... liefert an neuen Klauseln:

$$\begin{array}{lll} \mathsf{add}_1(s\,X) & \leftarrow & \mathsf{add}_1(X) \\ \mathsf{add}_1(s\,X) & \leftarrow & \mathsf{add}_0(X) \end{array}$$

Satz

Sei $\mathcal{C}$ eine endliche Menge von Klauseln, bei der bereits die Schritte 1 und 2 ausgeführt wurden und die anschließend unter den Hinzufügungen aus Schritt 3 abgeschlossen ist.

Sei $\mathcal{C}_0 \subseteq \mathcal{C}$ die Teilmenge der normalen Klauseln von $\mathcal{C}$. Dann gilt für alle vorkommenden Prädikate p,

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

Beweis:

Induktion nach der Tiefe der Terme in $[\![p]\!]_{\mathcal{C}}$:-)

... im Beispiel:

Für $\mathsf{add}(()_1\, X)$ erhalten wir die Klauseln:

$$\begin{array}{lll} \mathsf{add}(()_1\, X) & \leftarrow & \mathsf{add}_0(X) \\ \mathsf{add}_0(0) & \leftarrow & \\ \mathsf{add}(()_1\, X) & \leftarrow & \mathsf{add}_1(X) \\ \mathsf{add}_1(s\, X) & \leftarrow & \mathsf{add}_1(X) \\ \mathsf{add}_1(s\, X) & \leftarrow & \mathsf{add}_0(X) \end{array}$$

Die Klauseln sind bereits alle normal :-)

Umwandlung in normale Klauseln:

Führe neue Prädikate für Konjunktionen von Prädikaten ein.

Sei $A = \{p_1, \ldots, p_m\}$. Dann:

$$
\begin{array}{lll}
[A](b) \leftarrow & \text{sofern} & p_i(b) \leftarrow \quad \text{für alle } i. \\
[A](a\,X) \leftarrow [B](X) & \text{sofern} & B = \{p_{ij} \mid i = 1, \ldots, m\} \quad \text{für} \\
 & & p_i(a\,X) \leftarrow p_{i1}(X), \ldots, p_{ir_i}(X)
\end{array}
$$

Letzter Schritt: Umwandlung in einen Typ

- Erst machen wir den Automaten deterministisch ...

  Im Beispiel:

$$\begin{array}{lcll} q(0) & \leftarrow & & \\ q(s\,X) & \leftarrow & q(X) & \text{für} \\ q & = & \{\mathsf{add}_0, \mathsf{add}_1\} & \end{array}$$

Letzter Schritt: Umwandlung in einen Typ

- Erst machen wir den Automaten deterministisch ...

  Im Beispiel:

$$
\begin{array}{lll}
q(0) & \leftarrow & \\
q(s\,X) & \leftarrow & q(X) \qquad \text{für} \\
q & = & \{\mathsf{add}_0, \mathsf{add}_1\}
\end{array}
$$

- Dann fügen wir die Übergänge für die Komponenten des Konstruktors a:

$$p(\langle a, j\rangle\; X) \leftarrow p_j(X)$$

  zu einem Übergang für a zusammen:

$$p(a(X_1, \ldots, X_k)) \leftarrow p_1(X_1), \ldots, p_k(X_k)$$

Im Beispiel finden wir:

$$
\begin{array}{lcll}
\mathsf{add}(())(X,Y,Z) & \leftarrow & q(X), \mathsf{nat}(Y), q'(Z) & \quad \text{wobei} \\
q'(0) & \leftarrow & & \\
q'(s\,X) & \leftarrow & q'(X) & \\
q' & = & \{\mathsf{nat}, \mathsf{add}_2\} &
\end{array}
$$

Die Typen q, q', nat sind alle äquivalent :-)

Diskussion:

- Zur Typüberprüfung reicht es, für einzelne Zustände p zu überprüfen, dass:

$$[\![p]\!]_{C^\sharp} \subseteq \Pi(T)$$

- Da T topdown deterministisch ist, haben wir einen deterministischen Automaten für $\Pi(T)$:-)
- Darum können wir leicht einen DFA für das Komplement $\overline{\Pi(T)}$ konstruieren !!
- Dann überprüfen wir, ob:

$$[\![p]\!]_{C^\sharp} \cap \overline{\Pi(T)} = \emptyset$$

$\Longrightarrow$ das spart uns das Determinisieren :-))

Achtung:

- Das Leerheitsproblem für APS ist DEXPTIME-complete !
- In vielen Fällen terminiert unser Verfahren trotzdem schnell ;-)

Achtung:

- Das Leerheitsproblem für APS ist DEXPTIME-complete !
- In vielen Fällen terminiert unser Verfahren trotzdem schnell ;-)

- Inferierte Typen können auch verwendet werden, um Legacy Code zu verstehen.
- Dann sind sie jedoch nur nützlich, wenn sie nicht zu kompliziert sind !
- Unsere Typinferenz liefert sehr genaue Informationen :-)
- In praktischen Anwendungen wird man vergröbern bzw. die Analyse zu beschleunigen wollen, indem man die Anzahl der auftretenden Mengen reduziert.

Ausblick: Normale Hornklauseln

- Prolog mag als Programmiersprache nicht mehr sehr modern sein :-)
- Hornklauseln eignen sich jedoch perfekt zur Spezifikation von Analyseproblemen.
- Ein anderes Problem ist die Lösung des Analyseproblems :-)
- Lässt sich die kleinste Lösung nicht exakt ausrechnen, erlauben approximative Lösungen aber zumindest approximative Antworten ...

Beispiel: Kryptographische Protokolle

Regeln für den Austausch von Nachrichten:

$$\text{Alice} \xrightarrow{\{\mathsf{Alice}, Na\}_{\mathsf{pub}(\mathsf{Bob})}} \text{Bob}$$

$$\text{Alice} \xleftarrow{\{Na, Nb\}_{\mathsf{pub}(\mathsf{Alice})}} \text{Bob}$$

$$\text{Alice} \xrightarrow{\{Nb\}_{\mathsf{pub}(\mathsf{Bob})}} \text{Bob}$$

Zu überprüfende Eigenschaften:

secrecy, authenticity, ...

Das Dolev-Yao Modell:

- Nachrichten sind Terme:

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

$\Longrightarrow$ Verschiedene Terme repräsentieren unterschiedliche Nachrichten :-)

$\Longrightarrow$ perfekte Kryptographie. Deshalb etwa:

$\{m\}_k = \{m'\}_{k'}$ gdw. $m = m'$ und $k = k'$

- Der Angreifer hat volle Kontrolle über das Netzwerk:

  Alle Nachrichten werden mit dem Angreifer ausgetauscht.

Beispiel: Das Needham-Schroeder Protokoll

$$
\begin{array}{ll}
1. & A \longrightarrow B : \{a, n_a\}_{k_b} \\
2. & B \longrightarrow A : \{n_a, n_b\}_{k_a} \\
3. & A \longrightarrow B : \{n_b\}_{k_b}
\end{array}
$$

Abstraktion:

- Unbeschränkte Anzahl von Sitzungen !!
- Nonces sind möglicherweise nicht-neu ??

Idee:

Modelliere das Wissen des Angreifers durch Hornklauseln ...

$$
\begin{array}{lll}
1. & A \longrightarrow B : \{a, n_a\}_{k_b} & \mathsf{known}(\{a, n_a\}_{k_b}) \;\leftarrow \\
2. & B \longrightarrow A : \{n_a, n_b\}_{k_a} & \mathsf{known}(\{X, n_b\}_{k_a}) \leftarrow \mathsf{known}(\{a, X\}_{k_b}) \\
3. & A \longrightarrow B : \{n_b\}_{k_b} & \mathsf{known}(\{X\}_{k_b}) \leftarrow \mathsf{known}(\{n_a, X\}_{k_a})
\end{array}
$$

Secrecy von N_b : $\quad ?\,\mathsf{known}(n_b).$

Diskussion:

- Wir haben alle Nonces durch endlich viele abstrahiert.
- Weniger restriktive (immer noch korrekte) Abstraktionen sind jedoch möglich ...

1. $A \longrightarrow B : \{a, n_a\}_{k_b} \qquad \ldots$
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. $A \longrightarrow B : \{n_b\}_{k_b} \qquad \ldots$

Die erzeugte Nonce ist eine Funktion der erhaltenen Nonce :-)

Blanchet 2001

Weitere Fähigkeiten des Angreifers:

$$
\begin{array}{lcl}
\mathsf{known}(\{X\}_Y) & \leftarrow & \mathsf{known}(X), \mathsf{known}(Y) \\
 & & // \quad \text{Der Angreifer kann verschlüsseln} \\
\mathsf{known}(\langle X, Y\rangle) & \leftarrow & \mathsf{known}(X), \mathsf{known}(Y) \\
 & & // \quad \text{Der Angreifer kann Paare bilden} \\
\mathsf{known}(X) & \leftarrow & \mathsf{known}(\{X\}_Y), \mathsf{known}(Y) \\
 & & // \quad \text{Der Angreifer kann entschlüsseln} \\
\mathsf{known}(X) & \leftarrow & \mathsf{known}(\langle X, Y\rangle) \\
\mathsf{known}(Y) & \leftarrow & \mathsf{known}(\langle X, Y\rangle) \\
 & & // \quad \text{Der Angreifer kann projezieren}
\end{array}
$$

Diskussion

- Typinferenz für Prolog berechnete eine reguläre Approximation der Menge der Pfade der denotationellen Semantik.
- Diese ist möglicherweise zu ungenau :-(
- Stattdessen approximieren wir die denotationelle Semantik selbst durch reguläre Mengen :-)

- Dies ist allerdings teurer

  $\Longrightarrow$ nicht geeignet für Compiler :-(

  $\Longrightarrow$ i.a. erheblich genauer :-)

Vereinfachung:

Wir betrachten nur Klauseln mit Köpfen der Form:

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

Solche Klauseln nennen wir H1.

Theorem

- Jede Menge von H1-Klauseln ist äquivalent zu einer Menge von einfachen H1-Klauseln der Form:

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

- Jede Menge von H1-Klauseln ist äquivalent zu einer Menge von normalen H1-Klauseln.

Idee:

Wir führen schrittweise einfachere Klauseln hinzu, bis die komplizierten überflüssig werden ...

Regel 1: Splitting

Wir trennen unabängige Anteile der Voraussetzungen ab:

$$\mathit{head} \leftarrow \mathit{rest},\; p_1(X), \ldots, p_m(X)$$

$$(X \text{ kommt nicht in } \mathit{head}, \mathit{rest} \text{ vor})$$

wird ersetzt durch:

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

für ein neues Prädikat q.

Regel 2: Normalisierung

Wir fügen abgeleitete einfachere Klauseln hinzu:

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

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

Schritt 3 (Forts.): Normalisierung

$$\begin{array}{lll} head & \leftarrow & p(t), rest \\ p(X) & \leftarrow & \end{array}$$

impliziert:

$$\begin{array}{lll} head & \leftarrow & rest \end{array}$$

$$\begin{array}{lll} head & \leftarrow & p(b), rest \\ p(b) & \leftarrow & \end{array}$$

impliziert:

$$\begin{array}{lll} head & \leftarrow & rest \end{array}$$

Schritt 3 (Forts.): Normalisierung

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

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

Satz

Sei $\mathcal{C}$ eine endliche Menge von Klauseln, die unter Splitting und Normalisierung abgeschlossen ist.

Sei $\mathcal{C}_0 \subseteq \mathcal{C}$ die Teilmenge der einfachen Klauseln von $\mathcal{C}$. Dann gilt für alle vorkommenden Prädikate p,

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

Beweis:

Induktion nach der Tiefe der Terme in $[\![p]\!]_{\mathcal{C}}$:-)

Umwandlung in normale Klauseln:

Führe neue Prädikate für Konjunktionen von Prädikaten ein.

Sei $A = \{p_1, \ldots, p_m\}$. Dann:

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

Achtung:

- Das Leerheitsproblem für H1 ist DEXPTIME-complete !
- In vielen Fällen terminiert unser Verfahren trotzdem schnell ;-)

- Nicht alle Hornklauseln sind jedoch H1 :-(

$\Longrightarrow$ wir benötigen eine Approximationsmethode ...

Approximation von Hornklauseln

Schritt 1:

Vereinfachung des Rumpfs durch Splitting und Normalisierung (wie gehabt :-)

Schritt 2:

Einführung von Kopien einzelner Variablen X. Jede Kopie erhält die Literale von X als Vorbedingung.

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

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

Schritt 3:

Einführung eines Hilfsprädikats für jeden nicht-variablen Teilterm des Kopfs.

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

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