Advanced Solver Technology

Helmut Seidl

TUM + DTU

2006
Part 1

Beyond Finiteness:
Strongly Recognizable Relations
Outline of Part 1:

- The Spi Calculus
- Normalizable Horn Clauses
- Efficient Subclasses
- Instantiation
1. The Spi Calculus

allows formalization of (certain) cryptographic protocols;

generalizes $\pi$ by

- structured terms and
- explicit encryption / decryption.
Example:

\[
\text{send}(c_1, \{\text{pair}(x_1, x_2)\}_k);
\]
\[
\text{recv}(c_2, y);
\]
\[
\text{case } y \text{ of } \{x\}_k; 1
\]
Control-flow Analysis:

- Approximate possible values of variables;
- Approximate possible values sent through channels!

Who knows what?
Natural Formulation of the Analysis:

Horn Clauses

<table>
<thead>
<tr>
<th>terms</th>
<th>program parts</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>values</td>
</tr>
</tbody>
</table>

| predicates | reach/1 |
|            | occurs/1   |
|            | hasValue/2 |
|            | hasMessage/2 |
Decryption:

\[
reach(\text{prog}) \iff
\]

\[
hasValue(X, Y) \iff reach(\text{decrypt}(E, X, K, T)) \land
\]

\[
hasValue(K, V) \land
\]

\[
hasValue(E, \text{enc}(Y, V))
\]
\[ \text{occurs}(E) \iff \text{reach}(\text{decrypt}(E, _, _, _)) \]

\[ \text{reach}(T) \iff \text{reach}(\text{decrypt}(E, _, K, T)) \land \\
\quad \text{hasValue}(K, V) \land \\
\quad \text{hasValue}(E, \text{enc}(_, V)) \]
2. Normalizable Horn Clauses

Desired Properties:

- Relational formulation;
- Possibly infinite least model;
- Effective construction.
Comparison:

Set Constraints: Heintze, Jaffar 1990

\[ B \supseteq a(A, A) \]
\[ C \supseteq B \cap \pi_{b,2}(A) \]
Comparison:

Set Constraints: Heintze, Jaffar 1990

\[ B \supseteq a(A, A) \]
\[ C \supseteq B \cap \pi_{b,2}(A) \]

can be simulated by:

\[ \text{aux}_B(a(X_1, X_2)) \iff \text{aux}_A(X_1) \land \text{aux}_A(X_2) \]
\[ \text{aux}_C(X) \iff \text{aux}_B(X) \land \text{aux}_A(b(\_ , X)) \]
\[-:- \]
Uniform Horn Clauses:

Frühwirth, Shapiro, Vardi, Yardeni 1991

\[
\begin{align*}
  p(a(X_1, \ldots, X_k)) & \iff p_1(X_1) \land \ldots \land p_k(X_k) \\
  p(X) & \iff q_1(t_1) \land \ldots \land q_m(t_m) \\
  p(c) & \iff q_1(t_1) \land \ldots \land q_m(t_m)
\end{align*}
\]
Clauses which are not uniform:

\[ p(a(X, Y)) \iff q(b(X, Y)) \] (relabeling)

\[ p(X, Y) \iff e(X, Z) \land p(Z, Y) \] (composition)

\[ p(X, Y, Z) \iff q(Y, X, Z) \] (permutation)

\[ p(X, Z) \iff q(X, Y, Z) \] (general projection)
The Class $\mathcal{H}l$:  

Chr. Weidenbach 1999

Nielson, Nielson, Seidl 2002

\[
p(a(X_1, \ldots, X_k)) \iff \text{any}
\]

\[
p(X_1, \ldots, X_k) \iff \text{any}
\]
Examples:

\[ p(a(X, Y)) \iff q(b(X, Y)) \] (relabeling)
\[ p(X, Y) \iff e(X, Z) \land p(Z, Y) \] (composition)
\[ p(X, Y, Z) \iff q(Y, X, Z) \] (permutation)
\[ p(X, Z) \iff q(X, Y, Z) \] (general projection)

... all are \( H1 \) :-)

16
Theorem:

Assume $c$ is $\mathcal{H}I$ with least model $\mathcal{H}$. Then

1. $\mathcal{H}(p)$ is strongly recognizable;
2. $\mathcal{H}$ can be computed in deterministic exponential time.
Strongly Recognizable Relation

\[ = \]

finite union of Cartesian products of recognizable tree languages

Closure under:

- Boolean operations;
- Cartesian product;
- transitive closure.
Idea: Normalization

Goal: Automata Form:

\[ p(a(Z_1, \ldots, Z_k)) \iff p_1(X_1) \land \ldots \land p_n(X_n) \]

\[ p(Z_1, \ldots, Z_k) \iff p_1(X_1) \land \ldots \land p_n(X_n) \]

where all \( X_i \) are among \( Z_j \)
Idea: Normalization

Goal: Automata Form:

\[ p(a(Z_1, \ldots, Z_k)) \iff p_1(X_1) \land \ldots \land p_n(X_n) \]
\[ p(Z_1, \ldots, Z_k) \iff p_1(X_1) \land \ldots \land p_n(X_n) \]
where all \( X_i \) are among \( Z_j \)

Technique:

\[ \iff \] Add simpler implied rules until saturation  \( :-) \)
\[ \iff \] All non-automata clauses then are redundant  \( :-) \)
\[ \iff \] All non-automata clauses can be removed  \( :-) \)
Adding Simpler Clauses (0):

Expanding universal predicates:

For:

\[ p(X) \iff \]

add all clauses:

\[ p(a(Z_1, \ldots, Z_k)) \iff p(Z_1) \land \ldots \land p(Z_k) \]

for every constructor \( a \)  :-)}
Adding Simpler Clauses (1):

Resolving Complex Queries:

\[ p(X, Y) \iff q(a(b(X), Z)) \land h(Z) \]
\[ q(a(X_1, X_2)) \iff q_1(X_1) \land q_2(X_2) \]
Adding Simpler Clauses (1):

Resolving Complex Queries:

\[
p(X, Y) \iff q(a(b(X), Z)) \land h(Z)
\]

\[
q(a(X_1, X_2)) \iff q_1(X_1) \land q_2(X_2)
\]
Adding Simpler Clauses (1):

Resolving Complex Queries:

\[ p(X, Y) \iff q(a(b(X), Z)) \land h(Z) \]

\[ q(a(X_1, X_2)) \iff q_1(X_1) \land q_2(X_2) \]

resolves to:

\[ p(X, Y) \iff q_1(b(X)) \land q_2(Z) \land h(Z) \]
Adding Simpler Clauses (1):

Resolving Complex Queries:

\[ p(X, Y) \iff q(a(b(X), Z)) \land h(Z) \]

\[ q(a(X_1, X_2)) \iff q_1(X_1) \land q_2(X_2) \]

resolves to:

\[ p(X, Y) \iff q_1(b(X)) \land q_2(Z) \land h(Z) \]
Adding Simpler Clauses (1):

Resolving Complex Queries:

\[
\begin{align*}
  p(X, Y) & \iff q(a(b(X), Z)) \land h(Z) \\
  q(a(X_1, X_2)) & \iff q_1(X_1) \land q_2(X_2)
\end{align*}
\]

resolves to:

\[
\begin{align*}
  p(X, Y) & \iff q_1(b(X)) \land q_2(Z) \land h(Z)
\end{align*}
\]

- Unification does not introduce new bindings in the first clause.
- The variables in the automata clause are instantiated to sub-terms ...

\[
\implies \text{Every new clause has smaller terms} \quad :)
\]
Adding Simpler Clauses (2):

Resolving with a One-variable Clause:

\[ p(X) \iff q(X) \land h(X) \]
\[ q(a(X_1, X_2)) \iff q_1(X_1) \land q_2(X_2) \]

• The single variable is instantiated with the constructor term of the automata clause.
• New terms in the body are subsequently removed :-)}
Adding Simpler Clauses (2):

Resolving with a One-variable Clause:

\[
p(X) \iff q(X) \land h(X)
\]

\[
q(a(X_1, X_2)) \iff q_1(X_1) \land q_2(X_2)
\]

• The single variable is instantiated with the constructor term of the automata clause.
• New terms in the body are subsequently removed :-)

28
Adding Simpler Clauses (2):

Resolving with a One-variable Clause:

\[ p(X) \iff q(X) \land h(X) \]
\[ q(\ a(X_1, X_2) \ ) \iff q_1(X_1) \land q_2(X_2) \]

resolves to:

\[ p(\ a(X_1, X_2) \ ) \iff q_1(X_1) \land q_2(X_2) \land h(\ a(X_1, X_2) \ ) \]
Adding Simpler Clauses (2):

Resolving with a One-variable Clause:

\[
\begin{align*}
p(X) & \iff q(X) \land h(X) \\
q(a(X_1, X_2)) & \iff q_1(X_1) \land q_2(X_2)
\end{align*}
\]

resolves to:

\[
\begin{align*}
p(a(X_1, X_2)) & \iff q_1(X_1) \land q_2(X_2) \land h(a(X_1, X_2))
\end{align*}
\]
Adding Simpler Clauses (2):

Resolving with a One-variable Clause:

\[
\begin{align*}
  p(X) & \iff q(X) \land h(X) \\
  q(a(X_1, X_2)) & \iff q_1(X_1) \land q_2(X_2)
\end{align*}
\]

resolves to:

\[
\begin{align*}
  p(a(X_1, X_2)) & \iff q_1(X_1) \land q_2(X_2) \land h(a(X_1, X_2))
\end{align*}
\]

- The single variable is instantiated with the constructor term of the automata clause.
- New terms in the body are subsequently removed :-)}
Adding Simpler Clauses (3):

Splitting:

\[ p(X) \iff q(X) \land h_1(Z) \land h_2(Z) \]
Adding Simpler Clauses (3):

Splitting:

\[ p(X) \iff q(X) \land h_1(Z) \land h_2(Z) \]
Adding Simpler Clauses (3):

Splitting:

\[ p(X) \iff q(X) \land h_1(Z) \land h_2(Z) \]

is split into:

\[ p(X) \iff q(X) \land \text{flag}() \]
\[ \text{flag}() \iff h_1(Z) \land h_2(Z) \]
Adding Simpler Clauses (3):

Splitting:

\[ p(X) \Leftarrow q(X) \land h_1(Z) \land h_2(Z) \]

is split into:

\[ p(X) \Leftarrow q(X) \land \text{flag}() \]
\[ \text{flag}() \Leftarrow h_1(Z) \land h_2(Z) \]

If \( h_1(Z) \land h_2(Z) \) is satisfiable, i.e., the intersection of languages accepted by the corresponding automata states is non-empty, we add the clause:

\[ p(X) \Leftarrow q(X) \]
Discussion:

- Normalization results in a set of automata clauses :-)
- The automata representation allows to decide satisfiability of queries such as:

  \[ p(X, a(X, b(Y, c, Y))) \land q(c(X, X)) \]

  ... and also provide candidate substitutions :-)
- \(H1\)-clauses can be used to approximate general Horn clauses ...

36
Example:

\[ p(a(b(X), Y, X)) \Leftrightarrow \text{blabla} \]
Example:

\[ p(a(b(X), Y, X)) \iff \text{blabla} \]
Example:

\[ p(a(b(X), Y, X)) \iff \text{blabla} \]

can be approximated with:

\[ \text{aux}(b(X)) \iff \text{blabla} \]

\[ p(a(Z, Y, X)) \iff \text{aux}(Z) \land \text{blabla} \]
Example:

\[ p(a(b(X), Y, X)) \iff \text{blabla} \]

can be approximated with:

\[ \text{aux}(b(X)) \iff \text{blabla} \]

\[ p(a(Z, Y, X)) \iff \text{aux}(Z) \land \text{blabla} \]

The least model for the approximative set of clauses provides super-sets of tuples for every given predicate...

⇒

If a query is now unsatisfiable, then it was originally unsatisfiable :-)
If it is satisfiable, we know nothing :-(
3. Polynomial Subclasses

The Problem with $\mathcal{H}1$: 

Construction of least model is DEXPTIME-hard.

Frühwirth et al. 1990, Seidl 1994

The Restriction $\mathcal{H}2$: 

Head variables occur in pre-conditions at most once  :-)
Examples:

\[ p(a(X, Y)) \iff q(b(X, Y)) \]  \hspace{1cm} \text{(relabeling)}
\[ p(X, Y) \iff e(X, Z) \land p(Z, Y) \]  \hspace{1cm} \text{(composition)}
\[ p(X, Y, Z) \iff q(Y, X, Z) \]  \hspace{1cm} \text{(permutation)}
\[ p(X, Z) \iff q(X, Y, Z) \]  \hspace{1cm} \text{(general projection)}

... all are \( H2 \) :-))
Theorem:

Let $c$ an $\mathcal{H}2$-clause where $k$ is the max. number of variable occurrences in implications.

Then the least model of $c$ can be computed in time

$$|c|^{\mathcal{O}(k)}$$
Corollary:

The transitive closure of a strongly recognizable binary tree relation can be computed in **polynomial time**.

Proof:

\[
\begin{align*}
\text{trans}(X, Y) & \iff \text{edge}(X, Y) \\
\text{trans}(X, Z) & \iff \text{edge}(X, Y) \land \text{trans}(Y, Z)
\end{align*}
\]
The Problem with $\mathcal{H}2$:

- The exponent can be large :-(
- Not all implications of Spi analysis satisfy $\mathcal{H}2$ :-((
Example: Expression Evaluation

\[
\text{occurs}(E_1) \iff \text{occurs}(\text{pair}(E_1, E_2))
\]

\[
\text{occurs}(E_2) \iff \text{occurs}(\text{pair}(E_1, E_2))
\]

\[
\text{hasValue}(\text{pair}(E_1, E_2), \text{pair}(V_1, V_2)) \iff \text{occurs}(\text{pair}(E_1, E_2)) \land \text{hasValue}(E_1, V_1) \land \text{hasValue}(E_2, V_2)
\]

... is not even $\mathcal{H}1$ :-(

46
Forgetting about Expressivity: $\mathcal{H}3$

All variable dependences are acyclic.

Example:

$$p(a(X, Y)) \iff q_1(d(Z, c(X))) \land q_2(c(Y)) \land q_3(Z) \land s(a(b, b))$$
\[ p(a(X, Y)) \]

- **node**: literal
- **edge**: common variable

Diagram:
- **Node**: \( p(a(X, Y)) \)
- **Edges**:
  - \( X \) to \( q_1(d(Z, c(X))) \)
  - \( Y \) to \( q_2(c(Y)) \)
  - \( Z \) to \( q_3(Z) \)
  - \( s(a(b, b)) \)
Idea:

- Apply normalization as in the general case  :-)
- Introduce auxiliary predicates to break very complex right-hand sides into simple ones before-hand ...

A Simple Clause:

$$ \text{get}_1(X_1) \iff q_1(d(Z, X_1)) \land q_3(Z) $$
Resolving Simple Clauses:

\[
\begin{align*}
\text{get}_1(X_1) & \iff q_1(d(Z, X_1)) \land q_3(Z) \\
q_1(d(Z, X_1)) & \iff h_1(Z) \land h_2(X_1)
\end{align*}
\]
Resolving Simple Clauses:

\[ \text{get}_1(X_1) \iff q_1(d(Z, X_1)) \land q_3(Z) \]
\[ q_1(d(Z, X_1)) \iff h_1(Z) \land h_2(X_1) \]
Resolving Simple Clauses:

\[
\text{get}_1(X_1) \iff q_1(d(Z, X_1)) \land q_3(Z)
\]

\[
q_1(d(Z, X_1)) \iff h_1(Z) \land h_2(X_1)
\]

gives:

\[
\text{get}_1(X_1) \iff h_2(X_1) \land h_2(Z) \land q_3(Z)
\]
Resolving Simple Clauses:

\[
\begin{align*}
\text{get}_1(X_1) & \iff q_1(d(Z, X_1)) \land q_3(Z) \\
q_1(d(Z, X_1)) & \iff h_1(Z) \land h_2(X_1)
\end{align*}
\]

gives:

\[
\begin{align*}
\text{get}_1(X_1) & \iff h_2(X_1) \land h_2(Z) \land q_3(Z)
\end{align*}
\]

⇒ Resolving simple clauses introduces only binary intersections  :-)
⇒ After splitting these intersections, very simple propagation rules remain  :-))
Therefore, we translate:

\[ p(a(X, Y)) \iff q_1(d(Z, c(X))) \land q_2(c(Y)) \land q_3(Z) \]

into:

\[ p(a(X, Y)) \iff \text{aux}_X(X) \land \text{aux}_Y(Y) \]
Therefore, we translate:

\[ p(a(X, Y)) \iff q_1(d(Z, c(X))) \land q_2(c(Y)) \land q_3(Z) \]

into:

\[ p(a(X, Y)) \iff \text{aux}_X(X) \land \text{aux}_Y(Y) \]
Therefore, we translate:

\[ p(a(X, Y)) \iff q_1(d(Z, c(X))) \land q_2(c(Y)) \land q_3(Z) \]

into:

\[ p(a(X, Y)) \iff aux_X(X) \land aux_Y(Y) \]

\[ aux_Y(Y) \iff q_2(c(Y)) \]
Therefore, we translate:

\[ p(a(X, Y)) \iff q_1(d(Z, c(X))) \land q_2(c(Y)) \land q_3(Z) \]

into:

\[ p(a(X, Y)) \iff aux_X(X) \land aux_Y(Y) \]

\[ aux_Y(Y) \iff q_2(c(Y)) \]
Therefore, we translate:

\[ p(a(X, Y)) \iff q_1(d(Z, c(X))) \land q_2(c(Y)) \land q_3(Z) \]

into:

\[ p(a(X, Y)) \iff aux_X(X) \land aux_Y(Y) \]

\[ aux_X(X) \iff get_1(c(X)) \]

\[ aux_Y(Y) \iff q_2(c(Y)) \]
Therefore, we translate:

\[ p(a(X, Y)) \Leftarrow q_1(d(Z, c(X))) \land q_2(c(Y)) \land q_3(Z) \]

into:

\[
\begin{align*}
p(a(X, Y)) & \Leftarrow \text{aux}_X(X) \land \text{aux}_Y(Y) \\
\text{aux}_X(X) & \Leftarrow \text{get}_1(c(X)) \\
\text{aux}_Y(Y) & \Leftarrow q_2(c(Y))
\end{align*}
\]
Therefore, we translate:

\[ p(a(X, Y)) \iff q_1(d(Z, c(X))) \land q_2(c(Y)) \land q_3(Z) \]

into:

\[ p(a(X, Y)) \iff aux_X(X) \land aux_Y(Y) \]
\[ aux_X(X) \iff get_1(c(X)) \]
\[ get_1(X_1) \iff q_1(d(Z, X_1)) \land q_3(Z) \]
\[ aux_Y(Y) \iff q_2(c(Y)) \]
Note:

In expressiveness, $\mathcal{H}_3$ clauses correspond to set constraints without intersections – enhanced with:

$$X \supseteq \pi_{a,i}(a(B_1, \ldots, B_{i-1}, \top, B_{i+1}, \ldots, B_k) \cap Y)$$

// conditional projection
Note:

In expressiveness, $\mathcal{H}3$ clauses correspond to set constraints without intersections – enhanced with:

$$X \supseteq \pi_{a,i}(a(B_1, \ldots, B_{i-1}, \top, B_{i+1}, \ldots, B_k) \cap Y)$$

// conditional projection

Theorem:

The least model of an $\mathcal{H}3$-clause $c$ can be computed in time:

$$O(|c|^3)$$
Note:

In expressiveness, $\mathcal{H}3$ clauses correspond to set constraints without intersections – enhanced with:

$$X \supseteq \pi_{a,i}(a(B_1, \ldots, B_{i-1}, \top, B_{i+1}, \ldots, B_k) \cap Y)$$

// conditional projection

Theorem:

The least model of an $\mathcal{H}3$-clause $c$ can be computed in time:

$$O(|c|^3)$$

... so what?
4. Instantiation

Expression Evaluation, revisited:

\[ \text{occurs}(E_1) \iff \text{occurs}(\text{pair}(E_1, E_2)) \]

\[ \text{occurs}(E_2) \iff \text{occurs}(\text{pair}(E_1, E_2)) \]

\[ \text{hasValue}(\text{pair}(E_1, E_2) \land \text{pair}(V_1, V_2)) \iff \text{occurs}(\text{pair}(E_1, E_2)) \land \]

\[ \text{hasValue}(E_1, V_1) \land \]

\[ \text{hasValue}(E_2, V_2) \]

Observation:

\[ \text{occurs}/1 \text{ holds only for ground sub-terms of the input } \quad :-) \]
Instantiation of all Possible Values:

\[
\begin{align*}
\text{occurs}(e_1) & \iff \text{occurs}(\text{pair}(e_1, e_2)) \\
\text{occurs}(e_2) & \iff \text{occurs}(\text{pair}(e_1, e_2)) \\
\text{hasValue}(\text{pair}(e_1, e_2), \text{pair}(V_1, V_2)) & \iff \text{occurs}(\text{pair}(e_1, e_2)) \land \\
& \quad \land \text{hasValue}(e_1, V_1) \land \\
& \quad \land \text{hasValue}(e_2, V_2)
\end{align*}
\]

... now is \( \mathcal{H}3 \ :-)) \)
In General for Spi:

- Both `reach/1` and `occurs/1` hold for ground sub-terms of `prog` only.
- Why not instantiating the corresponding variables in all ways?
- Removal of these variables results in:

  ⇒ a clause of size $O(|prog|)$
  ⇒ ... which is $H3$. 

... we conclude:
Theorem:

Control-flow analysis for Spi is cubic.
Discussion:

- Mixing bounded components in predicates with unbounded ones, is convenient for practical applications :-)
- Manually performing instantiation is boring and error-prone ...
- Analyses are required for:
  1. determining the bounded components of predicates;
  2. determining the finitely many values for these components :-)
- The latter can be done with Datalog ;-)