text{Alarm} \]

\[ \omega_1 = \{ \text{Burglary, Earthquake, Alarm} \} \]
Truth of Sentences

\[ \text{Burglary} \lor \text{Earthquake} \rightarrow \text{Alarm} \]

\[ \omega_1 = \{ \text{Burglary, Earthquake, Alarm} \} \]

\[ \omega_2 = \{ \text{Burglary, Earthquake, Alarm} \} \]
Sentences as Boolean Functions

Each sentence is a function
- Plug in values (1/0) for variables
- Get value (1/0) for whole sentence
Each sentence is a function

- Plug in values (1/0) for variables
- Get value (1/0) for whole sentence

Function depends only on the mapping, not composition of sentence
Sentences as Boolean Functions

Each sentence is a function
  - Plug in values (1/0) for variables
  - Get value (1/0) for whole sentence

Function depends only on the mapping, not composition of sentence

Different sentences can define same function
  - \( A \lor (B \land C) \)
  - \((A \lor B) \land (A \lor C)\)
Propositional Reasoning

Knowledge base $\Delta$

- $A \land okX \rightarrow \neg B$
- $\neg A \land okX \rightarrow B$
- $B \land okY \rightarrow \neg C$
- $\neg B \land okY \rightarrow C$
Propositional Reasoning

Knowledge base $\Delta$

- $A \land okX \rightarrow \neg B$
- $\neg A \land okX \rightarrow B$
- $B \land okY \rightarrow \neg C$
- $\neg B \land okY \rightarrow C$

Is $A, \neg C$ normal?
- Is it possible for $okX$ and $okY$ to be both 1?
Propositional Reasoning

Knowledge base $\Delta$

- $A \land okX \rightarrow \neg B$
- $\neg A \land okX \rightarrow B$
- $B \land okY \rightarrow \neg C$
- $\neg B \land okY \rightarrow C$

Is $A, \neg C$ normal?

- Is it possible for $okX$ and $okY$ to be both 1?
- Is $(A \land \neg C \land okX \land okY \land \Delta)$ SAT?
Answering a Single Query

\[ A \land \text{okX} \rightarrow \neg B \]
\[ \neg A \land \text{okX} \rightarrow B \]
\[ B \land \text{okY} \rightarrow \neg C \]
\[ \neg B \land \text{okY} \rightarrow C \]

Is \( A, \neg C \) normal?

\[ A \land \text{okX} \rightarrow \neg B \]
\[ \neg A \land \text{okX} \rightarrow B \]
\[ B \land \text{okY} \rightarrow \neg C \]
\[ \neg B \land \text{okY} \rightarrow C \]
\[ A, \neg C, \text{okX}, \text{okY} \]

Satisfiability Algorithm

Jinbo Huang

Introduction to Knowledge Compilation
Multiple Queries?

\[\begin{align*}
A \land \text{okX} & \rightarrow \neg B \\
\neg A \land \text{okX} & \rightarrow B \\
B \land \text{okY} & \rightarrow \neg C \\
\neg B \land \text{okY} & \rightarrow C
\end{align*}\]

Is \(A, \neg C\) normal?

\[\begin{align*}
A \land \text{okX} & \rightarrow \neg B \\
\neg A \land \text{okX} & \rightarrow B \\
B \land \text{okY} & \rightarrow \neg C \\
\neg B \land \text{okY} & \rightarrow C \\
A, \neg C, \text{okX}, \text{okY}
\end{align*}\]

Satisfiability Algorithm
Multiple Queries

\[
\begin{align*}
A \land \text{okX} & \rightarrow \neg B \\
\neg A \land \text{okX} & \rightarrow B \\
B \land \text{okY} & \rightarrow \neg C \\
\neg B \land \text{okY} & \rightarrow C
\end{align*}
\]

Compiler

Compiled Structure

Queries

Evaluator (Polytime)
What to Compile To

\[ \begin{align*}
A \land \text{okX} & \rightarrow \neg B \\
\neg A \land \text{okX} & \rightarrow B \\
B \land \text{okY} & \rightarrow \neg C \\
\neg B \land \text{okY} & \rightarrow C
\end{align*} \]

Queries \rightarrow \text{Compiler} \rightarrow \text{Evaluator (Polytime)}
Target Compilation Forms

Representations of Boolean functions
Systematic space of normal forms
Satisfy different sets of properties
Support different sets of operations
Reasoning by Knowledge Compilation

Identify forms that support desired operations
Reasoning by Knowledge Compilation

Identify forms that support desired operations

Pick most *succinct* one
Identify forms that support desired operations

Pick most succinct one

Compile knowledge into target form
Reasoning by Knowledge Compilation

Identify forms that support desired operations

Pick most *succinct* one

Compile knowledge into target form

Perform reasoning
Applications

Diagnosis
- Is observation normal?
- What components might be broken?

Verification
- Does circuit/program correctly implement specification?
- Can some “bad” behavior happen?

Planning
- Is there sequence of actions to reach goal?

Probabilistic Reasoning
- Most likely explanation for observation?
Agenda

- Languages
- Operations
- Compilers
- Applications
Negation Normal Form (NNF)

AND/OR circuit over literals

\[
\begin{align*}
&\text{or} \\
&\text{and} \\
&\text{or} \\
&\text{and} \\
&\text{or} \\
&\text{and} \\
&\text{and} \\
&\text{and} \\
\end{align*}
\]

\[
\begin{align*}
&\neg A \\
&B \\
&\neg B \\
&A \\
&C \\
&\neg D \\
&D \\
&\neg C \\
\end{align*}
\]
Negation Normal Form (NNF)

Any Boolean function can be represented in NNF (completeness)

Impose conditions/properties over NNF

Obtain subsets/restrictions of NNF

Consider only complete subsets
Decomposable NNF (DNNF)
Smoothness
Deterministic DNNF (d-DNNF)

Introduction to Knowledge Compilation

Diagram:
- NNF
  - d-NNF
  - s-NNF
  - DNNF
  - CO, CE, ME
  - VA, IP, CT
  - d-DNNF
  - EQ?
  - sd-DNNF
Simple Disjunction

\[
\begin{align*}
\neg X & \quad \text{or} \quad Y \\
\text{and} & \quad \text{or} \\
X & \quad \text{or} \\
\end{align*}
\]
Simple Conjunction

or

and  and  and  and

X  Y  Z  -X  -Y  -Z
Introduction to Knowledge Compilation
Prime Implicates (PI)

Conjunction of clauses

Includes every entailed clause

No clause subsumes another
Prime Implicates (PI)

Conjunction of clauses
Includes every entailed clause
No clause subsumes another
Can obtain by running resolution to saturation, removing subsumed clauses
Prime Implicants (IP)

Disjunction of terms

Includes every entailing term

No term subsumes another
Prime Implicants (IP)

Disjunction of terms

Includes every entailing term

No term subsumes another

Can obtain by process dual to resolution
PI, IP

Introduction to Knowledge Compilation
Decision node: \textbf{true, false}, or following fragment where $\alpha, \beta$ are decision nodes
Decision

```
( (true) or (false) )
( (X1 or ~X1) and (X2 and ~X2) )
( (~X3 or X3) and (X2 or ~X2) )
( (false) or (true) )
```

Jinbo Huang
Introduction to Knowledge Compilation
Decision Drawn Compactly

\[
\text{or} \quad \begin{array}{c}
\text{and} \\
X \quad \alpha \quad \neg X \quad \beta
\end{array} \quad \begin{array}{c}
\text{and} \quad X
\end{array}
\]

\[
\begin{array}{c}
\alpha \\
\beta
\end{array}
\]
Binary Decision Diagrams (BDD)
BDD

NNF

d-NNF

s-NNF

DNNF

f-NNF

CO, CE, ME

VA, IP, CT

EQ?

sd-DNNF

MODS

SE, EQ

DNF

IP

VA, IP, SE, EQ

CNF

CO, CE, ME

VA, IP, SE, EQ

PI
Test-once
Ordering
Ordered BDDs (OBDD)
F at least as succinct as G, $F \leq G$, if

$$\forall \Delta \in G, \exists \Delta' \in F, \Delta' \equiv \Delta, |\Delta'| \text{ polynomial in } |\Delta|$$
Succinctness

\[ F \text{ at least as succinct as } G, \ F \leq G, \text{ if} \]
\[ \forall \Delta \in G, \ \exists \Delta' \in F, \ \Delta' \equiv \Delta, \ |\Delta'| \text{ polynomial in } |\Delta| \]

\[ F \text{ more succinct than } G, \ F < G, \text{ if } F \leq G, \ G \nleq F \]
Agenda

- Languages
- Operations
- Compilers
- Applications
Queries

- Consistency (CO)
- Validity (VA)
- Sentential entailment (SE)
- Clausal entailment (CE): $\Delta \models \text{clause}$
- Implicant test (IP): term $\models \Delta$
- Equivalence (EQ)
- Model count (CT)
- Model enumeration (ME)
Transformations

- Projection
- Conditioning
- Conjunction
- Disjunction
- Negation
Decomposability
Knowledge base $\Delta$

- $A \land okX \rightarrow \neg B$
- $\neg A \land okX \rightarrow B$
- $B \land okY \rightarrow \neg C$
- $\neg B \land okY \rightarrow C$
Decomposable
Decomposable

\[
\text{\textcolor{blue}{B}} \quad \text{or} \quad \text{\textcolor{magenta}{\sim A}} \\
\quad \text{and} \quad \text{\textcolor{green}{\sim C}} \\
\quad \text{\textcolor{black}{\sim \text{okX}}} \\
\]

\[
\text{\textcolor{red}{A}} \\
\quad \text{or} \\
\text{\textcolor{green}{\sim \text{okY}}} \\
\text{\textcolor{black}{\sim B}} \\
\text{\textcolor{black}{C}}
\]
Decomposable
Satisfiability

\[ \text{SAT}(A \lor B) \iff \text{SAT}(A) \text{ or } \text{SAT}(B) \]

\[ \text{SAT}(A \land B) \iff \text{SAT}(A) \text{ and } \text{SAT}(B) \]

\[ \text{SAT}(X), \text{SAT}(\neg X), \text{SAT}(\text{true}), \text{SAT}(\text{false}) \]
SAT(A ∨ B) ⇔ SAT(A) or SAT(B)
Satisfiability

\[ SAT(A \lor B) \iff SAT(A) \text{ or } SAT(B) \]

\[ SAT(A \land B) \iff SAT(A) \text{ and } SAT(B) \]
Satisfiability

\[ \text{SAT}(A \lor B) \iff \text{SAT}(A) \text{ or } \text{SAT}(B) \]
\[ \text{SAT}(A \land B) \iff \text{SAT}(A) \text{ and } \text{SAT}(B) \]
\[ \text{SAT}(X), \text{SAT}(\neg X) \]
Satisfiability

\[ \text{SAT}(A \lor B) \iff \text{SAT}(A) \text{ or } \text{SAT}(B) \]
\[ \text{SAT}(A \land B) \iff \text{SAT}(A) \text{ and } \text{SAT}(B) \]
\[ \text{SAT}(X), \text{SAT}(\neg X), \text{SAT}(\text{true}), \text{SAT}(\text{false}) \]
Satisfiability

\[
\begin{align*}
& \text{or} \\
& \text{and} & \text{and} \\
& B & \text{or} & \text{or} & \text{or} & \text{or} & \sim B \\
& \sim A & \text{or} & \sim C & \text{or} & A & C \\
& \sim \text{okX} & \text{or} & \text{okY} & \\
\end{align*}
\]
Satisfiability

![Truth tree diagram]
Partial Decomposability

Decomposable except on okZ

Diagram:
```
  or
   and
     B or
       ~A
     ~C
   ~okZ
  and
   or
     or
       A
     C
   ~B
   ~okZ
```
Clausal Entailment

\[ \Delta \models L_1 \lor L_2 \lor \ldots \lor L_n? \]
Clausal Entailment

$$\Delta \models L_1 \lor L_2 \lor \ldots \lor L_n ?$$

$$\Delta \land \overline{L_1} \land \overline{L_2} \land \ldots \land \overline{L_n} \text{ SAT?}$$
Literal Conjunction

\[ \text{or} \quad \text{and} \quad \text{and} \quad \text{or} \quad \text{or} \quad \text{or} \quad \text{or} \quad \sim B \]

\[ B \quad \text{or} \quad \text{or} \quad \text{or} \quad \text{or} \quad \sim B \quad \sim A \quad \sim C \quad \sim \text{okX} \quad \sim \text{okY} \quad A \quad C \]
Literal Conjunction

A

and

B or

and

false ~C

or

okX

or

okY

and

true ~B

C
Multiple Literals

A → X
B → Y
C

A ~C okX okY

or

and

or

B or

or

~A or

or

~C or

or

A or

and

~B or

~okX or

~okY or

C
Multiple Literals

\[
\begin{align*}
A & \quad X \quad B \quad Y \quad C \\
\text{and} & \quad \text{or} & \quad A \sim C \quad \text{okX} \quad \text{okY} \\
\text{and} & \quad \text{or} \quad \text{or} \quad \text{or} \quad \text{or} \quad \text{or} \quad \sim B \\
\text{false} & \quad \text{true} & \quad \text{true} & \quad \text{false} \\
\text{false} & \quad \text{false} & \quad \text{false} & \quad \text{false}
\end{align*}
\]
Multiple Literals
Partial Decomposability

Decomposable except on $okZ$
Clausal entailment works as long as clause mentions all variables on which we don’t have decomposability.
Partial Decomposability

\[
\begin{array}{c}
\text{or} \\
\text{and} \\
B \text{ or} \ \sim A \\
\sim C \\
\sim \text{okZ} \\
\text{or} \\
\text{and} \\
\sim \text{okZ} \\
A \text{ or} \ \sim B \\
C \\
\end{array}
\]
Partial Decomposability

\[
\begin{align*}
\text{and} & \quad \text{and} \\
\text{or} & \quad \text{okZ} \\
\text{and} & \\
\text{or} & \\
\text{and} & \\
B & \quad \text{or} \\
\text{or} & \quad \text{or} \\
\text{or} & \quad \text{or} \\
\sim A & \\
\sim C & \\
A & \quad C \\
\text{false} & \quad \text{true}
\end{align*}
\]
Partial Decomposability

![Logic Diagram]

The diagram represents a logical expression with a tree structure. The top node is labeled with a conjunction ("and"), the second level contains disjunctions ("or"), and the leaves contain literals and truth values. The expression depicted is:

\[ \text{and} \left( \text{or} \left( B, \text{or} \left( \neg A, \neg C \right) \right), \text{or} \left( A, C \right), \neg B \right) \]

This expression involves logical operators and variables, illustrating the concept of partial decomposability in knowledge compilation.
Projection

Knowledge base

\[
\Delta = A \rightarrow B, B \rightarrow C, C \rightarrow D
\]
Knowledge base

\[ \Delta = A \rightarrow B, B \rightarrow C, C \rightarrow D \]

Projecting on \( A, D \)
Forgetting \( B, C \)
Existentially quantifying \( B, C \)

\[ \exists BC \Delta = A \rightarrow D \]
Projection

Knowledge base

\[ \Delta = A \rightarrow B, B \rightarrow C, C \rightarrow D \]

Projecting on \( A, D \)
Forgetting \( B, C \)
Existentially quantifying \( B, C \)

\[ \exists BC \Delta = A \rightarrow D \]

Definition

\[ \exists X \Delta = (\Delta|_X) \lor (\Delta|_{\neg X}) \]
Projection on CNF
Projection on CNF

Close under resolution
Close under resolution

Remove all clauses mentioning $X$
Projection on DNNF
Projection on DNNF

\[ \exists X (\Delta \lor \Gamma) \]
\[ \exists X (\Delta \lor \Gamma) \]
\[ = (\Delta \lor \Gamma)|_x \lor (\Delta \lor \Gamma)|_{\neg x} \]
Projection on DNNF

\[ \exists X (\Delta \lor \Gamma) \]

\[ = (\Delta \lor \Gamma)|_X \lor (\Delta \lor \Gamma)|_{\neg X} \]

\[ = (\Delta|_X \lor \Gamma|_X) \lor (\Delta|_{\neg X} \lor \Gamma|_{\neg X}) \]
\[ \exists X (\Delta \lor \Gamma) \]
\[ = (\Delta \lor \Gamma)|_X \lor (\Delta \lor \Gamma)|_{\neg X} \]
\[ = (\Delta|_X \lor \Gamma|_X) \lor (\Delta|_{\neg X} \lor \Gamma|_{\neg X}) \]
\[ = (\Delta|_X \lor \Delta|_{\neg X}) \lor (\Gamma|_X \lor \Gamma|_{\neg X}) \]
Projection on DNNF

$$\exists X (\Delta \lor \Gamma)$$

$$= (\Delta \lor \Gamma)|_X \lor (\Delta \lor \Gamma)|_{\neg X}$$

$$= (\Delta|_X \lor \Gamma|_X) \lor (\Delta|_{\neg X} \lor \Gamma|_{\neg X})$$

$$= (\Delta|_X \lor \Delta|_{\neg X}) \lor (\Gamma|_X \lor \Gamma|_{\neg X})$$

$$= (\exists X \Delta) \lor (\exists X \Gamma)$$
Projection on DNNF

\[ \exists X (\Delta \land \Gamma) \]
\[ \exists X (\Delta \land \Gamma) \]
\[ = (\Delta \land \Gamma) \mid_X \lor (\Delta \land \Gamma) \mid_{\neg X} \]
\[ \exists X (\Delta \land \Gamma) \]
\[ = (\Delta \land \Gamma)|_X \lor (\Delta \land \Gamma)|_{\neg X} \]
\[ = (\Delta|_X \land \Gamma|_X) \lor (\Delta|_{\neg X} \land \Gamma|_{\neg X}) \]
Projection on DNNF

\[ \exists X (\Delta \land \Gamma) \]
\[ = (\Delta \land \Gamma) \mid_X \lor (\Delta \land \Gamma) \mid_{\neg X} \]
\[ = (\Delta \mid_X \land \Gamma \mid_X) \lor (\Delta \mid_{\neg X} \land \Gamma \mid_{\neg X}) \]
\[ = (\Delta \land \Gamma \mid_X) \lor (\Delta \land \Gamma \mid_{\neg X}) \]
\[ \exists X (\Delta \land \Gamma) \]
\[ = (\Delta \land \Gamma)|_X \lor (\Delta \land \Gamma)|_{\neg X} \]
\[ = (\Delta|_X \land \Gamma|_X) \lor (\Delta|_{\neg X} \land \Gamma|_{\neg X}) \]
\[ = (\Delta \land \Gamma|_X) \lor (\Delta \land \Gamma|_{\neg X}) \]
\[ = \Delta \land (\Gamma|_X \lor \Gamma|_{\neg X}) \]
\[ \exists X (\Delta \land \Gamma) \]
\[ = (\Delta \land \Gamma) \mid \neg X \lor (\Delta \land \Gamma) \mid X \]
\[ = (\Delta \mid X \land \Gamma \mid X) \lor (\Delta \mid \neg X \land \Gamma \mid \neg X) \]
\[ = (\Delta \land \Gamma \mid X) \lor (\Delta \land \Gamma \mid \neg X) \]
\[ = \Delta \land (\Gamma \mid X \lor \Gamma \mid \neg X) \]
\[ = \Delta \land \exists X \Gamma \]
Minimum Cardinality

Knowledge base

- $A \land okX \rightarrow \neg B$
- $\neg A \land okX \rightarrow B$
- $B \land okY \rightarrow \neg C$
- $\neg B \land okY \rightarrow C$

Models

<table>
<thead>
<tr>
<th>okX</th>
<th>okY</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
</tbody>
</table>

Jinbo Huang

Introduction to Knowledge Compilation
Minimum Cardinality

[Diagram showing a tree structure with logical operations and values]

Introduction to Knowledge Compilation
Minimization (requires smoothness)
Minimization
Minimization
Minimization
Minimization

[Diagram of a minimized Boolean expression]

- A, ¬B, C, D
- A
- ¬B
- A
- ¬A
- B
- ¬B
- A
- C
- ¬D
- D
- ¬C
Minimization

Jinbo Huang
Introduction to Knowledge Compilation
Minimization

Jinbo Huang  Introduction to Knowledge Compilation
## Decomposability

<table>
<thead>
<tr>
<th>Query</th>
<th>DNNF</th>
</tr>
</thead>
<tbody>
<tr>
<td>CO: Consistency</td>
<td>Yes</td>
</tr>
<tr>
<td>VA: Validity</td>
<td></td>
</tr>
<tr>
<td>CE: Clausal entailment</td>
<td>Yes</td>
</tr>
<tr>
<td>SE: Sentential entailment</td>
<td></td>
</tr>
<tr>
<td>IP: Implicant test</td>
<td></td>
</tr>
<tr>
<td>EQ: Equivalence</td>
<td></td>
</tr>
<tr>
<td>MC: Model count</td>
<td></td>
</tr>
<tr>
<td>ME: Model enumeration</td>
<td>Yes</td>
</tr>
<tr>
<td>Transformation</td>
<td>DNNF</td>
</tr>
<tr>
<td>---------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>CD: Conditioning</td>
<td>Yes</td>
</tr>
<tr>
<td>SFO: Single variable forgetting</td>
<td>Yes</td>
</tr>
<tr>
<td>FO: Forgetting</td>
<td>Yes</td>
</tr>
<tr>
<td>(\land): Conjunction</td>
<td></td>
</tr>
<tr>
<td>B(\land): Bounded conjunction</td>
<td></td>
</tr>
<tr>
<td>(\lor): Disjunction</td>
<td>Yes</td>
</tr>
<tr>
<td>B(\lor): Bounded disjunction</td>
<td>Yes</td>
</tr>
<tr>
<td>(\neg): Negation</td>
<td></td>
</tr>
</tbody>
</table>
Determinism
Determinism

\[
\begin{align*}
\text{or} & \quad \text{and} \quad \text{or} \\
\text{and} & \quad \text{and} & \quad \text{and} & \quad \text{and} \\
\text{and} & \quad \text{and} & \quad \text{and} & \quad \text{and} \\
\text{and} & \quad \text{and} & \quad \text{and} & \quad \text{and} \\
\begin{align*}
\neg A & \quad B & \quad \neg B & \quad A \\
C & \quad \neg D & \quad D & \quad \neg C
\end{align*}
\end{align*}
\]
Satisfiability

\[ A \land okX \rightarrow \neg B \]
\[ \neg A \land okX \rightarrow B \]
\[ B \land okY \rightarrow \neg C \]
\[ \neg B \land okY \rightarrow C \]
\[ A, \neg C, okX, okY \]

Is there a satisfying assignment?

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
<td>okX</td>
<td>okY</td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td></td>
</tr>
</tbody>
</table>
Model Count

\[
\begin{align*}
A \land \text{okX} & \rightarrow \neg B \\
\neg A \land \text{okX} & \rightarrow B \\
B \land \text{okY} & \rightarrow \neg C \\
\neg B \land \text{okY} & \rightarrow C \\
A, \neg C, \text{okX}, \text{okY}
\end{align*}
\]

How many satisfying assignments?

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>okX</th>
<th>okY</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>

Counting Algorithm
Counting Models
Counting Graph (requires smoothness)
Counting Models

\[ S = \{ A, \neg B \} \]
Counting Models

\[ S = \{ A, \neg B \} \]
Counting Models

\[ S = \{ A, \neg B \} \]
Counting Graph
$S = \{A, \neg B\}$
Counting Graph
Counting Graph

\[ S = \{A, \neg B, C\} \]
$S = \{A, \neg B, C\} \cup \{\neg D\}$
Retracting Literals

\[ S = \{A, \neg B, C\}\setminus\{\neg B\} \]
$S = \{ A, \neg B, C \} \setminus \{ \neg B \} \cup \{ B \}$
Weighted Model Counting

Knowledge base

- $A \land okX \rightarrow \neg B$
- $\neg A \land okX \rightarrow B$
- $B \land okY \rightarrow \neg C$
- $\neg B \land okY \rightarrow C$

$w(A) = 4$
$w(B) = 7$
$w(C) = 1$
$w(okX) = 9$
$w(okY) = 8$
Weighted Model Counting

Knowledge base

- $A \land okX \rightarrow \neg B$
  \[ w(A) = .4 \]
- $\neg A \land okX \rightarrow B$
  \[ w(B) = .7 \]
- $B \land okY \rightarrow \neg C$
  \[ w(C) = .1 \]
- $\neg B \land okY \rightarrow C$
  \[ w(okX) = .9 \]
  \[ w(okY) = .8 \]
Weighted Model Counting

Knowledge base

- \( A \land okX \rightarrow \neg B \)
- \( \neg A \land okX \rightarrow B \)
- \( B \land okY \rightarrow \neg C \)
- \( \neg B \land okY \rightarrow C \)

Models

<table>
<thead>
<tr>
<th>okX</th>
<th>okY</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>( w(A) )</th>
<th>( w(B) )</th>
<th>( w(C) )</th>
<th>( w(okX) )</th>
<th>( w(okY) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>(.00864)</td>
<td>(.7)</td>
<td>(.1)</td>
<td>(.9)</td>
<td>(.8)</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>(.01944)</td>
<td>(.)</td>
<td>(.)</td>
<td>(.)</td>
<td>(.)</td>
</tr>
</tbody>
</table>

\( w(\Delta) \)

Jinbo Huang
Introduction to Knowledge Compilation
Equivalence Test

\[ \Delta_1(A, B, C, okX, okY) \]
\[ \Delta_2(A, B, C, okX, okY) \]

\[ w(A) = .4 \]
\[ w(B) = .7 \]
\[ w(C) = .1 \]
\[ w(okX) = .9 \]
\[ w(okY) = .8 \]
Equivalence Test

$\Delta_1(A, B, C, okX, okY)$

$\Delta_2(A, B, C, okX, okY)$

If $w(\Delta_1) \neq w(\Delta_2)$,

$w(A) = .4$
$w(B) = .7$
$w(C) = .1$
$w(okX) = .9$
$w(okY) = .8$
Equivalence Test

\[ \Delta_1(A, B, C, okX, okY) \]
\[ \Delta_2(A, B, C, okX, okY) \]

If \( w(\Delta_1) \neq w(\Delta_2) \), \( \Delta_1 \not\equiv \Delta_2 \)

\[ w(A) = .4 \]
\[ w(B) = .7 \]
\[ w(C) = .1 \]
\[ w(okX) = .9 \]
\[ w(okY) = .8 \]
Equivalence Test

\[ \Delta_1(A, B, C, \text{okX}, \text{okY}) \]
\[ \Delta_2(A, B, C, \text{okX}, \text{okY}) \]

If \( w(\Delta_1) \neq w(\Delta_2) \), \( \Delta_1 \not\equiv \Delta_2 \)

If \( w(\Delta_1) = w(\Delta_2) \),

\[ w(A) = .4 \]
\[ w(B) = .7 \]
\[ w(C) = .1 \]
\[ w(\text{okX}) = .9 \]
\[ w(\text{okY}) = .8 \]
Equivalence Test

\[ \Delta_1(A, B, C, okX, okY) \]
\[ \Delta_2(A, B, C, okX, okY) \]

If \( w(\Delta_1) \neq w(\Delta_2), \Delta_1 \not\equiv \Delta_2 \)
If \( w(\Delta_1) = w(\Delta_2), Pr(\Delta_1 \equiv \Delta_2) \)

\[ w(A) = .4 \]
\[ w(B) = .7 \]
\[ w(C) = .1 \]
\[ w(okX) = .9 \]
\[ w(okY) = .8 \]
Equivalence Test

\[ \Delta_1(A, B, C, okX, okY) \]
\[ \Delta_2(A, B, C, okX, okY) \]
\[
\begin{align*}
\text{If } w(\Delta_1) &\neq w(\Delta_2), \quad \Delta_1 \not\equiv \Delta_2 \\
\text{If } w(\Delta_1) &= w(\Delta_2), \quad Pr(\Delta_1 \equiv \Delta_2) > \frac{1}{2}
\end{align*}
\]

\[
\begin{align*}
w(A) &= .4 \\
w(B) &= .7 \\
w(C) &= .1 \\
w(okX) &= .9 \\
w(okY) &= .8
\end{align*}
\]
Equivalence Test

\[ \Delta_1(A, B, C, okX, okY) \]
\[ \Delta_2(A, B, C, okX, okY) \]

If \( w(\Delta_1) \neq w(\Delta_2) \), \( \Delta_1 \not\equiv \Delta_2 \)

If \( w(\Delta_1) = w(\Delta_2) \), \( Pr(\Delta_1 \equiv \Delta_2) > \frac{1}{2} \)

Run test 100 times \( \Rightarrow \) error \( < \frac{1}{10^{30}} \)

\[
\begin{align*}
    w(A) &= 0.4 \\
    w(B) &= 0.7 \\
    w(C) &= 0.1 \\
    w(okX) &= 0.9 \\
    w(okY) &= 0.8
\end{align*}
\]
Weighted Model Counting

Jinbo Huang

Introduction to Knowledge Compilation
Probability of Error

Knowledge base

- $A \land okX \rightarrow \neg B$
- $\neg A \land okX \rightarrow B$
- $B \land okY \rightarrow \neg C$
- $\neg B \land okY \rightarrow C$

$w(A) = .4$
$w(B) = .7$
$w(C) = .1$

$w(okX) = .9$
$w(okY) = .8$
Probability of Error

Knowledge base

- $A \land okX \rightarrow \neg B$
  - $w(A) = .4$
- $\neg A \land okX \rightarrow B$
  - $w(B) = .7$
- $B \land okY \rightarrow \neg C$
  - $w(C) = .1$
- $\neg B \land okY \rightarrow C$
  - $w(okX) = .9$
  - $w(okY) = .8$

Choose weights randomly from $\{1, 2, \ldots, m\}$
Probability of Error

Knowledge base

- $A \land okX \rightarrow \neg B$
- $\neg A \land okX \rightarrow B$
- $B \land okY \rightarrow \neg C$
- $\neg B \land okY \rightarrow C$

Choose weights randomly from $\{1, 2, \ldots, m\}$

If $w(\Delta_1) = w(\Delta_2)$, $Pr(\Delta_1 \equiv \Delta_2) \geq \frac{(m-1)^n}{m^n}$
Probability of Error

Knowledge base

- $A \land okX \rightarrow \neg B$  
  $w(A) = .4$
- $\neg A \land okX \rightarrow B$  
  $w(B) = .7$
- $B \land okY \rightarrow \neg C$  
  $w(C) = .1$
- $\neg B \land okY \rightarrow C$  
  $w(okX) = .9$
- $w(okY) = .8$

Choose weights randomly from \{1, 2, \ldots, m\}

If $w(\Delta_1) = w(\Delta_2)$, $Pr(\Delta_1 \equiv \Delta_2) \geq \frac{(m-1)^n}{m^n}$

$> \frac{1}{2}$ for $m \geq 2n$
Projection Under Determinism

\[ \text{or} \quad \text{and} \quad \text{or} \quad \text{and} \quad \text{or} \quad \text{or} \quad \text{~B} \]

\[ \text{~A} \quad \text{or} \quad \text{~C} \quad \text{~okX} \quad \text{~okY} \quad \text{A} \quad \text{C} \]
Projection Under Determinism

- true
  - ~A
  - ~C

- ~okX

- or
  - and
    - true
    - ~A
    - ~okX
  - or
    - ~C
    - ~okY

- or
  - and
    - A
    - C
  - or
    - ~okY
    - true
### Determinism

<table>
<thead>
<tr>
<th>Query</th>
<th>d-DNNF</th>
</tr>
</thead>
<tbody>
<tr>
<td>CO: Consistency</td>
<td>Yes</td>
</tr>
<tr>
<td>VA: Validity</td>
<td>Yes</td>
</tr>
<tr>
<td>CE: Clausal entailment</td>
<td>Yes</td>
</tr>
<tr>
<td>SE: Sentential entailment</td>
<td></td>
</tr>
<tr>
<td>IP: Implicant test</td>
<td>Yes</td>
</tr>
<tr>
<td>EQ: Equivalence</td>
<td>?</td>
</tr>
<tr>
<td>MC: Model count</td>
<td>Yes</td>
</tr>
<tr>
<td>ME: Model enumeration</td>
<td>Yes</td>
</tr>
</tbody>
</table>
## Transformations in Determinism

<table>
<thead>
<tr>
<th>Transformation</th>
<th>d-DNNF</th>
</tr>
</thead>
<tbody>
<tr>
<td>CD: Conditioning</td>
<td>Yes</td>
</tr>
<tr>
<td>SFO: Single variable forgetting</td>
<td></td>
</tr>
<tr>
<td>FO: Forgetting</td>
<td></td>
</tr>
<tr>
<td>∧: Conjunction</td>
<td></td>
</tr>
<tr>
<td>B∧: Bounded conjunction</td>
<td></td>
</tr>
<tr>
<td>∨: Disjunction</td>
<td></td>
</tr>
<tr>
<td>B∨: Bounded disjunction</td>
<td></td>
</tr>
<tr>
<td>¬: Negation</td>
<td>?</td>
</tr>
</tbody>
</table>
Decision

Introduction to Knowledge Compilation
## Decision

<table>
<thead>
<tr>
<th>Query</th>
<th>FBDD</th>
</tr>
</thead>
<tbody>
<tr>
<td>CO: Consistency</td>
<td>Yes</td>
</tr>
<tr>
<td>VA: Validity</td>
<td>Yes</td>
</tr>
<tr>
<td>CE: Clausal entailment</td>
<td>Yes</td>
</tr>
<tr>
<td>SE: Sentential entailment</td>
<td></td>
</tr>
<tr>
<td>IP: Implicant test</td>
<td>Yes</td>
</tr>
<tr>
<td>EQ: Equivalence</td>
<td>?</td>
</tr>
<tr>
<td>MC: Model count</td>
<td>Yes</td>
</tr>
<tr>
<td>ME: Model enumeration</td>
<td>Yes</td>
</tr>
<tr>
<td>Transformation</td>
<td>FBDD</td>
</tr>
<tr>
<td>--------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>CD: Conditioning</td>
<td>Yes</td>
</tr>
<tr>
<td>SFO: Single variable forgetting</td>
<td></td>
</tr>
<tr>
<td>FO: Forgetting</td>
<td></td>
</tr>
<tr>
<td>∧: Conjunction</td>
<td></td>
</tr>
<tr>
<td>B∧ Bounded conjunction</td>
<td></td>
</tr>
<tr>
<td>∨: Disjunction</td>
<td></td>
</tr>
<tr>
<td>B∨: Bounded disjunction</td>
<td></td>
</tr>
<tr>
<td>¬: Negation</td>
<td>Yes</td>
</tr>
</tbody>
</table>
### Ordering

<table>
<thead>
<tr>
<th>Query</th>
<th>OBDD</th>
</tr>
</thead>
<tbody>
<tr>
<td>CO: Consistency</td>
<td>Yes</td>
</tr>
<tr>
<td>VA: Validity</td>
<td>Yes</td>
</tr>
<tr>
<td>CE: Clausal entailment</td>
<td>Yes</td>
</tr>
<tr>
<td>SE: Sentential entailment</td>
<td>Yes</td>
</tr>
<tr>
<td>IP: Implicant test</td>
<td>Yes</td>
</tr>
<tr>
<td>EQ: Equivalence</td>
<td>Yes</td>
</tr>
<tr>
<td>MC: Model count</td>
<td>Yes</td>
</tr>
<tr>
<td>ME: Model enumeration</td>
<td>Yes</td>
</tr>
</tbody>
</table>
## Ordering

<table>
<thead>
<tr>
<th>Transformation</th>
<th>OBDD</th>
</tr>
</thead>
<tbody>
<tr>
<td>CD: Conditioning</td>
<td>Yes</td>
</tr>
<tr>
<td>SFO: Single variable forgetting</td>
<td>Yes</td>
</tr>
<tr>
<td>FO: Forgetting</td>
<td></td>
</tr>
<tr>
<td>∧: Conjunction</td>
<td></td>
</tr>
<tr>
<td>B ∧ Bounded conjunction</td>
<td>Yes</td>
</tr>
<tr>
<td>∨: Disjunction</td>
<td></td>
</tr>
<tr>
<td>B ∨ Bounded disjunction</td>
<td>Yes</td>
</tr>
<tr>
<td>¬: Negation</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Agenda

- Languages
- Operations
- Compilers
- Applications
Building Compilers

Top-down approach
  ▶ Based on exhaustive search

Bottom-up approach
  ▶ Compiling subformulas, combining results
DPLL Search

\[ \begin{align*}
x \lor y \\
x \lor \neg z \\
\neg w \lor z \lor v \\
v \lor w \lor z
\end{align*} \]

SAT?
DPLL Search
DPLL Search

Terminating condition?

\( v = \text{true} \)

\( x \lor y \)

\( \neg x \lor \neg z \)

\( \neg w \lor z \)

\( v \lor w \lor z \)

\( \text{SAT?} \)

\( v = \text{false} \)

\( x \lor y \)

\( \neg x \lor \neg z \)

\( \neg w \lor z \)

\( \text{SAT?} \)
DPLL Search

Terminating condition?
Exhaustive DPLL

Search the space of all models

- Count models
- Generate representation of all models
Exhaustive DPLL

Search the space of all models
  ▶ Count models
  ▶ Generate representation of all models

Amount of work not necessarily proportional to number of models
The Language of Search

Exhaustive DPLL

Record Trace

Knowledge Compiler

Variations

Languages
Trace of DPLL

\[
\begin{align*}
X \lor Y \\
X \lor \neg Y \lor \neg Z \\
\neg X \lor Y \lor \neg Z
\end{align*}
\]
Trace of DPLL

Run to exhaustion

X \lor Y
\neg X \lor Y \lor \neg Z
Trace of DPLL

\[
\begin{align*}
\text{or} & \quad \text{and} \\
\text{or} & \quad \text{and} \\
\text{or} & \quad \text{and} \\
\text{or} & \quad \text{and} \\
\text{or} & \quad \text{and}
\end{align*}
\]

\[
\begin{align*}
0 & \quad \neg Y \\
Y & \quad \neg Z \\
\neg Z & \quad \neg Y \\
\neg Y & \quad Y \\
& \quad 1
\end{align*}
\]
Trace of DPLL

![DPLL Trace Diagram]
Trace of DPLL

Equivalent to original CNF

Tractable (satisfies some properties)
Dealing with Redundancy

Do not record redundant portions of trace. Try not to solve equivalent subproblems.
Dealing with Redundancy

Do not record redundant portions of trace
Dealing with Redundancy

Do not record redundant portions of trace

Try not to solve equivalent subproblems
Dealing with Redundancy
Dealing with Redundancy

![Diagram showing nodes X, Y, Z, unsat, and sat connected in a network.]

Jinbo Huang
Introduction to Knowledge Compilation
This is an OBDD!
Top-down OBDD Compiler

Exhaustive DPLL, Fixed variable order, Unique nodes

Compile

X v Y
X v ¬Y v ¬Z
¬X v Y v ¬Z
FBDD Compiler

Exhaustive DPLL, Dynamic variable order, Unique nodes

Compile
FBDD vs. OBDD

FBDD more succinct than OBDD

- Dynamic vs. static ordering in SAT
FBDD vs. OBDD

FBDD more succinct than OBDD

- Dynamic vs. static ordering in SAT

Equivalence test for OBDD, probabilistic version for FBDD
FBDD vs. OBDD

FBDD more succinct than OBDD
  ▶ Dynamic vs. static ordering in SAT

Equivalence test for OBDD, probabilistic version for FBDD

Both support model counting
Dealing with Redundancy

Level 1: Unique nodes (done)

Level 2: Avoiding redundant compilation/search
Redundant Compilation

\[ x_5 \lor x_6 \]
\[ x_4 \lor \neg x_5 \lor x_6 \]
\[ x_1 \lor x_3 \lor x_4 \lor x_5 \]
\[ x_2 \lor x_3 \]
\[ x_1 \lor x_2 \lor \neg x_3 \]
Caching

\[ \text{OBDD}(\Delta) = v_1 \]

\[ \text{OBDD}(\Delta|_{v_1=0}) \quad \text{OBDD}(\Delta|_{v_1=1}) \]

\[ v_2 \]

\[ \text{OBDD}(\Delta|_{v_1=0,v_2=0}) \quad \text{OBDD}(\Delta|_{v_1=1,v_2=1}) \]

...
Recursive calls may be made on equivalent subformulas
Caching

\[ \begin{align*}
\nu_5 + \nu_6 \\
\nu_4 + \neg \nu_5 + \nu_6 \\
\nu_1 + \nu_3 + \nu_4 + \nu_5 \\
\nu_2 + \nu_3 \\
\nu_1 + \nu_2 + \neg \nu_3
\end{align*} \]

<table>
<thead>
<tr>
<th>( \nu_1 \nu_2 \nu_3 )</th>
<th>( \Delta' )</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>contradiction</td>
</tr>
<tr>
<td>001</td>
<td>contradiction</td>
</tr>
<tr>
<td>010</td>
<td>( \nu_5 + \nu_6, \ \nu_4 + \neg \nu_5 + \nu_6, \ \nu_4 + \nu_5 )</td>
</tr>
<tr>
<td>011</td>
<td>( \nu_5 + \nu_6, \ \nu_4 + \neg \nu_5 + \nu_6 )</td>
</tr>
<tr>
<td>100</td>
<td>contradiction</td>
</tr>
<tr>
<td>101</td>
<td>( \nu_5 + \nu_6, \ \nu_4 + \neg \nu_5 + \nu_6 )</td>
</tr>
<tr>
<td>110</td>
<td>( \nu_5 + \nu_6, \ \nu_4 + \neg \nu_5 + \nu_6 )</td>
</tr>
<tr>
<td>111</td>
<td>( \nu_5 + \nu_6, \ \nu_4 + \neg \nu_5 + \nu_6 )</td>
</tr>
</tbody>
</table>
After instantiation of $v_1$, $v_2$, $v_3$, $\Delta'$ is either contradictory, or determined by clause $c_3$ alone. $c_3$ can only be in one of two states: satisfied or shrunk to $v_4 + v_5$.
After instantiation of \( v_1 v_2 v_3 \), \( \Delta' \) is either contradictory, or determined by clause \( c_3 \) alone.

- \( c_3 \) can only be in one of two states: satisfied or shrunk to \( v_4 + v_5 \)
Caching

$Cutset_i$: clauses having variable $\leq v_i$ and one $> v_i$
Caching

Cutset$_i$: clauses having variable $\leq v_i$ and one $> v_i$

After instantiation of $v_1 v_2 \ldots v_i$, $\Delta'$ is either contradictory, or determined by states of clauses Cutset$_i$
Caching

*Cutset*$_i$: clauses having variable $\leq v_i$ and one $> v_i$

After instantiation of $v_1 v_2 \ldots v_i$, $\Delta'$ is either contradictory, or determined by states of clauses *Cutset*$_i$

$\#$ of distinct $\Delta' \leq 2^{|\text{Cutset}_i|} + 1$
**Cutset**$_i$: clauses having variable $\leq v_i$ and one $> v_i$

After instantiation of $v_1 v_2 \ldots v_i$, $\Delta'$ is either contradictory, or determined by states of clauses $Cutset_i$

$\#$ of distinct $\Delta' \leq 2^{|Cutset_i|} + 1$

Maintain a cache for each $i$, use value of $Cutset_i$ (bit vector) as key
Complexity

For each $i$, $2^{|Cutset_i|}$ bounds

- # of recursive calls $\text{OBDD}(\Delta, i + 1)$
 Complexity

For each $i$, $2^{|Cutset_i|}$ bounds

- # of recursive calls $\text{OBDD}(\Delta, i + 1)$
- # of entries in cache
Complexity

For each $i$, $2^{|\text{Cutset}_i|}$ bounds

- # of recursive calls $\text{OBDD}(\Delta, i + 1)$
- # of entries in cache
- # of OBDD nodes labeled with $v_{i+1}$
Complexity

For each $i$, $2^{|Cutset_i|}$ bounds

- # of recursive calls $\text{OBDD}(\Delta, i + 1)$
- # of entries in cache
- # of OBDD nodes labeled with $v_{i+1}$

Complexity bound on

- Time complexity of compilation
- Space complexity of compilation
- Size of OBDD
For each $i$, $2^{|Cutset_i|}$ bounds

- # of recursive calls $\text{OBDD}(\Delta, i + 1)$
- # of entries in cache
- # of OBDD nodes labeled with $v_{i+1}$

Complexity bound on

- Time complexity of compilation
- Space complexity of compilation
- Size of OBDD

Linear in # of variables, exponential in cutwidth

- Size of largest cutset of variable order
Beyond BDDs

Plain DPLL $\Rightarrow$ FBDD

Fixed variable ordering $\Rightarrow$ OBDD
Beyond BDDs

Plain DPLL $\Rightarrow$ FBDD

Fixed variable ordering $\Rightarrow$ OBDD

Decomposition $\Rightarrow$ d-DNNF
Decomposition

\[ A \lor B \lor C \quad \neg A \lor \neg B \lor C \]
\[ A \lor D \lor E \quad \neg A \lor \neg D \lor E \]

Diagram:

A

\[ \lor \]
B \lor C
D \lor E

\[ \text{and} \]

B \lor C
D \lor E

\[ \text{and} \]

B
D

\[ \quad \]

C

\[ 0 \]
1

E

\[ \text{and} \]

A

\[ \lor \]
B \lor C
D \lor E

\[ \lor \]
C

\[ \quad \]

D

\[ \text{and} \]

B
D

\[ E \]
Decomposition
Decomposition Methods

Dynamically detect disjoint components

- Most effective, but very expensive
Decomposition Methods

Dynamically detect disjoint components
- Most effective, but very expensive

Static structural analysis
- Constructs a decomposition tree (dtree)
- Does not detect all decompositions
- Low cost
d-DNNF vs. FBDD

- d-DNNF more succinct than FBDD
  - Effectiveness of decomposition
d-DNNF vs. FBDD

d-DNNF more succinct than FBDD
  ▶ Effectiveness of decomposition

Probabilistic equivalence test works for both
d-DNNF vs. FBDD

- d-DNNF more succinct than FBDD
  - Effectiveness of decomposition

- Probabilistic equivalence test works for both

- Support for other queries the same
Fixed variable ordering $\rightarrow$ OBDD
Plain DPLL $\rightarrow$ FBDD
Decomposition $\rightarrow$ d-DNNF
SAT techniques harnessed for knowledge compilation

- Clause learning, efficient unit propagation
Implications

SAT techniques harnessed for knowledge compilation

- Clause learning, efficient unit propagation

Language properties (succinctness/tractability) help characterize power and limitations of search
Take any program $X$ that runs exhaustive DPLL.
Take any program $X$ that runs exhaustive DPLL

Examine traces, if traces $\subseteq L$, then

- $X$ can answer all queries tractable for $L$
- $X$ is hopeless on any input that has no polynomial-size representation in $L$
Limitation of DPLL

Decision nodes

\[
\text{or} \\
\text{and} \quad \text{and} \\
X \quad \alpha \quad \neg X \quad \beta
\]
Limitation of DPLL

Decision nodes

Deterministic nodes
Decomposability without determinism
Compile subformulas

Combine results
Bottom-up OBDD Construction

CNF: \((x + y)(y + z)\)
Variable order: \(x, y, z\)

*Apply*: combines two OBDDs using any binary operator
Bottom-up OBDD Construction

CNF: \((x + y)(y + z)\)
Variable order: \(x, y, z\)

Apply: combines two OBDDs using any binary operator
Bottom-up OBDD Construction

CNF: \((x + y)(y + z)\)
Variable order: \(x, y, z\)

*Apply*: combines two OBDDs using any binary operator
Bottom-up OBDD Construction

CNF: \((x + y)(y + z)\)
Variable order: \(x, y, z\)

*Apply*: combines two OBDDs using any binary operator
Bottom-up OBDD Construction

CNF: \((x + y)(y + z)\)
Variable order: \(x, y, z\)

*Apply*: combines two OBDDs using any binary operator
Bottom-up OBDD Construction

CNF: \((x + y)(y + z)\)
Variable order: \(x, y, z\)

Apply: combines two OBDDs using any binary operator
Bottom-up OBDD Construction

CNF: \((x + y)(y + z)\)
Variable order: \(x, y, z\)

*Apply*: combines two OBDDs using any binary operator
Bottom-up OBDD Construction

CNF: \((x + y)(y + z)\)
Variable order: \(x, y, z\)

Apply: combines two OBDDs using any binary operator
Top-down OBDD Construction

\[ \Delta = (x + y) (y + z) \]

\[ \Delta|_{x=0} = y \]

\[ \Delta|_{x=1} = y + z \]
$$\Delta = (x + y) (y + z)$$

$$\Delta|_{x=0} = y$$

$$\Delta|_{x=1} = y + z$$
\[
\Delta = (x + y) (y + z)
\]

\[
\Delta|_{x=0} = y \quad \Delta|_{x=1} = y + z
\]
$\Delta = (x + y) (y + z)$

$\Delta|_{x=0} = y$

$\Delta|_{x=1} = y + z$
Top-down vs. Bottom-up

Neither dominates the other
Top-down vs. Bottom-up

Neither dominates the other

Open problems: Bottom-up methods for DNNF and d-DNNF
Approximate Compilation

\[ \Delta = \Delta^l \land \Delta^r \]

Jinbo Huang

Introduction to Knowledge Compilation
\[ \Delta = \Delta^l \land \Delta^r \]
\[ \Gamma \models \Delta = \Delta^l \land \Delta^r \]

Incomplete but sound
Approximate Compilation

\[ \Gamma \models \Delta = \Delta' \land \Delta'' \equiv \Sigma \]

Complete but unsound
Approximate Compilation

Diagram:

```
\text{sat}\ \Delta\ ?
\downarrow
\text{sat}\ \Gamma\ ?
\downarrow\ y
\text{sat}\ \Delta\ y
\downarrow\ n
\text{sat}\ \Sigma\ ?
\downarrow\ y
\text{sat}\ \Delta\ ?
```

- **Sound Incomplete**
- **Unsound Complete**
Agenda

- Languages
- Operations
- Compilers
- Applications
Applications

- Probabilistic reasoning
- Model-based diagnosis
Bayesian Network

Diagram showing the relationships between various components such as Battery, Alternator, Fan Belt, Battery Age, Charge Delivered, Battery Power, Starter, Engine Turn Over, Radios, Lights, Gas Gauge, Engine Start, Fuel Pump, Fuel Line, Distributor, and Spark Plugs.
Bayesian Network

If Battery Power = OK, then Lights = ON (99%)

<table>
<thead>
<tr>
<th>Battery Power</th>
<th>Lights</th>
<th>OFF</th>
</tr>
</thead>
<tbody>
<tr>
<td>OK</td>
<td>.99</td>
<td>.01</td>
</tr>
<tr>
<td>WEAK</td>
<td>.20</td>
<td>.80</td>
</tr>
<tr>
<td>DEAD</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Reasoning Tasks

\[ Pr(\neg \text{Gas} | \neg \text{Engine Start})? \]
Reasoning Tasks

\[ Pr(\neg \text{Gas} | \neg \text{Engine Start})? \]

\[ Pr(\text{Alternator} = \text{broken} | \text{Lights} = \text{off})? \]
Reasoning Tasks

\[ Pr(\neg \text{Gas} | \neg \text{Engine Start})? \]

\[ Pr(\text{Alternator} = \text{broken} | \text{Lights} = \text{off})? \]

\[ Pr(X | e): \text{belief update} \]
Reasoning Tasks

\[ Pr(\neg Gas | \neg Engine \_Start) ? \]

\[ Pr(Alternator = broken | Lights = off) ? \]

\[ Pr(X|e): \text{belief update} \]

\[ Pr(X|e) = \frac{Pr(X|e)}{Pr(e)} \]
Reasoning Tasks

\[ Pr(\neg \text{Gas} | \neg \text{Engine Start})? \]

\[ Pr(\text{Alternator} = \text{broken} | \text{Lights} = \text{off})? \]

\[ Pr(X | e): \text{belief update} \]

\[ Pr(X | e) = \frac{Pr(X|e)}{Pr(e)} \]

Reduces to \( Pr(e): \text{marginalization} \)
Joint probability table

<table>
<thead>
<tr>
<th>$A$</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
</tr>
<tr>
<td>true</td>
</tr>
<tr>
<td>true</td>
</tr>
<tr>
<td>false</td>
</tr>
<tr>
<td>false</td>
</tr>
<tr>
<td>false</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$B$</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
</tr>
<tr>
<td>false</td>
</tr>
<tr>
<td>true</td>
</tr>
<tr>
<td>false</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$Pr(AB)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>.03</td>
</tr>
<tr>
<td>.27</td>
</tr>
<tr>
<td>.56</td>
</tr>
<tr>
<td>.14</td>
</tr>
</tbody>
</table>
Marginalization

Joint probability table

<table>
<thead>
<tr>
<th>$A$</th>
<th>$B$</th>
<th>$Pr(AB)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>true</td>
<td>.03</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>.27</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>.56</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>.14</td>
</tr>
</tbody>
</table>

$Pr(\neg B) = .27 + .14 = .41$
Joint probability table

<table>
<thead>
<tr>
<th>$A$</th>
<th>$B$</th>
<th>$Pr(AB)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>true</td>
<td>.03</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>.27</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>.56</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>.14</td>
</tr>
</tbody>
</table>

$Pr(\neg B) = .27 + .14 = .41$

$Pr(e)$: sum entries consistent with $e$
Bayesian Network

If Battery Power = OK, then Lights = ON (99%) ....

<table>
<thead>
<tr>
<th>Battery Power</th>
<th>Lights</th>
</tr>
</thead>
<tbody>
<tr>
<td>OK</td>
<td>.99</td>
</tr>
<tr>
<td>WEAK</td>
<td>.20</td>
</tr>
<tr>
<td>DEAD</td>
<td>0</td>
</tr>
</tbody>
</table>
Joint Probabilities from Bayesian Network

\[
\begin{align*}
\text{Pr}(a_1 b_2 c_2) &= 0.1 \times 0.9 \times 0.9 = 0.081 \\
\text{Pr}(a_2 b_1 c_1) &= 0.9 \times 0.2 \times 0.2 = 0.036
\end{align*}
\]

Compact representation of joint probability table

| $A$ | $\Theta_A$ | $B$ | $\Theta_{B|A}$ | $C$ | $\Theta_{C|A}$ |
|-----|------------|-----|----------------|-----|----------------|
| $a_1$ | 0.1       | $b_1$ | 0.1           | $c_1$ | 0.1 |
| $a_1$ | 0.9       | $b_2$ | 0.9           | $c_2$ | 0.9 |
| $a_2$ | 0.2       | $b_1$ | 0.2           | $c_1$ | 0.2 |
| $a_2$ | 0.8       | $b_2$ | 0.8           | $c_2$ | 0.8 |
Joint Probabilities from Bayesian Network

Pr(a₁b₂c₂) = 0.1 × 0.9 × 0.9 = 0.081
Joint Probabilities from Bayesian Network

\[ Pr(a_1 b_2 c_2) = 0.1 \times 0.9 \times 0.9 = 0.081 \]
\[ Pr(a_2 b_1 c_1) = 0.9 \times 0.2 \times 0.2 = 0.036 \]
Joint Probabilities from Bayesian Network

\[ \Pr(a_1 b_2 c_2) = 0.1 \times 0.9 \times 0.9 = 0.081 \]
\[ \Pr(a_2 b_1 c_1) = 0.9 \times 0.2 \times 0.2 = 0.036 \]

One entry from each CPT consistent with \( e \)
Joint Probabilities from Bayesian Network

\[ \Pr(a_1 b_2 c_2) = 0.1 \times 0.9 \times 0.9 = 0.081 \]
\[ \Pr(a_2 b_1 c_1) = 0.9 \times 0.2 \times 0.2 = 0.036 \]

One entry from each CPT consistent with e

Compact representation of joint probability table
Marginalization

\[ Pr(e) : \text{sum entries (of JPT) consistent with } e \]

- Exponential in \# of variables (not in \( e \))
Marginalization

\( Pr(e) \): sum entries (of JPT) consistent with \( e \)

- Exponential in \( \# \) of variables (not in \( e \))

Represent \( Pr(e) \) as function
Marginalization

\( Pr(e) \): sum entries (of JPT) consistent with \( e \)
- Exponential in \# of variables (not in \( e \))

Represent \( Pr(e) \) as function

Create compact representation of function that supports summation (in polynomial time)
Marginalization

$Pr(e)$: sum entries (of JPT) consistent with $e$
  - Exponential in $\#$ of variables (not in $e$)

Represent $Pr(e)$ as function

Create compact representation of function that supports summation (in polynomial time)
  - Compile knowledge into normal form that supports given operation
## Network Polynomial

<table>
<thead>
<tr>
<th>$A$</th>
<th>$B$</th>
<th>$Pr(AB)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>true</td>
<td>0.03</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>0.27</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>0.56</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>0.14</td>
</tr>
</tbody>
</table>
## Network Polynomial

### Indicator variables

<table>
<thead>
<tr>
<th>$A$</th>
<th>$B$</th>
<th>$Pr(AB)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>true</td>
<td>$0.03 \cdot \lambda_a \cdot \lambda_b$</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>$0.27 \cdot \lambda_a \cdot \lambda_b$</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>$0.56 \cdot \lambda_a \cdot \lambda_b$</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>$0.14 \cdot \lambda_a \cdot \lambda_b$</td>
</tr>
</tbody>
</table>
# Network Polynomial

## Indicator variables

<table>
<thead>
<tr>
<th>$A$</th>
<th>$B$</th>
<th>$Pr(AB)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>true</td>
<td>$.03 \cdot \lambda_a \cdot \lambda_b$</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>$.27 \cdot \lambda_a \cdot \lambda_b$</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>$.56 \cdot \lambda_a \cdot \lambda_b$</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>$.14 \cdot \lambda_a \cdot \lambda_b$</td>
</tr>
</tbody>
</table>

$$f(\lambda_a, \lambda_a, \lambda_b, \lambda_b) = 0.03 \lambda_a \lambda_b + 0.27 \lambda_a \lambda_b + 0.56 \lambda_a \lambda_b + 0.14 \lambda_a \lambda_b$$
Network Polynomial

Indicator variables

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>$Pr(AB)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>true</td>
<td>$0.03 \cdot \lambda_a \cdot \lambda_b$</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>$0.27 \cdot \lambda_a \cdot \lambda_b$</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>$0.56 \cdot \lambda_{\overline{a}} \cdot \lambda_b$</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>$0.14 \cdot \lambda_{\overline{a}} \cdot \lambda_{\overline{b}}$</td>
</tr>
</tbody>
</table>

$$f(\lambda_a, \lambda_{\overline{a}}, \lambda_b, \lambda_{\overline{b}}) = 0.03\lambda_a\lambda_b + 0.27\lambda_a\lambda_{\overline{b}} + 0.56\lambda_{\overline{a}}\lambda_b + 0.14\lambda_{\overline{a}}\lambda_{\overline{b}}$$

$$Pr(ab) = f(1, 0, 0, 1) = 0.27$$
Network Polynomial

Indicator variables

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>Pr((AB))</td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td>.03 (\lambda_a \cdot \lambda_b)</td>
<td></td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>.27 (\lambda_a \cdot \lambda_\overline{b})</td>
<td></td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>.56 (\lambda_\overline{a} \cdot \lambda_b)</td>
<td></td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>.14 (\lambda_\overline{a} \cdot \lambda_\overline{b})</td>
<td></td>
</tr>
</tbody>
</table>

\[
f(\lambda_a, \lambda_{\overline{a}}, \lambda_b, \lambda_{\overline{b}}) = \]
\[
.03\lambda_a\lambda_b + .27\lambda_a\lambda_{\overline{b}} + .56\lambda_{\overline{a}}\lambda_b + .14\lambda_{\overline{a}}\lambda_{\overline{b}}
\]

\[Pr(ab) = f(1, 0, 0, 1) = .27\]

\[Pr(a) = f(1, 0, 1, 1) = .03 + .27 = .3\]
Network Polynomial

Network parameters

\[ \theta_a \rightarrow \theta_{c|a} \]

\[ \theta_b \rightarrow \theta_{b|a} \]
Network Polynomial

Network parameters

![Diagram of network parameters](image)

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>$Pr(ABC)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>$b$</td>
<td>$c$</td>
<td>$\theta_a \theta_b</td>
</tr>
<tr>
<td>$a$</td>
<td>$b$</td>
<td>$\bar{c}$</td>
<td>$\theta_a \theta_b</td>
</tr>
<tr>
<td>$a$</td>
<td>$\bar{b}$</td>
<td>$c$</td>
<td>$\theta_a \theta_{\bar{b}}</td>
</tr>
<tr>
<td>$a$</td>
<td>$\bar{b}$</td>
<td>$\bar{c}$</td>
<td>$\theta_a \theta_{\bar{b}}</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Jinbo Huang
Introduction to Knowledge Compilation
Network Polynomial

Network parameters

\[ \begin{array}{ccc|c}
A & B & C & Pr(ABC) \\
\hline
a & b & c & \lambda_a \lambda_b \lambda_c \theta_a \theta_b \theta_c | a \\
\hline
a & b & \overline{c} & \lambda_a \lambda_b \lambda_{\overline{c}} \theta_a \theta_b \theta_{\overline{c}} | a \\
\hline
a & \overline{b} & c & \lambda_a \lambda_{\overline{b}} \lambda_c \theta_a \theta_{\overline{b}} \theta_c | a \\
\hline
a & \overline{b} & \overline{c} & \lambda_a \lambda_{\overline{b}} \lambda_{\overline{c}} \theta_a \theta_{\overline{b}} \theta_{\overline{c}} | a \\
\hline
\ldots & \ldots & \ldots & \ldots \\
\end{array} \]
Network Polynomial

\[ f = \lambda_a \lambda_b \lambda_c \theta_a \theta_{b|a} \theta_{c|a} + \lambda_a \lambda_b \lambda_{\overline{c}} \theta_a \theta_{b|a} \theta_{\overline{c}|a} + \lambda_a \lambda_{\overline{b}} \lambda_c \theta_a \theta_{\overline{b}|a} \theta_{c|a} + \lambda_a \lambda_{\overline{b}} \lambda_{\overline{c}} \theta_a \theta_{\overline{b}|a} \theta_{\overline{c}|a} + \ldots \]
Network Polynomial

\[ f = \lambda_a \lambda_b \lambda_c \lambda_d \theta_a \theta_b | a \theta_c | a \theta_d | bc \\
+ \lambda_a \lambda_b \lambda_c \lambda_{\bar{d}} \theta_a \theta_b | a \theta_c | a \theta_{\bar{d}} | bc \\
+ \ldots \]
Network Polynomial

\[ f = \lambda_a \lambda_b \lambda_c \lambda_d \theta_a \theta_b | a \theta_c | a \theta_d | bc \]
\[ + \lambda_a \lambda_b \lambda_c \lambda_d \theta_a \theta_b | a \theta_c | a \theta_d | bc \]
\[ + \ldots \]

Each term has 2n variables
Network Polynomial

\[ f = \lambda_a \lambda_b \lambda_c \lambda_d \theta_{a|b} \theta_{c|a} \theta_{d|bc} + \lambda_a \lambda_b \lambda_c \lambda_d \theta_{a|b} \theta_{c|a} \theta_{d|bc} + \ldots \]

Each term has 2\(n\) variables

Each variable has degree 1: multilinear function
Multilinear Function $\mapsto$ Arithmetic Circuit

$$f = \lambda_a \lambda_b \theta_a \theta_{b|a} + \lambda_a \lambda_{b\overline{a}} \theta_a \theta_{b|a} + \lambda_{\overline{a}} \lambda_b \theta_a \theta_{b|\overline{a}} + \lambda_{\overline{a}} \lambda_{b\overline{a}} \theta_{a\overline{a}}$$

Jinbo Huang

Introduction to Knowledge Compilation
Multilinear Function $\mapsto$ Arithmetic Circuit

Propositional theory: $C \land (A \lor \neg B)$

Encode

Multilinear polynomial: $ac + ab + bc + c$

Compile

Smooth d-DNNF

Decode

Arithmetic Circuit
Multilinear Function $\iff$ Propositional Theory

$\Delta = C \land (A \lor \neg B)$

3 models: $abc$, $\overline{abc}$, $\overline{a}bc$
\[ \Delta = C \land (A \lor \neg B) \]

3 models: \( abc, \overline{abc}, \overline{a}bc \)

Model \( \leftrightarrow \) term in multilinear polynomial

- \( abc \leftrightarrow abc \)
- \( \overline{a}bc \leftrightarrow ac \)
- \( \overline{a}bc \leftrightarrow c \)
\[ \Delta = C \land (A \lor \neg B) \]

3 models: \( abc, \overline{abc}, \overline{a}bc \)

Model \( \mapsto \) term in multilinear polynomial

- \( abc \mapsto abc \)
- \( \overline{abc} \mapsto ac \)
- \( \overline{a}bc \mapsto c \)

\( \Delta \) encodes \( f = abc + ac + c \)
Network $\Rightarrow$ Propositional Theory

Jinbo Huang

Introduction to Knowledge Compilation
Network $\Rightarrow$ Propositional Theory

\[ \lambda_a \lor \lambda \overline{a}, \quad \neg \lambda_a \lor \neg \lambda \overline{a} \]
Network \implies Propositional Theory

\[ \lambda_a \lor \lambda_a, \quad \neg \lambda_a \lor \neg \lambda_a \]

\[ \lambda_b \lor \lambda_b, \quad \neg \lambda_b \lor \neg \lambda_b \]
Network $\Rightarrow$ Propositional Theory

\[ \lambda_a \lor \lambda_a, \quad \neg \lambda_a \lor \neg \lambda_a \]
\[ \lambda_b \lor \lambda_b, \quad \neg \lambda_b \lor \neg \lambda_b \]
\[ \lambda_a \leftrightarrow \theta_a, \quad \lambda_a \leftrightarrow \theta_a \]
\[
\begin{align*}
\lambda_a \lor \lambda_{\neg a}, & \quad \neg \lambda_a \lor \neg \lambda_{\neg a} \\
\lambda_b \lor \lambda_{\neg b}, & \quad \neg \lambda_b \lor \neg \lambda_{\neg b} \\
\lambda_a & \iff \theta_a, \quad \lambda_{\neg a} \iff \theta_{\neg a} \\
\lambda_a \land \lambda_b & \iff \theta_{b|a} \\
\lambda_a \land \lambda_{\neg b} & \iff \theta_{\neg b|a} \\
\lambda_{\neg a} \land \lambda_b & \iff \theta_{b|\neg a} \\
\lambda_{\neg a} \land \lambda_{\neg b} & \iff \theta_{\neg b|\neg a}
\end{align*}
\]
Why Logic?

Encoding local structure is easy

Determinism
  ▶ 0/1 probabilities

Context specific independence (CSI)
  ▶ Independence between $X$ and $Y$ given $Z$
Global Structure: Treewidth

\[ O(n \exp(w)) \]
Local Structure: Determinism

- If Battery Power = Dead, then Lights = OFF

<table>
<thead>
<tr>
<th>Lights</th>
<th>ON</th>
<th>OFF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Battery Age</td>
<td>9</td>
<td>.01</td>
</tr>
<tr>
<td>Alternator</td>
<td>.80</td>
<td>1</td>
</tr>
<tr>
<td>Fan Belt</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Charge Delivered</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Battery</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Starter</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Radio</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fuel Pump</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fuel Line</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Distributor</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Spark Plugs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Engine Start</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Jinbo Huang

Introduction to Knowledge Compilation
Local Structure: Determinism

Assert $\neg \theta_{\text{lights}=\text{on}|\text{battery_power}=\text{dead}}$
Local Structure: CSI

[Diagram of a car's electrical and mechanical system, showing components like battery age, alternator, fan belt, charge delivered, battery, battery power, radio, lights, engine turn over, starter, gas gauge, engine start, fuel pump, fuel line, distributor, and spark plugs.]

Jinbo Huang

Introduction to Knowledge Compilation
Local Structure: CSI

Context Specific Independence (CSI)
If $X$, $Y$ independent given $Z = z$, collapse $\theta_{x|yz}$ and $\theta_{x|\overline{yz}}$ into one variable
Exploiting Local Structure

Reasoning not necessarily exponential in treewidth
Probabilistic Reasoning via Knowledge Compilation

Belief network:
- X
- Y
- Edges indicate conditional dependencies

Arithmetic Circuit:
- λ_x
- λ_y
- θ_{x|y}
- Operations: +, *

CNF:
- \lambda_x \lor \lambda_{\neg x}
- \lambda_x \rightarrow \lambda_{\neg x}
- \lambda_x \land \lambda_y \rightarrow \theta_{x|y}
- ... ...

Smooth d-DNNF:
- λ_x
- λ_y
- θ_{x|y}
- ~λ_x

Jinbo Huang
Introduction to Knowledge Compilation
Applications

- Probabilistic reasoning
- Model-based diagnosis
An Abnormal System

- Model: circuit description + expected abnormal behavior
  - \( \text{Pr}(\text{broken}) = 0.1; \) if broken, \( \text{Pr}(\text{output} = 1) = 0.5 \)
- Diagnosis: set of faulty gates
- Can’t always be determined
Measure system variables until faults are located

Assume equal measurement costs—minimize \# of measurements

Optimal policy (tree) exists, intractable

Greedily maximize utility of each measurement
okA → (A ↔ (J ∧ D)), okJ → (J ↔ ¬P)

▶ okX → NORMALBEHAVIOR(X)

▶ Pr(okA) = Pr(okJ) = 0.9

¬okA → (A ↔ θA), ¬okJ → (J ↔ θJ)

▶ Pr(θA) = Pr(θJ) = 0.5
System as Bayesian Network

\[ \text{okJ} \rightarrow \text{J} \rightarrow \text{okA} \rightarrow \text{A} \rightarrow \text{okK} \rightarrow \text{K} \rightarrow \text{V} \]

\[ \text{okD} \rightarrow \text{D} \rightarrow \text{B} \rightarrow \text{okB} \rightarrow \text{Q} \rightarrow \text{P} \]

\[ \text{okV} \rightarrow \text{V} \]

\[ \text{1} \rightarrow \text{A} \rightarrow \text{B} \rightarrow \text{D} \rightarrow \text{V} \]

\[ \text{1} \rightarrow \text{J} \rightarrow \text{B} \rightarrow \text{D} \rightarrow \text{V} \]

\[ \text{okA} \rightarrow \text{A} \rightarrow \text{okK} \rightarrow \text{K} \rightarrow \text{V} \]

\[ \text{okJ} \rightarrow \text{J} \rightarrow \text{okA} \rightarrow \text{A} \rightarrow \text{okK} \rightarrow \text{K} \rightarrow \text{V} \]
System as Bayesian Network

\[ P \theta_P \]

\[ okJ \theta_{okJ} \]

\[
\begin{array}{c|c}
P & \theta_P \\
\hline
1 & 0.5 \\
0 & 0.5 \\
\end{array}
\]

\[
\begin{array}{c|c}
okJ & \theta_{okJ} \\
\hline
1 & 0.9 \\
0 & 0.1 \\
\end{array}
\]

\[
\begin{array}{c|c|c|c}
P & okJ & J & \theta_{J|P,okJ} \\
\hline
1 & 1 & 1 & 0 \\
1 & 1 & 0 & 1 \\
1 & 0 & 1 & 0.5 \\
1 & 0 & 0 & 0.5 \\
0 & 1 & 1 & 1 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0.5 \\
0 & 0 & 0 & 0.5 \\
\end{array}
\]
okA → (A ↔ (J ∧ D))
okJ → (J ↔ ¬P)
Pr(okA) = Pr(okJ) = 0.9
¬okA → (A ↔ θ_A)
¬okJ → (J ↔ θ_J)
Pr(θ_A) = Pr(θ_J) = 0.5
observation: A ∧ P ∧ D
Compilation into Arithmetic Circuit

\[ \text{okA} \rightarrow (A \leftrightarrow (J \land D)) \]
\[ \text{okJ} \rightarrow (J \leftrightarrow \neg P) \]
\[ \text{Pr}(\text{okA}) = \text{Pr}(\text{okJ}) = 0.9 \]
\[ \neg \text{okA} \rightarrow (A \leftrightarrow \theta_A) \]
\[ \neg \text{okJ} \rightarrow (J \leftrightarrow \theta_J) \]
\[ \text{Pr}(\theta_A) = \text{Pr}(\theta_J) = 0.5 \]

observation: \( A \land P \land D \)

\[ \text{Pr}(\neg \text{okJ} \mid \text{obs})? \]
Compilation into Arithmetic Circuit

\[ \text{okA} \rightarrow (A \leftrightarrow (J \land D)) \]
\[ \text{okJ} \rightarrow (J \leftrightarrow \neg P) \]
\[ Pr(\text{okA}) = Pr(\text{okJ}) = 0.9 \]
\[ \neg \text{okA} \rightarrow (A \leftrightarrow \theta_A) \]
\[ \neg \text{okJ} \rightarrow (J \leftrightarrow \theta_J) \]
\[ Pr(\theta_A) = Pr(\theta_J) = 0.5 \]

**Observation:** \( A \land P \land D \)

\[ Pr(\neg \text{okJ} \mid \text{obs})? \]
\[ Pr(\ldots \mid \text{obs} \land \ldots) \]

in linear time
Measurement Selection

Entropy over each system variable

\[ \xi(X) = -(p_x \log p_x + p_{\bar{x}} \log p_{\bar{x}}) \]

- reflects expected info gain by measurement
- measure variable with highest \( \xi(X) \)
- \( Pr(X) \) obtainable in linear time post compilation
Entropy over each system variable

\[ \xi(X) = -(p_X \log p_X + p_{\bar{X}} \log p_{\bar{X}}) \]

- reflects expected info gain by measurement
- measure variable with highest \( \xi(X) \)
- \( Pr(X) \) obtainable in linear time post compilation

Pick component with highest \( Pr(\neg okX) \), pick its variable with highest entropy
Measurement Selection

Good in terms of diagnostic cost

Generally efficient: exploits structure
Measurement Selection

Good in terms of diagnostic cost

Generally efficient: exploits structure

What if compilation unsuccessful or too large?
Identify cones, treat as abstract component

Cone: subsystem where all components are dominated by some component $X$

$X$ dominates $Y$ if any path from $Y$ to output contains $X$

Identification automatic, efficient
Abstraction

- # of components, # of health variables reduced
- Compilation scales to larger systems
- Look inside cone only if cone as a whole identified as faulty in abstract level—then compile & diagnose recursively
Previous full encoding

- $\text{ok}A \rightarrow (A \leftrightarrow (J \land D))$
- $\text{ok}J \rightarrow (J \leftrightarrow \neg P)$

- need $\text{ok}X$ for every component
Abstraction: Encoding of Cone

Previous full encoding

- \( okA \rightarrow (A \leftrightarrow (J \land D)) \)
- \( okJ \rightarrow (J \leftrightarrow \neg P) \)
- need \( okX \) for every component

Abstract encoding

- \( okA \rightarrow (A \leftrightarrow (J \land D)) \)
- \( \neg okA \rightarrow (A \nleftrightarrow (J \land D)), J \leftrightarrow \neg P \)

- single \( okA \) for root
- explicitly force wrong output under \( \neg okA \)
- need to compute \( Pr(okA) \)
XOR healthy cone & actual cone

Compute \( Pr(output = 1) \) by compilation

Done once (recursively) for all cones as preprocessing
Abstraction

System size reduced by abstraction

- Initially, only measure variables outside cones
- Look inside cone only if root identified as faulty
- Diagnose cone recursively

Empirical performance

- Extended solvable benchmarks from c1355 (546 gates) to c2670 (1193 gates, 160 abstract gates)
- Diagnostic cost similar to baseline (on problems solvable by both)
Abstraction

System size reduced by abstraction
- Initially, only measure variables outside cones
- Look inside cone only if root identified as faulty
- Diagnose cone recursively

Empirical performance
- Extended solvable benchmarks from c1355 (546 gates) to c2670 (1193 gates, 160 abstract gates)
- Diagnostic cost similar to baseline (on problems solvable by both)
For Even Larger Systems

Intractable even after abstraction

Idea: reduce abstraction size by creating more cones
Any More Cones?
Component Cloning

\begin{equation*}
\begin{array}{c}
\text{P} \\
\text{Q} \\
\text{R} \\
\end{array}
\end{equation*}

\begin{equation*}
\begin{array}{c}
\text{J} \\
\text{E} \\
\text{A} \\
\end{array}
\end{equation*}

\begin{equation*}
\begin{array}{c}
\text{B} \\
\text{D} \\
\text{K} \\
\end{array}
\end{equation*}

\begin{equation*}
\begin{array}{c}
\text{K} \\
\text{V} \\
\end{array}
\end{equation*}
Component Cloning

Pick component
Create one or more clones
Distribute parents among clones

Jinbo Huang
Introduction to Knowledge Compilation
Component Cloning

- Pick component
- Create one or more clones
- Distribute parents among clones
Component Cloning: Choices

Pick component

- components in abstraction that are not roots of cones

Create one or more clones

- partition parents into $P_1, P_2, \ldots, P_q$ such that each $P_i$ lies entirely in a cone
- create $q - 1$ clones

Distribute parents among clones

- give each clone one $P_i$
Component Cloning

- Smaller abstraction, easier to compile & diagnose
- What’s the catch?
Component Cloning

- Smaller abstraction, easier to compile & diagnose
- What’s the catch?

- New system is a relaxation
  - two copies can fail independently
Component Cloning

- Smaller abstraction, easier to compile & diagnose
- What’s the catch?

- New system is a relaxation
  - two copies can fail independently
- Solution: filter spurious diagnoses (insist on same health state for all copies)
Component Cloning

- Smaller abstraction, easier to compile & diagnose
- What’s the catch?

- Probability space different, skewing measurement selection heuristic
Component Cloning

- Smaller abstraction, easier to compile & diagnose
- What’s the catch?

- Probability space different, skewing measurement selection heuristic
- Solution: none needed, diagnostic cost only slightly affected
Component Cloning

Abstraction size substantially reduced

Extended solvable benchmarks from c2670 (1193 gates, 160 abstract gates) to c7552 (3512 gates, 545 abstract gates, 378 after cloning)
Diagnosis via Compilation and Abstraction

Measurement point selection computed by compiling system as Bayesian network

Abstraction by cone identification

Component cloning to reduce abstraction size
The End

- Languages
- Operations
- Compilers
- Applications