“If you have a bottom clause, you have almost solved the problem.”

A Bottom Set Strategy for Tractable Feature Construction

Filip Železný

ČVUT Prague, School of Electrical Engineering, Dept. of Cybernetics
The Gerstner Laboratory for Intelligent Decision Making and Control
Introduction

:: We are concerned with constructing **features**, ie. expressions such as

\[ \text{hasCar}(C), \text{hasLoad}(C,L), \text{isBig}(L) \]

:: The **language of features** \( \mathcal{F}(n) \) is a set where elements comply to constraints:

- **Size.** Each \( f \in \mathcal{F}(n) \) has at most \( n \) atoms.
- **Typing & Moding.** Specify predicates, arguments types and i/o modes.

\[ \text{hasCar}(-\text{car}), \text{hasLoad}(+\text{car},-\text{load}), \]
\[ \text{isBig}(+\text{load}), \text{isSmall}(+\text{load}) \]

- **ETA Constraint.** Each var has both an input and an output occurrence.
  \( \text{(ETA = ‘Extended Transformation Approach’ [Lavrač & Flach, 2001])} \)
The Bottom Set Theorem

:: 2 Problems of size $n$
   - **Existence**: Find an element of $\mathcal{F}(n)$.
   - **Enumeration**: Find all elements of $\mathcal{F}(n)$ (up to var renaming).

:: A bottom set is an expression $\bot(n)$ complying to the typing constraint such that $f \subseteq \bot(n)$ (up to var renaming) whenever $f \in \mathcal{F}(n)$.

:: **Theorem.** Given a $\bot(n)$ such that $|\bot(n)| \leq poly(n)$, one can solve efficiently the existence problem, i.e. find in time $poly(n)$ a feature $f \in \mathcal{F}(n)$ if it exists, or answer NO.

:: **Proof.** Naive approach: must check ETA constraint for $O(|\bot(n)|^n)$ subsets :-(.
   - **Trick**: reduce polynomially onto HORNSAT.
The Essence of the Proof in an Example

:: Bottom set
\[ \perp(3) = \text{hasCar}(C), \text{hasLoad}(C,L), \text{isBig}(L), \text{isSmall}(L) \]
propositional vars:
\[
\begin{array}{cccc}
P_1 & P_2 & P_3 & P_4 \\
\end{array}
\]

:: HORNSAT assignment: \( P_i \) false iff \( i \)-th atom **included** in feature

<table>
<thead>
<tr>
<th>variable</th>
<th>production</th>
<th>consumption</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>( P_2 \leftarrow P_1 )</td>
<td>( P_1 \leftarrow P_2 )</td>
</tr>
<tr>
<td>L</td>
<td>( P_3 \leftarrow P_2 )</td>
<td>( P_2 \leftarrow P_3 \land P_4 )</td>
</tr>
<tr>
<td></td>
<td>( P_4 \leftarrow P_2 )</td>
<td>( P_2 \leftarrow P_3 \land P_4 )</td>
</tr>
<tr>
<td>non-emptiness</td>
<td>( \leftarrow P_1 \land P_2 \land P_3 \land P_4 )</td>
<td></td>
</tr>
</tbody>
</table>

:: \( P_1 = P_2 = P_3 = \text{false} \) is a maximal (fewest false assignments) **solution** (found efficiently \([\text{Dowling} \& \text{Gallier}, 1984]\)), thus \( f = \text{hasCar}(C), \text{hasLoad}(C,L), \text{isBig}(L) \) is a **minimal feature**. Because \(|f| \leq 3\), we output \( f \) (otherwise would say NO).
Tractable Cases: Example

:: A connected **depth-limited** language $\mathcal{F}_D(n) \subset \mathcal{F}(n)$

branching factor $\leq n$

\[
\begin{align*}
\text{hasCar}(C) & \rightarrow \text{hasRoof}(C) \\
& \rightarrow \text{hasLoad}(C, L_1) \rightarrow \text{isBig}(L_1) \\
& \rightarrow \text{hasLoad}(C, L_2) \rightarrow \text{isBig}(L_2) \rightarrow \text{isSmall}(L_2) \\
\leftarrow \text{depth} \leq D & \rightarrow
\end{align*}
\]

:: If literals with multiple inputs, then branching factor $O(n^A)$ ($A \sim \text{max arity}$).

:: Therefore $|\bot(n)| \leq O(n^{A \times D}) = \text{poly}(n)$.

:: Therefore tractable.
Intractable Cases: Example (W-I-P)

:: A “no variable reuse” language $\mathcal{F}_R(n) \subset \mathcal{F}(n)$. Represent typing/moding by a matrix $M$. Find a feature $f \in \mathcal{F}_R(n)$ a represent it by a vector $\vec{f}$. For example

\[
M = \begin{bmatrix}
    \text{hasCar} & \text{hasLoad} & \text{hasRoof} & \text{isBig} & \text{isSmall} \\
    -1 & +1 & +1 & 0 & 0 \\
    0 & -1 & 0 & +1 & +1 \\
    \end{bmatrix}
\]

\[
\begin{array}{c}
\text{car} \\
\text{load}
\end{array}
\]

\[f = \text{hasCar}(C1), \text{hasLoad}(C1,L), \text{isBig}(L), \text{hasCar}(C2), \text{hasRoof}(C2)\]

\[
\vec{f} = [2, 1, 1, 1, 0]^{-1}
\]

\[
\text{hasCar hasLoad hasRoof isBig isSmall}
\]

$\vec{f}$ solves (due to no-var-reuse assumption) the matrix equation

\[
M \times \vec{f} = \vec{0},
\]

I.e. solves an instance of the \textbf{NP-complete integer programming} problem.
Remains to show reduction from an \textbf{arbitrary} integer programming instance.
“If you have a bottom clause, you have almost solved the problem.”

* THE END *