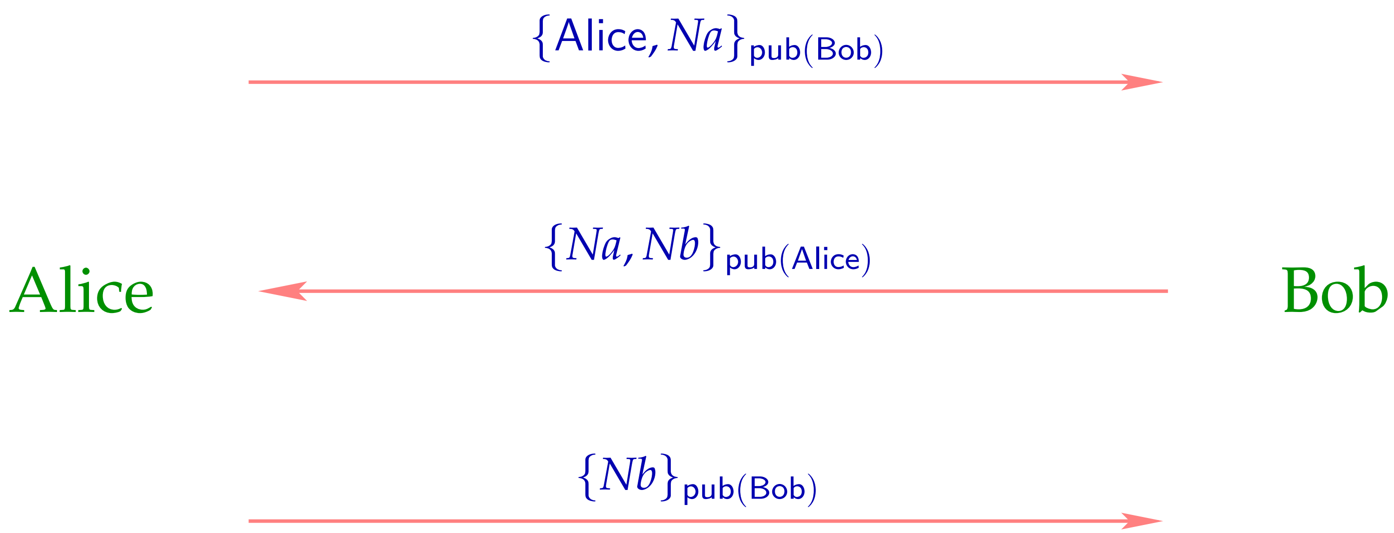
Perspective: Normal Horn Clauses

- Prolog may no longer be the sexiest programming language :-)
- Horn clauses, though, are very well suited for the specification of analysis problems.
- It is a separate problem then to solve the stated analysis problem :-)
- If the least solution cannot be computed exactly, approximate solutions may at least yield approximative answers ...

Example: Cryptographic Protocols

Rules for the Exchange of Messages:



Properties to be verified:

secrecy, authenticity, ...

The Dolev-Yao Model:

- Messages are terms:

	Representation
$\{m\}_k$	$\texttt{encrypt}(m, k)$
$\langle m_1, m_2 \rangle$	$\texttt{pair}(m_1, m_2)$

$\Longrightarrow$ Distinct terms represent distinct messages :-)

$\Longrightarrow$ perfect cryptography. Therefore, we have:

$\{m\}_k = \{m'\}_{k'}$ iff $m = m'$ and $k = k'$

- The attacker has full control over the network:

  All messages are exchanged with the attacker.

Example: The Needham-Schroeder Protocol

1. $A \longrightarrow B : \{a, n_a\}_{k_b}$
2. $B \longrightarrow A : \{n_a, n_b\}_{k_a}$
3. $A \longrightarrow B : \{n_b\}_{k_b}$

Abstraction:

- Unbounded number of sessions !!
- Nonces sind not necessarily fresh ??

Idea:

Characterize the knowledge of the attacker by means of Horn clauses ...

1.	$A \longrightarrow B : \{a, n_a\}_{k_b}$	$\text{known}(\{a, n_a\}_{k_b}) \leftarrow$
2.	$B \longrightarrow A : \{n_a, n_b\}_{k_a}$	$\text{known}(\{X, n_b\}_{k_a}) \leftarrow \text{known}(\{a, X\}_{k_b})$
3.	$A \longrightarrow B : \{n_b\}_{k_b}$	$\text{known}(\{X\}_{k_b}) \leftarrow \text{known}(\{n_a, X\}_{k_a})$

Secrecy of N_b : $\quad \leftarrow \text{known}(n_b).$

Discussion:

- We have abstracted all nonces with finitely many.
- Less restrictive (though still correct) abstractions are still possible ...

1. $A \longrightarrow B : \{a, n_a\}_{k_b}$ $\quad \ldots$
2. $B \longrightarrow A : \{n_a, n_b\}_{k_a}$ $\quad \mathsf{known}(\{X, n_b(X)\}_{k_a}) \leftarrow \mathsf{known}(\{a, X\}_{k_b})$
3. $A \longrightarrow B : \{n_b\}_{k_b}$ $\quad \ldots$

The fresh nonce is a function of the received nonce :-)

Blanchet 2001

Further capabilities of the attacker:

$$
\begin{array}{lcl}
\mathsf{known}(\{X\}_Y) & \leftarrow & \mathsf{known}(X), \mathsf{known}(Y) \\
 & & // \quad \text{The attacker can encode} \\
\mathsf{known}(\langle X, Y \rangle) & \leftarrow & \mathsf{known}(X), \mathsf{known}(Y) \\
 & & // \quad \text{The attacker can construct pairs} \\
\mathsf{known}(X) & \leftarrow & \mathsf{known}(\{X\}_Y), \mathsf{known}(Y) \\
 & & // \quad \text{The attacker can decode} \\
\mathsf{known}(X) & \leftarrow & \mathsf{known}(\langle X, Y \rangle) \\
\mathsf{known}(Y) & \leftarrow & \mathsf{known}(\langle X, Y \rangle) \\
 & & // \quad \text{The attacker can project}
\end{array}
$$

Discussion

- Type inference for Prolog computed a regular abstraction of the set of paths of the denotational semantics.
- Sometimes, this is too imprecise :-(
- Instead, we now approximate the denotational semantics directly :-)

- This, however, can be quite expensive

  $\Longrightarrow$ not well suited for compilers :-(

  $\Longrightarrow$ in general, much more precise :-)

Simplification:

We only consider clauses whose heads are of the form:

$$p(f(X_1, \ldots, X_k)) \qquad \text{or} \qquad p(b) \qquad \text{or} \qquad p(X_1, \ldots, X_k)$$

Such clauses are called H1.

Theorem

- Every finite set of H1-clauses is equivalent to a finite set of simple H1-clauses of the form:

$$\begin{array}{lll} p(f(X_1, \ldots, X_k)) & \leftarrow & p_1(X_{i_1}), \ldots, p_r(X_{i_1}) \\ p(X_1, \ldots, X_k) & \leftarrow & p_1(X_{i_1}), \ldots, p_r(X_{i_1}) \\ p(b) & \leftarrow & \end{array}$$

- ... or even to a finite set of normal H1-clauses.

Idea:

We successively introduce simper clauses until the complicated ones become superfluous ...

Rule 1: Splitting

We separate independent parts from the pre-conditions:

$$head \leftarrow rest,\ p_1(X), \ldots, p_m(X)$$

$$(X \text{ does not occur in } head, rest\ )$$

is replaced with:

$$\begin{array}{lll} head & \leftarrow & rest, q() \\ q() & \leftarrow & p_1(X), \ldots, p_m(X) \end{array}$$

for a new predicate $q/0$.

Rule 2: Simplification

We introduce simpler derived clauses:

$$\begin{array}{lcl} head & \leftarrow & p(f(t_1, \ldots, t_k)), rest \\ p(f(X_1, \ldots, X_k)) & \leftarrow & p_1(X_{i_1}), \ldots, p_r(X_{i_r}) \end{array}$$

implies:

$$\begin{array}{lcl} head & \leftarrow & p_1(t_{i_1}), \ldots, p_r(t_{i_r}), rest \end{array}$$

$$\begin{array}{lcl} head & \leftarrow & p(t_1, \ldots, t_k), rest \\ p(X_1, \ldots, X_k) & \leftarrow & p_1(X_{i_1}), \ldots, p_r(X_{i_r}) \end{array}$$

implies:

$$\begin{array}{lcl} head & \leftarrow & p_1(t_{i_1}), \ldots, p_r(t_{i_r}), rest \end{array}$$

Rule 3 (Cont.): Simplification

$$p(X) \leftarrow p_1(X), \ldots, p_m(X)$$

$$p_i(f(X_1, \ldots, X_k)) \leftarrow p_{i1}(X_{i1}), \ldots, p_{ir_i}(X_{ir_i})$$

implies:

$$p(f(X_1, \ldots, X_k))) \leftarrow p_{11}(X_{11}), \ldots, p_{mr_m}(X_{mr_m})$$

$$\mathit{head} \leftarrow p(b), \mathit{rest}$$

$$p(b) \leftarrow$$

implies:

$$\mathit{head} \leftarrow \mathit{rest}$$

Rule 4: Guard Simplification

$$\begin{array}{lcl} p() & \leftarrow & p_1(X), \ldots, p_m(X) \\ p_i(f(X_1, \ldots, X_k)) & \leftarrow & p_{i1}(X_{i1}), \ldots, p_{ir_i}(X_{ir_i}) \\ & & \quad \text{implies:} \\ p() & \leftarrow & p_{11}(X_{11}), \ldots, p_{mr_m}(X_{mr_m}) \end{array}$$

$$\begin{array}{lcl} p() & \leftarrow & p_1(X), \ldots, p_m(X) \\ p_i(b) & \leftarrow & \quad \text{implies:} \\ p() & \leftarrow & \end{array}$$

Theorem

Assume that $\mathcal{C}$ is finite set of clauses which is closed under splitting and simplification and guard simplification.

Let $\mathcal{C}_0 \subseteq \mathcal{C}$ denote the subset of simple clauses of $\mathcal{C}$. Then for all occurring predicates p,

$$[\![p]\!]_{\mathcal{C}_0} = [\![p]\!]_{\mathcal{C}}$$

Proof:

Induction on the depth of terms in tuples of $[\![p]\!]_{\mathcal{C}}$:-)

Transformation into normal clauses:

Introduce fresh predicates for conjunctions of unary predicates.

Assume $A = \{p_1, \ldots, p_m\}$. Then:

$$
\begin{array}{lcl}
[A](b) & \leftarrow & \text{whenever} \quad p_i(b) \leftarrow \quad \text{for all } i. \\
[A](f(X_1, \ldots, X_k)) & \leftarrow & [B_1](X_1), \ldots, [B_k](X_k) \\
& & \text{whenever} \quad B_i = \{p_{jl} \mid X_{i_{jl}} = X_i\} \quad \text{for} \\
& & p_j(f(X_1, \ldots, X_k)) \leftarrow p_{j1}(X_{i_{j1}}), \ldots, p_{jr_j}(X_{i_{jr_j}})
\end{array}
$$

Warning:

- The emptiness problem for Horn clauses in H1 is DEXPTIME-complete !
- In many cases, our method still terminates quickly ;-)

- Not all Horn clauses are in H1 :-(

  $\Longrightarrow$ an approximation technique is required ...

Approximation of Horn Clauses

Step 1:

Simplification of pre-conditions by splitting, simplification and guard simplification (as before :-)

Step 2:

Introduction of copies of variables X. Every copy receives all literals of X as pre-condition.

$$p(f(X,X)) \quad \leftarrow \quad q(X) \qquad \text{yields :}$$

$$p(f(X,X')) \quad \leftarrow \quad q(X), q(X')$$

Step 3:

Introduction of an auxiliary predicate for every non-variable subterm of the head.

$$p(f(g(X,Y),Z)) \quad \leftarrow \quad q_1(X), q_2(Y), q_3(Z) \qquad \text{yields :}$$

$$\begin{aligned} p_1(g(X,Y)) &\quad \leftarrow \quad q_1(X), q_2(Y), q_3(Z) \\ p(f(H,Z)) &\quad \leftarrow \quad p_1(H), q_1(X), q_2(Y), q_3(Z) \end{aligned}$$