A dynamic programming algorithm for prime implicates.

Andrew Matusiewicz
What are prime implicates and why care?

- Smallest, simplest disjunctions implied by a statement
What are prime implicates and why care?

- Smallest, simplest disjunctions implied by a statement
- Knowing all prime implicates, clausal implication is easy
What are prime implicates and why care?

- Smallest, simplest disjunctions implied by a statement
- Knowing all prime implicates, clausal implication is easy
- Used in non-monotonic and abductive reasoning
What are prime implicates and why care?

- Smallest, simplest disjunctions implied by a statement
- Knowing all prime implicates, clausal implication is easy
- Used in non-monotonic and abductive reasoning
- Dual, prime implicants, used for circuit minimization

Research partner uses in code gen. for embedded systems
(Sandeep Shukla, Virginia Tech.)
What are prime implicates and why care?

- Smallest, simplest disjunctions implied by a statement
- Knowing all prime implicates, clausal implication is easy
- Used in non-monotonic and abductive reasoning
- Dual, prime implicants, used for circuit minimization
- Research partner uses in code gen. for embedded systems (Sandeep Shukla, Virginia Tech.)
Propositional Clauses

- Propositional variables $v_1, v_2, \ldots$ are either true or false
- Connectives $\land$ (and), $\lor$ (or), $\neg$ (not).
- A literal $\ell$ is a negated or non-negated variable: $v_i, \neg v_i$

- A clause is an “or” of literals:
  
  \[
  (v_1 \lor \neg v_3 \lor v_2 \lor v_9) = \{v_1, \neg v_3, v_2, v_9\}
  \]
Prime Implicates

A clause $C$ is an "implicate" of a formula $F$ iff the statement

$$F \Rightarrow C$$

is a tautology.
Prime Implicates

A clause $C$ is an "implicate" of a formula $F$ iff the statement

$$F \Rightarrow C$$

is a tautology.

$C$ is a "prime implicate" of $F$, written

$$C \in \mathcal{P}(F)$$

iff $C$ is an implicate of minimal length.
Formulas as Graphs

\[(a, \neg c, e) \land (\neg b, c, d) \land (\neg c, \neg d, g) \land (b, f, d) \land (\neg a, b)\]
Formulas as Graphs

\[(a, \neg c, e) \land (\neg b, c, d) \land (\neg c, \neg d, g) \land (b, f, d) \land (\neg a, b)\]
Formulas as Graphs

\[(a, \neg c, e) \land (\neg b, c, d) \land (\neg c, \neg d, g) \land (b, f, d) \land (\neg a, b)\]
“Tree-Like” formula

- Their graphs “resemble trees” at a macro level
“Tree-Like” formula

- Their graphs “resemble trees” at a macro level
- The common “closeness” measure to a tree is \texttt{treewidth} (Robertson and Seymour)
“Tree-Like” formula

- Their graphs “resemble trees” at a macro level
- The common “closeness” measure to a tree is treewidth (Robertson and Seymour)
- SAT instances $F$ with fixed treewidth $k$ may be solved in linear time.
Treewidth

- Synonyms for treewidth
  - thin junction-tree or jointree (AI literature)
  - $k$-tree embeddable (graph theory)
  - triangularization, clique number $k$ (Lauritzen and Spiegelhalter)
  - channelwidth $k$ (Hunt and Stearns)
- The channelwidth formalism is constructed for analysis of formulas
- Channelwidth uses objects called structure trees.
Structure Trees

\[ F = \{\{a, b\}, \{\neg a, d, \neg c\}, \{c, \neg e\}, \{\neg c, f\}\} \]
Structure Trees

\[ F = \{\{a, b\}, \{-a, d, \neg c\}, \{c, \neg e\}, \{\neg c, f\}\} \]

\[ S = \langle T, A, B \rangle \]
Structure Trees

\[ F = \{\{a, b\}, \{-a, d, \neg c\}, \{c, \neg e\}, \{\neg c, f\}\} \]

\[ S = \langle T, A, B \rangle \]

\[ T = \langle I, E \rangle \]
Structure Trees

$F = \{ \{a, b\}, \{\neg a, d, \neg c\}, \{c, \neg e\}, \{\neg c, f\}\}$

$S = \langle T, A, B \rangle$

$T = \langle I, E \rangle$

$A : I \rightarrow 2^V$

$\begin{array}{c}
\begin{array}{c}
C_1 \\
C_2 \\
C_3 \\
C_4
\end{array}
\end{array}$

$\begin{array}{c}
\begin{array}{c}
 b \\
 e \\
f
\end{array}
\end{array}$

$\begin{array}{c}
\begin{array}{c}
 a, d \\
c
\end{array}
\end{array}$
Structure Trees

\[ F = \{\{a, b\}, \\{\neg a, d, \neg c\}, \{c, \neg e\}, \{\neg c, f\}\} \]

\[ S = \langle T, A, B \rangle \]

\[ T = \langle I, E \rangle \quad A : I \to 2^V \quad B : I \to 2^F \]

```
    C1
   /|
  /  |
 /    |
C2    C3
       /|
      /  |
     /    |
    C1    C2
       /|
      /  |
     /    |
    C3    C4
       /|
      /  |
     /    |
    C3    C4
       /|
      /  |
     /    |
    C3    C4
       /|
      /  |
     /    |
    C3    C4
```
Channel Variables

- Channel variables $CV(i)$ of a node $i$ are those that both
Channel Variables

- Channel variables $CV(i)$ of a node $i$ are those that both
  - appear at or above that node
Channel Variables

- Channel variables $CV(i)$ of a node $i$ are those that both
  - appear at or above that node
  - appear in a clause at or below that node

The channelwidth of a formula is $\min_{S \in S} \max_{i \in I} |CV(i)|$ where $S$ is all valid structure trees for that formula.

Channelwidth corresponds exactly to the graph theoretic notion of treewidth.
Channel Variables

- Channel variables $CV(i)$ of a node $i$ are those that both
  - appear at or above that node
  - appear in a clause at or below that node
- The channelwidth of a formula is

$$\min_{S \in \mathcal{S}} \max_{i \in I} |CV(i)|$$

where $\mathcal{S}$ is all valid structure trees for that formula
Channel Variables

- Channel variables $CV(i)$ of a node $i$ are those that both
  - appear at or above that node
  - appear in a clause at or below that node
- The channelwidth of a formula is

\[
\min_{S \in \mathcal{S}} \max_{i \in I} |CV(i)|
\]

where $\mathcal{S}$ is all valid structure trees for that formula
- Channelwidth corresponds exactly to the graph theoretic notion of treewidth
Channel Variables

\[ F = \{\{a, b\}, \{\neg a, d, \neg c\}, \{c, \neg e\}, \{\neg c, f\}\} \]

\[ S = \langle T, A, B \rangle \]

\[ T = \langle I, E \rangle \]

\[ A : I \to 2^V \quad B : I \to 2^F \quad CV : I \to 2^V \]

\[
\begin{align*}
T & = \langle I, E \rangle \\
A & : I \to 2^V \\
B & : I \to 2^F \\
CV & : I \to 2^V
\end{align*}
\]
Properties of prime implicates

With \( P_0 = \mathcal{P}(F[0/v]) \) and \( P_1 = \mathcal{P}(F[1/v]) \), we say

\[
S(P_0, P_1, v) = \mathcal{P}(F)
\]

(S runs in \( O(n^4) \) time)
Properties of prime implicates

- With $P_0 = \mathcal{P}(F[0/v])$ and $P_1 = \mathcal{P}(F[1/v])$, we say
  
  $$S(P_0, P_1, v) = \mathcal{P}(F)$$

  ($S$ runs in $O(n^4)$ time)

- If $F$ and $G$ share no variables and are satisfiable,

  $$\mathcal{P}(F) \cup \mathcal{P}(G) = \mathcal{P}(F \land G)$$
Properties of prime implicates

- With $P_0 = \mathcal{P}(F[0/v])$ and $P_1 = \mathcal{P}(F[1/v])$, we say
  \[ S(P_0, P_1, v) = \mathcal{P}(F) \]
  ($S$ runs in $O(n^4)$ time)

- If $F$ and $G$ share no variables and are satisfiable,
  \[ \mathcal{P}(F) \cup \mathcal{P}(G) = \mathcal{P}(F \land G) \]

- $\mathcal{P}(0) = \{\square\}$ and $\mathcal{P}(1) = \{\}$
Let $\Gamma(V)$ be the set of interpretations of the variables $V$.
Tables

Let $\Gamma(V)$ be the set of interpretations of the variables $V$. The table $t$ for $F$ with scope $V$ is a mapping from $\Gamma(V)$ to clause sets such that

$$t(\gamma) = \mathcal{P}(F[\gamma])$$
Let $\Gamma(V)$ be the set of interpretations of the variables $V$.

The table $t$ for $F$ with scope $V$ is a mapping from $\Gamma(V)$ to clause sets such that

$$t(\gamma) = \mathcal{P}(F[\gamma])$$

Tables are determined exactly by their scope and formula.
Let $\Gamma(V)$ be the set of interpretations of the variables $V$. The table $t$ for $F$ with scope $V$ is a mapping from $\Gamma(V)$ to clause sets such that

$$t(\gamma) = \mathcal{P}(F[\gamma])$$

Tables are determined exactly by their scope and formula. Tables “split” the prime implicates of a formula along all interpretations of some of its variables.
Obtaining tables

- For a structure tree $S$ of $F$, let $F_i$ be the formula obtained by collecting all the clauses under node $i$. 

$\quad$
 obtaining tables

- For a structure tree $S$ of $F$, let $F_i$ be the formula obtained by collecting all the clauses under node $i$.
- Let $\Sigma_i$ be the table for $F_i$ with scope $CV(i)$
Obtaining tables

- For a structure tree $S$ of $F$, let $F_i$ be the formula obtained by collecting all the clauses under node $i$.
- Let $\Sigma_i$ be the table for $F_i$ with scope $CV(i)$.
- We call $\Sigma_i$ “$i$’s table” or “the table of $i$”.
Obtaining tables

- If $i$ is a leaf, $\Sigma_i$ can be obtained by the rules

  \[ \mathcal{P}(0) = \{\square\} \text{ and } \mathcal{P}(1) = \{\} \]
Obtaining tables

- If $i$ is a leaf, $\mathcal{T}_i$ can be obtained by the rules
  \[ P(0) = \{\Box\} \text{ and } P(1) = \{\} \]

- If $i$ is an internal node, $\mathcal{T}_i$ can be obtained from the tables of $i$’s children using the rules
  \[ S(P_0, P_1, v) = P(F) \]
  and
  \[ P(F) \cup P(G) = P(F \land G) \]
Obtaining tables

- If $i$ is a leaf, $\mathcal{I}_i$ can be obtained by the rules

  \[ P(0) = \{ \square \} \text{ and } P(1) = \{ \} \]

- If $i$ is an internal node, $\mathcal{I}_i$ can be obtained from the tables of $i$’s children using the rules

  \[ S(P_0, P_1, v) = P(F) \]

  and

  \[ P(F) \cup P(G) = P(F \land G) \]

- Thus the table of any node may be obtained recursively.
Let $r$ be the root of $S$. Note that $F_r = F$.

- Obtain $\mathcal{T}_r$ recursively.
Obtaining $\mathcal{P}(F)$

Let $r$ be the root of $S$. Note that $F_r = F$.

- Obtain $\mathcal{T}_r$ recursively.
- Narrow the scope of $\mathcal{T}_r$ until it is empty, using

$$S(P_0, P_1, v) = \mathcal{P}(F)$$
Obtaining $\mathcal{P}(F)$

- A table $\tau$ with scope $\emptyset$ results
Obtaining $\mathcal{P}(F)$

- A table $\tau$ with scope $\emptyset$ results
- The domain of this table is $\Gamma(\emptyset)$, which contains only the empty assignment $\gamma_\emptyset$
Obtaining \( \mathcal{P}(F) \)

- A table \( \tau \) with scope \( \emptyset \) results
- The domain of this table is \( \Gamma(\emptyset) \), which contains only the empty assignment \( \gamma_\emptyset \)
- \( \tau(\gamma_\emptyset) = \mathcal{P}(Fr[\gamma_\emptyset]) \)
Obtaining $\mathcal{P}(F)$

- A table $r$ with scope $\emptyset$ results
- The domain of this table is $\Gamma(\emptyset)$, which contains only the empty assignment $\gamma_{\emptyset}$
- $r(\gamma_{\emptyset}) = \mathcal{P}(F_r[\gamma_{\emptyset}])$
- $F_r[\gamma_{\emptyset}] = F_r = F$
Obtaining $\mathcal{P}(F)$

- A table $\tau$ with scope $\emptyset$ results
- The domain of this table is $\Gamma(\emptyset)$, which contains only the empty assignment $\gamma_{\emptyset}$
- $\tau(\gamma_{\emptyset}) = \mathcal{P}(F_r[\gamma_{\emptyset}])$
- $F_r[\gamma_{\emptyset}] = F_r = F$
- Thus $\tau(\gamma_{\emptyset}) = \mathcal{P}(F)$
Final words

▷ This is the first width-parameterized algorithm for prime implicate production
Final words

- This is the first width-parameterized algorithm for prime implicate production
- The task is still difficult with small treewidth, but this style of algorithm can prevent much repeated work.
Final words

- This is the first width-parameterized algorithm for prime implicate production
- The task is still difficult with small treewidth, but this style of algorithm can prevent much repeated work.
- Questions? Comments?