

From set constraints to Horn clauses

Let \mathcal{C} have the single constraint

$$\mathcal{X} \supseteq a \cup f(f(\mathcal{X})) \cup (f_1^{-1}(\mathcal{X}) \cap f(\mathcal{X}))$$

Correspondingly we create clauses:

$$\begin{array}{lll} q_1(a) & q(x) \Leftarrow q_1(x) & q_2(f(x)) \Leftarrow q(x) \\ q_3(f(x)) \Leftarrow q_2(x) & q(x) \Leftarrow q_3(x) & q_4(x) \Leftarrow q(f(x)) \\ q_5(x) \Leftarrow q_4(x) \wedge q_2(x) & q(x) \Leftarrow q_5(x) & \end{array}$$

If I is the least solution of \mathcal{C} then

$t \in I(\mathcal{X})$ iff t is accepted at q .

In general for every set expression se occurring in \mathcal{C} , create state q_{se} .

Add clauses

$$q_{\top}(f(x_1, \dots, x_n)) \Leftarrow q_{\top}(x_1) \wedge \dots \wedge q_{\top}(x_n) \text{ for every } f.$$

$$q_{f(se_1, \dots, se_n)}(f(x_1, \dots, x_n)) \Leftarrow q_{se_1}(x_1) \wedge \dots \wedge q_{se_n}(x_n)$$

$$q_{f_i^{-1}(se)}(x_i) \Leftarrow q_{se}(f(x_1, \dots, x_n))$$

$$q_{se_1 \cup se_2}(x) \Leftarrow q_{se_1}(x)$$

$$q_{se_1 \cup se_2}(x) \Leftarrow q_{se_2}(x)$$

$$q_{se_1 \cap se_2}(x) \Leftarrow q_{se_1}(x) \wedge q_{se_2}(x)$$

Also for every constraint $\mathcal{X} \supseteq se$ in \mathcal{C} create the clause $q_{\mathcal{X}}(x) \Leftarrow q_{se}(x)$.

If I is the least solution of \mathcal{C} then

$t \in I(\mathcal{X})$ iff t is accepted at $q_{\mathcal{X}}$.

Eliminating alternation and two-wayness

Idea 1: To eliminate alternation, create new states which represent intersections of certain number of old states.

Idea 2: To eliminate two-wayness **short cut** pop and push clauses.

E.g. from clauses $q_1(x) \Leftarrow q_2(f(x))$ and $q_2(f(x)) \Leftarrow q_3(x)$, deduce the new ϵ clause $q_1(x) \Leftarrow q_3(x)$.

Let \mathcal{A} be an alternating two-way automaton on set of states Q . We denote by \mathcal{A}_{simple} the set of pop and ϵ clauses in \mathcal{A} at any point of our procedure.

Create fresh states q_S for each set $S \subseteq Q$. We rename the old states P by $q_{\{P\}}$. Alternation clauses $P(x) \Leftarrow P_1(x) \wedge P_2(x)$ become $q_{\{P\}}(x) \Leftarrow q_{\{P_1, P_2\}}(x)$.

If clauses

$$q_{\{P^1\}}(f(x_1, \dots, x_n)) \Leftarrow q_{S_1^1}(x_1) \wedge \dots \wedge q_{S_n^1}(x_n)$$

...

$$q_{\{P^k\}}(f(x_1, \dots, x_n)) \Leftarrow q_{S_1^k}(x_1) \wedge \dots \wedge q_{S_n^k}(x_n)$$

are already in \mathcal{A} , then we add to \mathcal{A} the clause

$$q_{\{P^1, \dots, P^k\}}(f(x_1, \dots, x_n)) \Leftarrow q_{S_1^1 \cup \dots \cup S_1^k}(x_1) \wedge \dots \wedge q_{S_n^1 \cup \dots \cup S_n^k}(x_n)$$

If clauses

$$q_{\{P\}}(x) \Leftarrow q_S(x)$$

and

$$q_S(f(x_1, \dots, x_n)) \Leftarrow q_{S_1}(x_1) \wedge \dots \wedge q_{S_n}(x_n)$$

are already in \mathcal{A}

then we add to \mathcal{A} the clause

$$q_{\{P\}}(f(x_1, \dots, x_n)) \Leftarrow q_{S_1}(x_1) \wedge \dots \wedge q_{S_n}(x_n)$$

If clauses

$$q_{\{P\}}(x_i) \Leftarrow q_{\{P'\}}(f(x_1, \dots, x_n))$$

and

$$q_{\{P'\}}(f(x_1, \dots, x_n)) \Leftarrow q_{S_1}(x_1) \wedge \dots \wedge q_{S_n}(x_n)$$

are already in \mathcal{A} , and

for $j \neq i$, the q_{S_j} accepts at least one term in \mathcal{A}_{simple} , then we add to \mathcal{A} the clause

$$q_{\{P\}}(x_i) \Leftarrow q_{S_i}(x_i)$$

We keep adding new clauses until no new clauses can be added. Then we remove all push clauses to get a one-way automaton.

This can be done in EXPTIME.

⇒ Alternating two-way tree automata have the same expressiveness as ordinary tree automata.

⇒ The least model of our class of set constraints can be represented using tree automata.

Also more general push clauses can be represented using the previous clauses.

The clause

$$P_1(x) \Leftarrow P_2(f(x)) \wedge P_3(x) \wedge P_4(x)$$

is represented using the clauses

$$P_5(x) \Leftarrow P_3(x) \wedge P_4(x)$$

$$P_6(f(x)) \Leftarrow P_5(x)$$

$$P_7(x) \Leftarrow P_2(x) \wedge P_6(x)$$

$$P_1(x) \Leftarrow P_7(f(x))$$

Similarly any clause of the form

$$P(x_i) \Leftarrow Q(f(x_1, \dots, x_n)) \wedge P_1(x_{i_1}) \wedge \dots \wedge P_k(x_{i_k})$$

$$(1 \leq i, i_1, \dots, i_k \leq n)$$

can be translated to the previous pop, push and alternation clauses.

Application to verification of cryptographic protocols

Cryptographic protocols are used for secure communication on an insecure network. E.g. protocols used in electronic commerce.

Use cryptographic algorithms like encryption, decryption, also random number generators, hash functions, . . .

We are interested in verifying security properties, notably **secrecy** properties stating that certain information is not known to any third party.

We are interested in detecting **logical flaws** in the protocols, hence we assume that the **cryptographic algorithms** are **perfect**.

E.g. if $\{m\}_k$ is the encryption of message m with key k , then m cannot be deduced from it without knowing the key k .

Private key cryptography:

A and B agree on a common key K_{ab} . Messages encoded with K_{ab} can only be decrypted with K_{ab} .

Public key cryptography:

A chooses a pair of keys K_a (public key) and K_a^{-1} (private key). Messages encrypted with K_a can only be decrypted K_a^{-1} , and vice-versa.

Nonces: these are random numbers generated by participants during protocol sessions.

Cryptographic protocol = rules for exchanging messages

1. $A \longrightarrow B : M_1$
2. $B \longrightarrow C : M_2$
3. $C \longrightarrow B : M_3$
4. $B \longrightarrow A : M_4$
- ...

The Public-key Needham-Schroeder protocol

1. $A \longrightarrow B : \{A, N_a\}_{K_b}$
2. $B \longrightarrow A : \{N_a, N_b\}_{K_a}$
3. $A \longrightarrow B : \{N_b\}_{K_b}$

After the protocol session, A and B think that they are talking to each other, and N_a and N_b are not known to any other person.

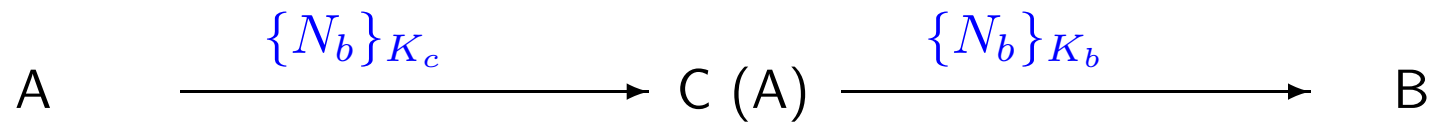
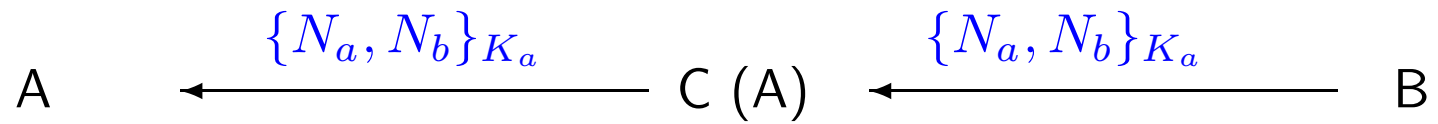
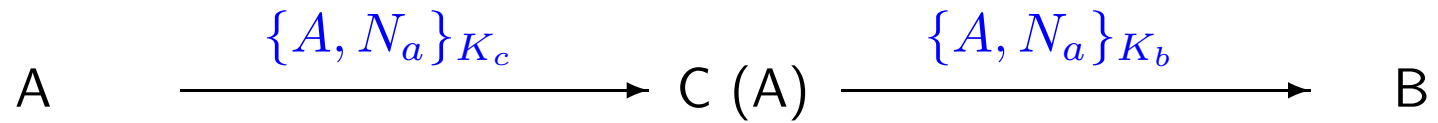
The Public-key Needham-Schroeder protocol

1. $A \longrightarrow B : \{A, N_a\}_{K_b}$
2. $B \longrightarrow A : \{N_a, N_b\}_{K_a}$
3. $A \longrightarrow B : \{N_b\}_{K_b}$

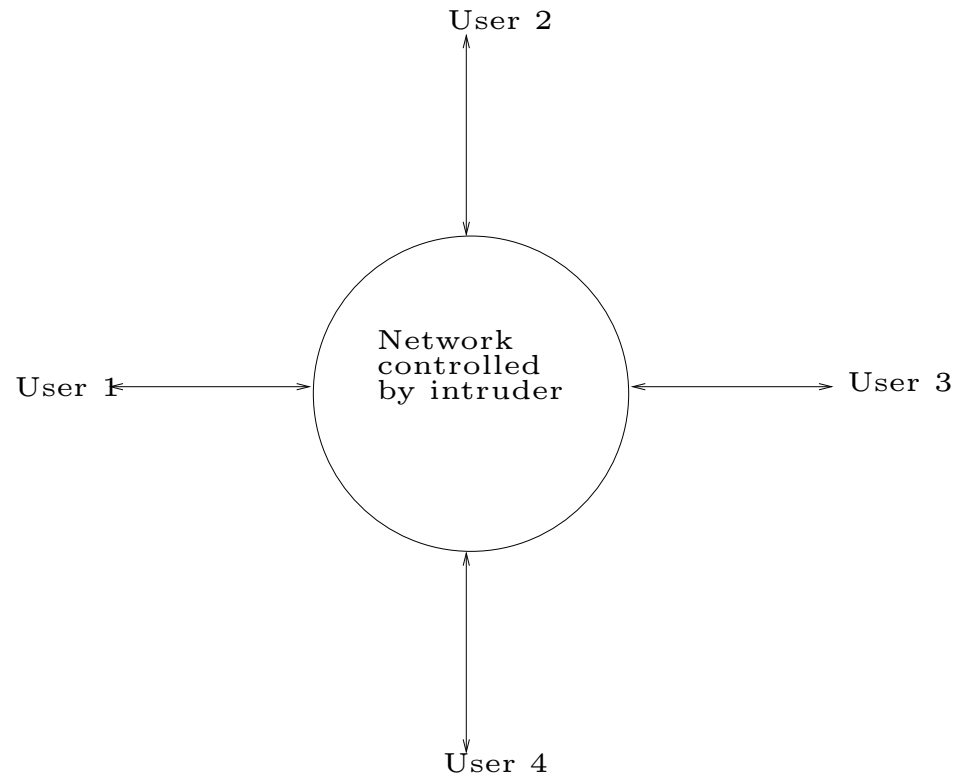
After the protocol session, A and B think that they are talking to each other, and N_a and N_b are not known to any other person.

Attack against the protocol found 17 years after its publication!

Man in the middle attack:



The Dolev-Yao model



Intruder can send messages, read messages, delete messages, generate nonces, encrypt and decrypt messages using known keys.
Intruder has unlimited memory to remember messages.

Assumption of perfect cryptography

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

$$\{\dots \{m\}_k \dots\}_k \neq m$$

$$\langle m_1, m_2 \rangle = \langle m'_1, m'_2 \rangle \text{ iff } m_1 = m'_1 \text{ and } m_2 = m'_2$$

....

Formally we model messages as terms.

$\{m\}_k$ is represented as *enc*(m, k)

$\langle m_1, m_2 \rangle$ is represented by the term *pair*(m_1, m_2)

To solve the secrecy problem our approach is now to represent the knowledge of the intruder as the language accepted by a tree automaton.

Checking whether certain kinds of messages are not known to the intruder reduces to membership or intersection-emptiness problem of automata.

We over-approximate the the knowledge of the intruder.

Useful for certifying protocols. May introduce false attacks.

Use state I to accept all terms known by the intruder.

The public key Needham-Schroeder example:

1. $A \longrightarrow B : \{A, N_a\}_{K_b}$
2. $B \longrightarrow A : \{N_a, N_b\}_{K_a}$
3. $A \longrightarrow B : \{N_b\}_{K_b}$

Fix honest agents A, B and one dishonest agent C .

Choose a finite set of nonces $n_{xy}^1, n_{xy}^2, n_{yx}^1, n_{yx}^2$ for distinct agents x and y .

1. $A \longrightarrow B : \{A, N_a\}_{K_b}$
2. $B \longrightarrow A : \{N_a, N_b\}_{K_a}$
3. $A \longrightarrow B : \{N_b\}_{K_b}$

$$I(\{A, n_{ab}^1\}_{K_b})$$

$$I(\{A, n_{ac}^1\}_{K_c})$$

$$I(\{B, n_{ba}^1\}_{K_a})$$

$$I(\{B, n_{bc}^1\}_{K_c})$$

$$I(\{C, n_{ca}^1\}_{K_a})$$

$$I(\{C, n_{cb}^1\}_{K_b})$$

These can be written as clauses of alternating two-way tree automata:

The clause $I(\{A, n_{ab}^1\}_{K_b})$ is written as

$$q_1(A)$$

$$q_2(n_{ab}^1)$$

$$q_3(K_b)$$

$$q_4(\langle x, y \rangle) \Leftarrow q_1(x) \wedge q_2(y)$$

$$I(\{x\}_y) \Leftarrow q_4(x) \wedge q_3(y)$$

for fresh states q_1, q_2, q_3, q_4 .

The above are all pop clauses.

1. $A \longrightarrow B : \{A, N_a\}_{K_b}$
2. $B \longrightarrow A : \{N_a, N_b\}_{K_a}$
3. $A \longrightarrow B : \{N_b\}_{K_b}$

$$I(\{x, n_{ab}^2\}_{K_a}) \Leftarrow I(\{A, x\}_{K_b})$$

$$I(\{x, n_{ac}^2\}_{K_a}) \Leftarrow I(\{A, x\}_{K_c})$$

$$I(\{x, n_{ba}^2\}_{K_b}) \Leftarrow I(\{B, x\}_{K_a})$$

$$I(\{x, n_{bc}^2\}_{K_b}) \Leftarrow I(\{B, x\}_{K_c})$$

$$I(\{x, n_{ca}^2\}_{K_c}) \Leftarrow I(\{C, x\}_{K_a})$$

$$I(\{x, n_{cb}^2\}_{K_c}) \Leftarrow I(\{C, x\}_{K_b})$$

The clause $I(\{x, n_{ab}^2\}_{K_a}) \Leftarrow I(\{A, x\}_{K_b})$ can be written as:

$$\begin{aligned} q_1(K_b) & \text{ (pop)} \\ q_2(y) & \Leftarrow I(\{y\}_z) \wedge q_1(z) \text{ (push)} \\ q_3(A) & \text{ (pop)} \\ q_4(x) & \Leftarrow q_2(\langle y', x \rangle) \wedge q_3(y') \text{ (push)} \\ q_5(n_{ab}^2) & \text{ (pop)} \\ q_6(\langle x, x' \rangle) & \Leftarrow q_4(x) \wedge q_5(x') \text{ (pop)} \\ q_7(K_a) & \text{ (pop)} \\ I(\{x''\}_{x'''}) & \Leftarrow q_6(x'') \wedge q_7(x''') \text{ (pop)} \end{aligned}$$

1. $A \longrightarrow B : \{A, N_a\}_{K_b}$
2. $B \longrightarrow A : \{N_a, N_b\}_{K_a}$
3. $A \longrightarrow B : \{N_b\}_{K_b}$

$$I(\{x\}_{K_b}) \Leftarrow I(\{n_{ab}^1, x\}_{K_a})$$

$$I(\{x\}_{K_c}) \Leftarrow I(\{n_{ac}^1, x\}_{K_a})$$

$$I(\{x\}_{K_c}) \Leftarrow I(\{n_{ba}^1, x\}_{K_b})$$

$$I(\{x\}_{K_a}) \Leftarrow I(\{n_{bc}^1, x\}_{K_b})$$

$$I(\{x\}_{K_a}) \Leftarrow I(\{n_{ca}^1, x\}_{K_c})$$

$$I(\{x\}_{K_b}) \Leftarrow I(\{n_{cb}^1, x\}_{K_c})$$

The intruder's initial knowledge:

$$\begin{array}{cccccc} I(n_{ca}^1) & I(n_{cb}^1) & I(n_{ac}^2) & I(n_{bc}^2) & I(K_a) & I(K_b) \\ I(K_c) & I(K_c^{-1}) & I(A) & I(B) & I(C) & \end{array}$$

The intruder's deductive abilities:

$$\begin{aligned} I(\{x\}_y) &\Leftarrow I(x) \wedge I(y) \\ I(\langle x, y \rangle) &\Leftarrow I(x) \wedge I(y) \\ I(x) &\Leftarrow I(\langle x, y \rangle) \\ I(y) &\Leftarrow I(\langle x, y \rangle) \\ I(x) &\Leftarrow I(\{x\}_k) \wedge I(k^{-1}) \end{aligned}$$

Example secrecy question: is n_{ab}^2 accepted at state I .

Weakening the assumption of perfect cryptography

Often the implementation details of encryption algorithms are exploited by protocols or by attackers.

Example 1: modular exponentiation

Messages are of the form $\alpha^{x_1 \dots x_k}$ for some suitable number α .

Permuting the exponents does not change the message.

Model such messages as terms $e(x_1 + \dots + x_k)$ where $e, +$ are fresh symbols. Then the terms obey the laws of the equational theory *ACU*:

Associativity: $x + (y + z) = (x + y) + z$

Commutativity: $x + y = y + x$

Unit: $x + 0 = 0$

(0 is a fresh symbol with $e(0)$ representing α).

Example 2: the *XOR* operation.

Encryption of m with k is often implemented as the bit-wise exclusive-or of m and k . Representing this as $m+k$, we now need the following equational theory:

Associativity: $x+(y+z) = (x+y)+z$

Commutativity: $x+y = y+x$

Unit: $x+0 = 0$

cancellation: $x+x = 0$

E.g. if the intruder knows m_1+K and m_2+K then he can obtain the message m_1+m_2 by performing an *XOR* operation.

To deal with protocols using such operations, we use **equational tree automata**.

Following our convention of representing tree automata as Horn clauses, equational tree automata are simply a set of Horn clauses together with an equational theory.

Consider the following clauses modulo *ACU*:

$$\begin{array}{lll} P_1(a) & P_2(b) & P_3(c) \\ P_4(x+y) \Leftarrow P_1(x) \wedge P_2(y) & & P_5(x+y) \Leftarrow P_4(x) \wedge P_3(y) \\ P_6(x+y) \Leftarrow P_2(x) \wedge P_3(y) & & P_7(x+y) \Leftarrow P_6(x) \wedge P_1(y) \\ P_8(x) \Leftarrow P_5(x) \wedge P_7(x) & & \end{array}$$

The terms $(a+b)+c$, $a+(b+c)$, $(a+c)+b \dots$ are accepted at state P_8 .
In absence of the equational theory, nothing is accepted at P_8 .

ACU automata

Consider first the simple **constants-only** case:

Signature $\Sigma = \{a_1, \dots, a_p\} \cup \{+, 0\}$.

Clauses are of the form

$$P(0) \qquad P(a_i) \qquad P(x+y) \Leftarrow P_1(x) \wedge P_2(y)$$

Already we can accept languages that are non-regular.

E.g. the language $\{t \mid t =_{ACU} na_1 + na_2, n \geq 0\}$ is accepted by ACU automata, but is not regular.

What properties do these automata have?

Modulo *ACU*, terms are of the form $\sum_{i=1}^p n_i a_i$. We can consider them as vectors $(n_1, \dots, n_p) \in \mathbb{N}^p$.

A language is a subset of \mathbb{N}^p .

What kind of languages are accepted by constant-only ACU automata?

Answer: exactly those definable in Presburger arithmetic.

Given tuples $\nu_0, \nu_1, \dots, \nu_k \in \mathbb{N}^p$, define the language

$$L(\nu_0, \nu_1, \dots, \nu_k) = \{\nu_0 + n_1\nu_1 + \dots + n_k\nu_k \mid n_1, \dots, n_k \geq 0\}.$$

A subset of \mathbb{N}^p is called **linear** iff it is of the form $L(\nu_0, \nu_1, \dots, \nu_k)$ for some $\nu_0, \nu_1, \dots, \nu_k \in \mathbb{N}^p$.

Semilinear sets are finite unions of linear sets.

Example: $\nu_0 = (1, 3)$, $\nu_1 = (1, 2)$, $\nu_2 = (0, 3)$

$L(\nu_0, \nu_1, \nu_2) = \{(1 + m, 3 + 2m + 3n) \mid m, n \geq 0\}$

$= \{(1, 3), (2, 5), (3, 7), (2, 8), (2, 11), (3, 10), \dots\}$ is a linear set.

$L' = \{(n, n) \mid n \geq 0\}$ is a linear set.

$L(\nu_0, \nu_1, \nu_2) \cup L'$ is a semilinear set.

Recall: a Presburger formula $\phi(x_1, \dots, x_n)$ defines the set $\{(n_1, \dots, n_k) \in \mathbb{N}^p \mid \phi(n_1, \dots, n_k) \text{ is true}\}$.

Semilinear sets \equiv Presburger definable sets

Connection with string languages: the Parikh mapping

A string x is mapped to the tuple $\text{parikh}(x) = (n_1, \dots, n_p)$ where n_i is the number of occurrences of a_i in x .

A set L of strings is mapped to a set $\text{parikh}(L) \subseteq \mathbb{N}^p$.

For $p = 3$, $\text{parikh}(a_1 a_2 a_2 a_1 a_3 a_1 a_1) = (4, 2, 1)$.

$\text{parikh}((a_1 a_2 a_3 a_2)^* a_2) = \{(n, 2n + 1, n) \mid n \geq 0\}$.

$\text{parikh}(\{a_1^n a_2^n a_3^n \mid n \geq 0\}) = \{(n, n, n) \mid n \geq 0\}$.

Parikh's Theorem

- $L \subseteq \mathbb{N}^p$ is **semilinear** iff $L = \text{parikh}(L')$ for some **regular** language L' .
- $L \subseteq \mathbb{N}^p$ is **semilinear** iff $L = \text{parikh}(L')$ for some **context free** language L' .

ACU automata \equiv context free grammars.

Automata clauses	Productions of CFG
$P(0)$	$P \rightarrow \epsilon$
$P(a_i)$	$P \rightarrow a_i$
$P(x+y) \Leftarrow P_1(x) \wedge P_2(y)$	$P \rightarrow P_1P_2$

If P_f is the final state then we make P_f the starting non-terminal of the CFG.

If L is the language accepted by the automaton, then $parikh(L)$ is the language generated by the CFG.

ACU automata \equiv context free grammars.

Automata clauses	Productions of CFG
$P(0)$	$P \rightarrow \epsilon$
$P(a_i)$	$P \rightarrow a_i$
$P(x+y) \Leftarrow P_1(x) \wedge P_2(y)$	$P \rightarrow P_1 P_2$

If P_f is the final state then we make P_f the starting non-terminal of the CFG.

If L is the language accepted by the automaton, then $parikh(L)$ is the language generated by the CFG.

\Rightarrow Constant-only ACU automata accept exactly semilinear sets.

\Rightarrow Closed under Boolean operations, emptiness decidable.