Electronic Notes in Theoretical Computer Science 62 (2002)
URL: http://www.elsevier.nl/locate/entcs/volume62.html 17 pages

Cryptographic Analysis in Cubic Time

Flemming Nielson [1] Hanne Riis Nielson [2]

Informatics and Mathematical Modelling, Richard Petersens Plads bldg. 321, The Technical University of Denmark, DK-2800 Kongens Lyngby, Denmark.

Helmut Seidl [3]

FB IV – Informatik, Universität Trier, D-54286 Trier, Germany.

Abstract

The spi-calculus is a variant of the polyadic π-calculus that admits symmetric cryptography and that admits expressing communication protocols in a precise though still abstract way. This paper shows that context-independent control flow analysis can be calculated in cubic time despite the fact that the spi-calculus operates over an infinite universe of values. Our approach is based on Horn Clauses with Sharing and we develop transformations to pass from the infinite to the finite and to deal with the polyadic nature of input and output. We prove that this suffices for obtaining a cubic time implementation without sacrificing precision and without making simplifying assumptions on the nature of keys.

1 Introduction

The polyadic π-calculus [9] has been widely used to describe communication protocols. The spi-calculus [1,2] is an extension that has been used to describe and analyse communication protocols based on symmetric cryptography.

More specifically the spi-calculus (see Section 2) contains an explicit operation for encryption (turning plaintext into ciphertext) whereas decryption (turning ciphertext into plaintext) is handled more implicitly by a matching operation. This means that the syntax enforces a distinction between cleartext and plaintext, in that decryption only succeeds when applied to ciphertext, unlike what is the case for the low-level primitives implementing symmetric cryptography.

[1] E-mail: nielson@imm.dtu.dk. This work was supported in part by the Danish Natural Science Research Council.
[2] E-mail: riis@imm.dtu.dk
[3] E-mail: seidl@uni-trier.de

The spi-calculus takes the view [4] that encryption and decryption are perfect. On the one hand, decrypting a message M encrypted with a key K, always succeeds when the key K is used for decryption and gives the correct message M. On the other hand, decryption does not succeed when attempted with a key K′ distinct from K; in particular, it does not erroneously "succeed" although producing a distinct message M′.

This paper considers the context-independent control flow analysis (or 0-CFA) developed in [4,6]. It is specified (in Section 2) as a Flow Logic [10] and operates over an infinite space of values. This makes it harder to implement the analysis than is the case for the π-calculus upon which it is modelled [5]. Indeed, it is not a priori clear that the analysis can be implemented without loss of precision and in cubic time.

As a first step we show (in Section 3) how to transform the analysis from operating over an infinite universe to operating over a finite universe. We perform this step explicitly because we find general logical formulae, in particular Horn Clauses with Sharing or our more recent extension to Alternation-free Least Fixed Point Logic [13], to be so expressive that they are not easily mapped into set-constraints or similar formalisms where this technique is already well-known. We show (in Proposition 3.1) that this transformation is not only correct but that it also incurs no lack of precision.

As a second step we show (in Section 4) how to deal with the polyadic nature of input and output by encoding multi-ary relations using fixed-arity relations. Again we show (in Proposition 4.1) that this transformation is not only correct but that it also incurs no lack of precision.

The final analysis has been transformed into Horn Clauses with Sharing (see Section 5) and implemented using our Succinct Solver. The theoretical evaluation of its run-time shows that it performs in cubic time (see Proposition 5.2) and empirical measurements show that it behaves quite well in practice.

2 The Spi-Calculus

Introduction to the spi-calculus. The polyadic spi-calculus [1,2] extends the π-calculus with primitives for encryption and decryption and this facilitates expressing cryptographic protocols in a rather direct manner. The syntax of *expressions* ($E \in \mathcal{E}$), *terms* ($M, N \in \mathcal{M}$) and *processes* ($P, Q \in \mathcal{P}$) are given by:

[4] It is beyond the scope of the present paper to discuss whether or not this assumption is realistic and whether or not probabilistic reasoning can be used to the same effect. As a consequence, encrypted messages cannot be decrypted even by brute-force attack, unless the key is known.

$$
\begin{aligned}
E &::= M^l \\
M, N &::= n \mid x \mid (E_1, E_2) \mid 0 \mid suc(E) \mid \{E_1, \cdots, E_k\}_E \\
P, Q &::= \mathbf{0} \mid \overline{E}\langle E_1, \cdots, E_k\rangle.P \mid E(x_1^{\beta_1}, \cdots, x_k^{\beta_k}).P \mid P_1|P_2 \mid (\nu n^{\chi})P \mid \\
&\quad [E_1 \; is \; E_2]P \mid !P \mid let \; (x^{\beta_x}, y^{\beta_y}) = E \; in \; P \mid \\
&\quad case \; E \; of \; 0 : P \; suc(x^{\beta}) : Q \mid case \; E \; of \; \{x_1^{\beta_1}, \cdots, x_k^{\beta_k}\}_{E'} \; in \; P
\end{aligned}
$$

The spi-calculus extends the π-calculus with numbers and pairs and distinguishes between names and variables; the term $\{E_1, \cdots, E_k\}_E$ represents the ciphertext obtained by encrypting $E_1, \cdots, E_k$ under the symmetric key E. The process (ignoring the superscript annotations) $let \; (x, y) = E \; in \; P$ behaves as $P[V_1/x, V_2/y]$ if E is (V_1, V_2); the process $case \; E \; of \; 0 : P \; suc(x) : Q$ behaves as P if E is 0, and as $Q[V/x]$ if E is $suc(V)$; $case \; E \; of \; \{x_1, \cdots, x_k\}_{E'} \; in \; P$ attempts to decrypt E with the key E': if E has the form $\{V_1, \cdots, V_k\}_V$ and if E' has the value V, then the process behaves as $P[V_1/x_1, \cdots, V_k/x_k]$; otherwise all the processes above are stuck (i.e. cannot execute) in accordance with the assumption of perfect cryptography. We refer to [1,2,4,6] for a formal semantics of the spi-calculus.

Control flow analysis. The formulation of the control flow analysis is facilitated by annotating the relevant objects of a process. This is done by assigning "labels" (ranged over by $l, l', l_i \in \mathcal{L}$) to the occurrences [5] of terms in order to indicate program points; by assigning "variable types" (ranged over by $\beta, \beta' \in \mathcal{B}$) to the binding occurrences of variables within input actions; and by assigning "channel types" (ranged over by $\chi, \chi' \in \mathcal{C}$) to the binding occurrences of names within restrictions. These annotations are *not changed* by α-conversion.

A process P is analysed with respect to a type environment me (that maps names and variables to their channel and variable types). To state the functionality of the analysis result we use the powerset $\wp(Val)$ of sets of abstract values, ranged over by W, where the infinite set Val of *abstract values*, ranged over by w, v, is *inductively* defined:

$$Val = \mathcal{C} \cup \{0\} \cup \Big(\{suc\} \times Val \Big) \cup \Big(\{pair\} \times Val \times Val \Big) \cup \Big(\{enc\} \times Val^+ \Big)$$

The intended analysis estimate is a triple (R, K, C) where:

- $R : \mathcal{B} \to \wp(Val)$ is the *abstract environment* that maps variable types to the sets of abstract values that they can be bound to.

[5] We use the label l to refer to the term M^l; other authors prefer to dispense with labels and instead to use notation like $\lfloor M \rfloor$ to refer to the term M; the choice is largely a matter of preference.

$$(R,K,C) \models_{me} n^l \text{ iff } \{me(n)\} \subseteq C(l)$$

$$(R,K,C) \models_{me} x^l \text{ iff } R(me(x)) \subseteq C(l)$$

$$(R,K,C) \models_{me} (M_1^{l_1}, M_2^{l_2})^l \text{ iff } (R,K,C) \models_{me} M_1^{l_1} \wedge (R,K,C) \models_{me} M_2^{l_2} \wedge \text{PAIR}(C(l_1), C(l_2)) \subseteq C(l)$$

$$(R,K,C) \models_{me} 0^l \text{ iff } \{0\} \subseteq C(l)$$

$$(R,K,C) \models_{me} suc(M^{l_0})^l \text{ iff } (R,K,C) \models_{me} M^{l_0} \wedge \text{SUC}(C(l_0)) \subseteq C(l)$$

$$(R,K,C) \models_{me} \{M_1^{l_1}, \cdots, M_k^{l_k}\}^l_{N^{l_e}} \text{ iff } \begin{array}{l} \wedge_{i=1}^k (R,K,C) \models_{me} M_i^{l_i} \wedge \\ (R,K,C) \models_{me} N^{l_e} \wedge \\ \text{ENC}\{C(l_1), \cdots, C(l_k)\}_{C(l_e)} \subseteq C(l) \end{array}$$

$$(R,K,C) \models_{me} \mathbf{0} \text{ iff } \mathit{true}$$

$$(R,K,C) \models_{me} \overline{M^l}\langle N_1^{l_1}, \cdots, N_k^{l_k}\rangle.P \text{ iff } \begin{array}{l} (R,K,C) \models_{me} M^l \wedge \\ \wedge_{i=1}^k (R,K,C) \models_{me} N_i^{l_i} \wedge \\ C(l) \neq \emptyset \Rightarrow (R,K,C) \models_{me} P \wedge \\ \forall \chi \in C(l) : C(l_1) \times \cdots \times C(l_k) \subseteq K(\chi) \end{array}$$

$$(R,K,C) \models_{me} M^l(x_1^{\beta_1}, \cdots, x_k^{\beta_k}).P \text{ iff } \begin{array}{l} (R,K,C) \models_{me} M^l \wedge \\ C(l) \neq \emptyset \Rightarrow (R,K,C) \models_{me[\tilde{x} \mapsto \tilde{\beta}]} P \wedge \\ \forall \chi \in C(l) : \forall (w_1, \cdots, w_{k'}) \in K(\chi) : \\ \qquad k = k' \Rightarrow \wedge_{i=1}^k \{w_i\} \subseteq R(\beta_i) \end{array}$$

$$(R,K,C) \models_{me} P_1 | P_2 \text{ iff } (R,K,C) \models_{me} P_1 \wedge (R,K,C) \models_{me} P_2$$

$$(R,K,C) \models_{me} (\nu n^\chi) P \text{ iff } (R,K,C) \models_{me[n \mapsto \chi]} P$$

$$(R,K,C) \models_{me} !P \text{ iff } (R,K,C) \models_{me} P$$

$$(R,K,C) \models_{me} [M_1^{l_1} \mathit{ is } M_2^{l_2}]P \text{ iff } \begin{array}{l} (R,K,C) \models_{me} M_1^{l_1} \wedge (R,K,C) \models_{me} M_2^{l_2} \wedge \\ C(l_1) \cap C(l_2) \neq \emptyset \Rightarrow (R,K,C) \models_{me} P \end{array}$$

$$(R,K,C) \models_{me} \begin{array}{l} \mathit{let}\ (x^{\beta_x}, y^{\beta_y}) = M^l \\ \mathit{in}\ P \end{array} \text{ iff } \begin{array}{l} (R,K,C) \models_{me} M^l \wedge \\ C(l) \neq \emptyset \Rightarrow (R,K,C) \models_{me[x \mapsto \beta_x, y \mapsto \beta_y]} P \wedge \\ \forall\, pair(w_1, w_2) \in C(l) : \\ \qquad \{w_1\} \subseteq R(\beta_x) \wedge \{w_2\} \subseteq R(\beta_y) \end{array}$$

$$(R,K,C) \models_{me} \begin{array}{l} \mathit{case}\ M^l\ \mathit{of} \\ 0 : P\ \mathit{suc}(x^\beta) : Q \end{array} \text{ iff } \begin{array}{l} (R,K,C) \models_{me} M^l \wedge \\ 0 \in C(l) \Rightarrow (R,K,C) \models_{me} P \wedge \\ (\exists\, suc(w) : suc(w) \in C(l)) \Rightarrow (R,K,C) \models_{me[x \mapsto \beta]} Q \wedge \\ \forall\, suc(w) \in C(l) : \{w\} \subseteq R(\beta) \end{array}$$

$$(R,K,C) \models_{me} \begin{array}{l} \mathit{case}\ M^l\ \mathit{of} \\ \{x_1^{\beta_1}, \cdots, x_k^{\beta_k}\}_{N^{l_d}} \\ \mathit{in}\ P \end{array} \text{ iff } \begin{array}{l} (R,K,C) \models_{me} M^l \wedge (R,K,C) \models_{me} N^{l_d} \wedge \\ (\exists\, enc\{w_1, \cdots, w_{k'}\}_{w_e} \in C(l) : k = k' \wedge \\ \qquad w_e \in C(l_d)) \Rightarrow (R,K,C) \models_{me[\tilde{x} \mapsto \tilde{\beta}]} P \wedge \\ \forall\, enc\{w_1, \cdots, w_{k'}\}_{w_e} \in C(l) : \\ \qquad (k = k' \wedge w_e \in C(l_d)) \Rightarrow \wedge_{i=1}^k \{w_i\} \subseteq R(\beta_i) \end{array}$$

Table 1
Flow Logic for the spi-calculus (adapted from [4,6]).

- $K : \mathcal{C} \to \wp(Val^*)$ is the *abstract channel environment* that maps channel types to the sets of tuples of abstract values that can be communicated over them.
- $C : \mathcal{L} \to \wp(Val)$ is the *abstract cache* that maps labelled terms to the sets of abstract values that the term can evaluate to.

The *acceptability* of an estimate (R, K, C) is defined by the judgements:

$$(R, K, C) \models_{me} P \quad \text{and} \quad (R, K, C) \models_{me} M$$

The analysis of expressions and of processes is given in Table 1 and makes use of the following shorthands:

- $suc(w)$ for (suc, w), and $\text{SUC}(W)$ for $\{suc(w) | w \in W\}$;
- $pair(w_1, w_2)$ for $(pair, w_1, w_2)$, and $\text{PAIR}(W_1, W_2)$ for $\{pair(w_1, w_2) | w_1 \in W_1, w_2 \in W_2\}$;
- $enc\{w_1, \cdots, w_k\}_{w_0}$ for $(enc, (w_1, \cdots, w_k, w_0))$, and $\text{ENC}\{W_1, \cdots, W_k\}_{W_0}$ for $\{enc\{w_1, \cdots, w_k\}_{w_0} | w_0 \in W_0, \cdots, w_k \in W_k\}$.

All rules for validating a compound term or process require that the components are validated (except when it is blatantly clear that they are unreachable). The rules for an expression M^l demand that $C(l)$ contains all the abstract values associated with M. Moreover, the rule for output requires that the sets of $k-$tuples of abstract values associated with each component of the object can be passed on each channel associated with the subject. Symmetrically, the rule for input requires that for each $k-$tuple of abstract values passing along the subject, the corresponding components are included among the values of $x_1, \cdots, x_k$. The last three rules ask that for each abstract value associated with the expression to decompose, its sub-components are contained, componentwise, in $x_1, \cdots, x_k$.

The above analysis has been adapted from [4,6] where further explanations and proofs of semantic correctness can be found. They also establish that

$$\{(R, K, C) \mid (R, K, C) \models_{me} P\} \quad \text{is a Moore family}$$

meaning that it is closed under greatest lower bounds; it follows that its least element, $\sqcap\{(R, K, C) \mid (R, K, C) \models_{me} P\}$, itself safisfies the acceptability judgement.

The analysis is closest to the presentation in [4] in that it avoids the notion of history-free cryptography studied in [6]. Since we do not deal with the dynamic semantics we have dispensed with the syntactic extensions "abstraction" and "concretion" used in the semantics.

3 From the Infinite to the Finite

It is not immediate how to implement the analysis in Table 1 because it operates over sets of values of unbounded size; we therefore explicitly massage the specification to obtain Table 2 that operates only over a finite universe. The motivation behind the development performed here is the observation that grammar-based or tree-automata based approaches to describing infinite sets of values by finite representations works in the case of model-checking [8] and set-constraints [3]. In essence we show that Table 1 may be viewed as defining a finite tree-grammar with nonterminals of the form $R(\beta)$, $K(\kappa)$ and $C(l)$ despite the use of intersections of sets.

So instead of using the set Val of abstract values we shall be using the set

$$DVal = \mathcal{C} \cup \{0\} \cup \Big(\{suc\} \times \mathcal{L} \Big) \cup \Big(\{pair\} \times \mathcal{L} \times \mathcal{L} \Big) \cup \Big(\{enc\} \times \mathcal{L}^+ \Big)$$

of descriptions of values that only records the "top-level" structure of terms; then $R : \mathcal{B} \to \wp(DVal)$, $K : \mathcal{C} \to \wp(DVal^*)$ and $C : \mathcal{L} \to \wp(DVal)$. The specification of the new analysis is given in Table 2 writing $\models'_{me} P$ for $(R, K, C) \models'_{me} P$ to save space. Also we write $[\![C]\!](l)$ for the tree-language generated by the nonterminal $C(l)$ and the test $k = k'$ has been made implicit (by only considering the possibility that k' equals k). To avoid confusion we shall later write R', K' and C' for R, K and C as they relate to Table 2 (unless there is no risk of confusion).

In preparation for establishing the relationship between the specifications of Tables 1 and 2, and for formally defining the notation used in Table 2, we define three auxiliary operations. First, define the "one-level" extension $G^\# : DVal \to \wp(Val)$ of a function $G : \mathcal{L} \to \wp(Val)$ as follows:

$$\begin{aligned}
G^\#[\![\chi]\!] &= \{\chi\} \\
G^\#[\![0]\!] &= \{0\} \\
G^\#[\![suc(l_0)]\!] &= \textsc{suc}(G(l_0)) \\
G^\#[\![pair(l_1, l_2)]\!] &= \textsc{pair}(G(l_1), G(l_2)) \\
G^\#[\![enc\{l_1, \cdots, l_k\}_{l_0}]\!] &= \textsc{enc}(\{G(l_1), \cdots, G(l_k)\}_{G(l_0)})
\end{aligned}$$

Second, the pointwise extension of $G^\# : DVal \to \wp(Val)$ to sequences of elements from $DVal$ is denoted $G^{\overline{\#}} : DVal^* \to \wp(Val^*)$ and is defined by:

$$G^{\overline{\#}}[\![(w_1, \cdots, w_k)]\!] = G^\#[\![w_1]\!] \times \cdots \times G^\#[\![w_k]\!]$$

Third, define the pointwise extension $H^\dagger : \wp(DVal) \to \wp(Val)$ of a function $H : DVal \to \wp(Val)$ as follows:

$$H^\dagger(W) = \bigcup_{w \in W} H(w)$$

$$
\begin{array}{rl}
\models'_{me} n^l \text{ iff } & \{me(n)\} \subseteq C(l) \\
\models'_{me} x^l \text{ iff } & R(me(x)) \subseteq C(l) \\
\models'_{me} (M_1^{l_1}, M_2^{l_2})^l \text{ iff } & \models'_{me} M_1^{l_1} \wedge \models'_{me} M_2^{l_2} \wedge \{pair(l_1, l_2)\} \subseteq C(l) \\
\models'_{me} 0^l \text{ iff } & \{0\} \subseteq C(l) \\
\models'_{me} suc(M^{l_0})^l \text{ iff } & \models'_{me} M^{l_0} \wedge \{suc(l_0)\} \subseteq C(l) \\
\models'_{me} \{M_1^{l_1}, \cdots, M_k^{l_k}\}^l_{N^{l_e}} \text{ iff } & \wedge_{i=1}^k \models'_{me} M_i^{l_i} \wedge \models'_{me} N^{l_e} \wedge \\
& \{enc\{l_1, \cdots, l_k\}_{l_e}\} \subseteq C(l)
\end{array}
$$

$$
\begin{array}{rl}
\models'_{me} \mathbf{0} \text{ iff } & true \\
\models'_{me} \overline{M^l}\langle N_1^{l_1}, \cdots, N_k^{l_k}\rangle.P \text{ iff } & \models'_{me} M^l \wedge \wedge_{i=1}^k \models'_{me} N_i^{l_i} \wedge (\llbracket C \rrbracket(l) \neq \emptyset \Rightarrow \models'_{me} P) \wedge \\
& \forall \chi \in C(l) : C(l_1) \times \cdots \times C(l_k) \subseteq K(\chi) \\
\models'_{me} M^l(x_1^{\beta_1}, \cdots, x_k^{\beta_k}).P \text{ iff } & \models'_{me} M^l \wedge (\llbracket C \rrbracket(l) \neq \emptyset \Rightarrow \models'_{me[\tilde{x} \mapsto \tilde{\beta}]} P) \wedge \\
& \forall \chi \in C(l) : \forall (w_1, \cdots, w_k) \in K(\chi) : \wedge_{i=1}^k \{w_i\} \subseteq R(\beta_i) \\
\models'_{me} P_1 | P_2 \text{ iff } & \models'_{me} P_1 \wedge \models'_{me} P_2 \\
\models'_{me} (\nu n^\chi) P \text{ iff } & \models'_{me[n \mapsto \chi]} P \\
\models'_{me} !P \text{ iff } & \models'_{me} P \\
\models'_{me} [M_1^{l_1} \text{ is } M_2^{l_2}] P \text{ iff } & \models'_{me} M_1^{l_1} \wedge \models'_{me} M_2^{l_2} \wedge (\llbracket C \rrbracket(l_1) \cap \llbracket C \rrbracket(l_2) \neq \emptyset \Rightarrow \models'_{me} P) \\
\models'_{me} \begin{array}{l} let\ (x^{\beta_x}, y^{\beta_y}) = M^l \\ in\ P \end{array} \text{ iff } & \models'_{me} M^l \wedge (\llbracket C \rrbracket(l) \neq \emptyset \Rightarrow \models'_{me[x \mapsto \beta_x, y \mapsto \beta_y]} P) \wedge \\
& \forall\, pair(l_1, l_2) \in C(l) : (C(l_1) \subseteq R(\beta_x) \wedge C(l_2) \subseteq R(\beta_y)) \\
\models'_{me} \begin{array}{l} case\ M^l\ of \\ 0 : P\ suc(x^\beta) : Q \end{array} \text{ iff } & \models'_{me} M^l \wedge (0 \in C(l) \Rightarrow \models'_{me} P) \wedge \\
& ((\exists\, suc(l_0) : suc(l_0) \in C(l) \wedge \llbracket C \rrbracket(l_0) \neq \emptyset) \Rightarrow \models'_{me[x \mapsto \beta]} Q) \wedge \\
& \forall\, suc(l_0) \in C(l) : C(l_0) \subseteq R(\beta) \\
\models'_{me} \begin{array}{l} case\ M^l\ of \\ \{x_1^{\beta_1}, \cdots, x_k^{\beta_k}\}_{N^{l_d}} \\ in\ P \end{array} \text{ iff } & \models'_{me} M^l \wedge \models'_{me} N^{l_d} \wedge \\
& ((\exists\, enc\{l_1, \cdots, l_k\}_{l_e} \in C(l) : \llbracket C \rrbracket(l_e) \cap \llbracket C \rrbracket(l_d) \neq \emptyset \wedge \\
& \qquad \wedge_{i=1}^k \llbracket C \rrbracket(l_i) \neq \emptyset) \Rightarrow \models'_{me[\tilde{x} \mapsto \tilde{\beta}]} P) \wedge \\
& \forall\, enc\{l_1, \cdots, l_k\}_{l_e} \in C(l) : \\
& \qquad (\llbracket C \rrbracket(l_e) \cap \llbracket C \rrbracket(l_d) \neq \emptyset) \Rightarrow \wedge_{i=1}^k C(l_i) \subseteq R(\beta_i)
\end{array}
$$

Table 2
Flow Logic over a finite universe for the spi-calculus.

Given a function $C : \mathcal{L} \to \wp(DVal)$ we are now ready to formally define the function $\llbracket C \rrbracket : \mathcal{L} \to \wp(Val)$ used in Table 2 and informally specified above. It is *inductively* defined by the equation

$$\llbracket C \rrbracket = \llbracket C \rrbracket^{\#\dagger} \circ C$$

which is equivalent to setting $\llbracket C \rrbracket(l) = \bigcup_{w \in C(l)} \llbracket C \rrbracket^{\#}(w)$ for all $l \in \mathcal{L}$. In other words, $\llbracket C \rrbracket$ intuitively is the language generated by the tree-grammar

C. Finally, we define a concretization function γ by

$$\begin{aligned}\gamma(R,K,C) &= (\llbracket C \rrbracket^{\#\dagger} \circ R,\ \llbracket C \rrbracket^{\overline{\#}\dagger} \circ K,\ \llbracket C \rrbracket^{\#\dagger} \circ C) \\ &= (\llbracket C \rrbracket^{\#\dagger} \circ R,\ \llbracket C \rrbracket^{\overline{\#}\dagger} \circ K,\ \llbracket C \rrbracket)\end{aligned}$$

and use it to state that the analysis in Table 1 computes the *same* least solution as the one in Table 2:

Proposition 3.1

$$\gamma\left(\sqcap\{(R',K',C') \mid (R',K',C') \models'_{me} P\}\right) = \sqcap\{(R,K,C) \mid (R,K,C) \models_{me} P\}$$

Proof. The proof is given in Appendix A. □

4 Getting Rid of Polyvariance

Assuming that the arity of the polyadic operations (i.e. encryption and decryption as well as input and output) is bounded by some constant, one can show that the formulation of Table 2 can be implemented in polynomial time, by using the techniques of Section 5 for translating it to *linear* Horn Clauses with Sharing [12]. To obtain a guaranteed cubic time algorithm we begin by taking a closer look at how to deal with the polyadicity of input and output.

For this we replace the abstract channel environment $K' : \mathcal{C} \to \wp(DVal^*)$ of Section 3 with a component

- $K'' : \mathcal{C} \times \mathbf{N} \times \mathbf{N} \to \wp(DVal)$ such that $v \in K'(\chi,i,k)$ whenever v is the i'th component of a k-tuple in $K'(\chi)$.

(As before we shall sometimes write K for K'' when no confusion is likely to arise.) Formally the relationship between the two notions of abstract channel environment can be captured by a concretization function γ' defined by:

$$\gamma'(R,K'',C) = (R,K',C) \text{ where } K'(\chi) = \{(v_1,\cdots,v_k) \mid \forall i : v_i \in K''(\chi,i,k)\}$$

Due to its pointwise definition it induces an abstraction function α' such that (α',γ') is a Galois connection. We shall say that a triple (R,K'',C) is *well-formed* whenever $\alpha'(\gamma'(R,K'',C)) = (R,K'',C)$; this just means that no $K''(\chi,i,k)$ is empty for $1 \le i \le k$ unless all of $K(\chi,1,k),\cdots,K''(\chi,k,k)$ are, and that $K''(\chi,i,k)$ is empty for $i > k$.

To make this work we need to change the clauses for analysing input and output; this results in a new specification $\models''$ that differs from $\models'$ as follows

(writing K for K''):

$$\begin{array}{ll}\models''_{me} \overline{M^l}\langle N_1^{l_1},\cdots,N_k^{l_k}\rangle.P \text{ iff } & \models''_{me} M^l \wedge \bigwedge_{i=1}^k \models''_{me} N_i^{l_i} \wedge (\llbracket C \rrbracket(l) \neq \emptyset \Rightarrow \models''_{me} P) \wedge \\ & (\bigwedge_{i=1}^k C(l_i) \neq \emptyset) \Rightarrow \\ & \quad \bigwedge_{i=1}^k \forall \chi \in C(l) : C(l_i) \subseteq K(\chi,i,k) \\ \models''_{me} M^l(x_1^{\beta_1},\cdots,x_k^{\beta_k}).P \text{ iff } & \models''_{me} M^l \wedge (\llbracket C \rrbracket(l) \neq \emptyset \Rightarrow \models''_{me[\tilde{x}\mapsto\tilde{\beta}]} P) \wedge \\ & \bigwedge_{i=1}^k \forall \chi \in C(l) : K(\chi,i,k) \subseteq R(\beta_i)\end{array}$$

The formal relationship between the two analyses can now be stated:

Proposition 4.1

$$\gamma'\left(\sqcap\{(R,K'',C) \mid (R,K'',C) \models''_{me} P\}\right) = \sqcap\{(R,K',C) \mid (R,K',C) \models'_{me} P\}$$

Proof. The proof is in four parts. First, it is a straightforward structural induction on P and M to prove that

$$\begin{array}{lcl}(R,K'',C) \models''_{me} P & \Leftrightarrow & (\gamma'(R,K'',C)) \models'_{me} P \\ (R,K'',C) \models''_{me} M & \Leftrightarrow & (\gamma'(R,K'',C)) \models'_{me} M\end{array}$$

whenever (R,K'',C) is well-formed.

Second, we establish that $\sqcap\{(R,K'',C) \mid (R,K,C) \models''_{me} P\}$ is well-formed. The proof is by contradiction. So suppose that the least (R,K'',C) satisfying $\models''_{me} P$ is not well-formed.

One possibility is that there exists χ, k, $i \leq k$ and $j \leq k$ such that $K''(\chi,i,k) = \emptyset$ but $K''(\chi,j,k) \neq \emptyset$. By the choice of (R,K'',C) there must be a constraint that forces $K''(\chi,j,k)$ to be non-empty; by inspection of the clauses it is clear that this constraint must be imposed by the analysis of output where it takes the form

$$(\bigwedge_{i=1}^k C(l_i) \neq \emptyset) \Rightarrow \bigwedge_{i=1}^k \forall \chi \in C(l) : C(l_i) \subseteq K''(\chi,i,k).$$

However, then it is clear that also $K''(\chi,i,k)$ must be non-empty, thereby establishing the desired contradiction.

Another possibility is that there exists χ, k, $i > k$ such that $K''(\chi,i,k) \neq \emptyset$. As before there must be a constraint that forces $K''(\chi,i,k)$ to be non-empty; however, inspection of the clauses shows this to be impossible.

Third, to prove "$\sqsupseteq$" in the statement of the Proposition, we note that

$$\gamma'(\sqcap\{(R,K'',C) \mid (R,K'',C) \models''_{me} P\}) \models'_{me} P$$

as follows because $\{(R,K'',C) \mid (R,K'',C) \models''_{me} P\}$ is a Moore family, because $\sqcap\{(R,K'',C) \mid (R,K'',C) \models''_{me} P\}$ is well-formed and because of the "if and only if" established above.

Fourth, to prove "$\sqsubseteq$" in the statement of the Proposition, simply note that

$$\gamma'(\alpha'(\sqcap\{(R,K',C) \mid (R,K',C) \models'_{me} P\})) = \sqcap\{(R,K',C) \mid (R,K',C) \models'_{me} P\}$$

$$\alpha'(\sqcap\{(R,K',C) \mid (R,K',C) \models'_{me} P\}) \models''_{me} P$$

because $\{(R,K',C) \mid (R,K',C) \models'_{me} P\}$ is a Moore family, $\alpha'(\sqcap\{(R,K',C) \mid (R,K',C) \models'_{me} P\}$ is well-formed and because of the "$\Leftrightarrow$" established above.$\Box$

The intuitive content of Proposition 4.1 is that the "relational formulation" (see e.g. [11]) of output in Table 2 is only "apparent" in that it considers *all* possible combinations of values; hence the "relational formulation" can be converted into an "independent attribute formulation" as illustrated above, without any loss of precision.

5 Horn Clauses with Sharing

Review of Horn Clauses with Sharing. The set of Horn Clauses with Sharing was introduced in [12] as a useful subset of the *Alternation-free Least Fixed Point Logic* implemented in our *Succinct Solver*. Horn Clauses with Sharing may be viewed as extending Horn Clauses by more powerful preconditions and conclusions; they are formally defined by the nonterminal clause generated by the grammar:

$$\begin{array}{lcl}
\mathsf{pre} & ::= & R\,(x_1,\cdots,x_k) \quad | \quad \mathsf{pre}_1 \wedge \mathsf{pre}_2 \quad | \quad \mathsf{pre}_1 \vee \mathsf{pre}_2 \quad | \quad \exists x : \mathsf{pre} \\
\mathsf{clause} & ::= & R\,(x_1,\cdots,x_k) \quad | \\
 & & \mathbf{1} \quad | \quad \mathsf{clause}_1 \wedge \mathsf{clause}_2 \quad | \quad \mathsf{pre} \Rightarrow \mathsf{clause} \quad | \quad \forall x : \mathsf{clause}
\end{array}$$

where R is a k-ary relation symbol for $k \geq 1$, $x, x_1, \cdots$ denote arbitrary variables, and $\mathbf{1}$ is the always true clause.

Given a universe $\mathcal{U}$ of atomic values and interpretations ρ and σ for relation symbols and free variables, respectively, we define the satisfaction relation

$$(\rho,\sigma) \models t$$

(t a pre-condition or clause) as follows:

$$\begin{array}{lll}
(\rho,\sigma) \models \mathbf{1} & \text{iff} & \text{true} \\
(\rho,\sigma) \models R\,(x_1,\cdots,x_k) & \text{iff} & (\sigma\,x_1,\cdots,\sigma\,x_k) \in \rho\,R \\
(\rho,\sigma) \models \exists x : \mathsf{pre} & \text{iff} & (\rho,\sigma \oplus \{x \mapsto a\}) \models \mathsf{pre} \qquad \text{for some } a \in \mathcal{U} \\
(\rho,\sigma) \models \forall x : t & \text{iff} & (\rho,\sigma \oplus \{x \mapsto a\}) \models t \qquad \text{for all } a \in \mathcal{U} \\
(\rho,\sigma) \models t_1 \wedge t_2 & \text{iff} & (\rho,\sigma) \models t_1 \quad \text{and} \quad (\rho,\sigma) \models t_2 \\
(\rho,\sigma) \models \mathsf{pre}_1 \vee \mathsf{pre}_2 & \text{iff} & (\rho,\sigma) \models \mathsf{pre}_1 \quad \text{or} \quad (\rho,\sigma) \models \mathsf{pre}_2 \\
(\rho,\sigma) \models \mathsf{pre} \Rightarrow \mathsf{clause} & \text{iff} & (\rho,\sigma) \models \mathsf{clause} \quad \text{whenever} \quad (\rho,\sigma) \models \mathsf{pre}
\end{array}$$

We view the free variables occurring in a Horn Clause with Sharing as constant symbols from the universe $\mathcal{U}$. Thus, given an interpretation σ of the constant

$$
\begin{aligned}
\mathcal{H}[\![n^l]\!]_{me} &= C(me(n), l)\\
\mathcal{H}[\![x^l]\!]_{me} &= \forall u : R(u, me(x)) \Rightarrow C(u, l)\\
\mathcal{H}[\![(M_1^{l_1}, M_2^{l_2})^l]\!]_{me} &= \mathcal{H}[\![M_1^{l_1}]\!]_{me} \wedge \mathcal{H}[\![M_2^{l_2}]\!]_{me} \wedge C(pair(l_1, l_2), l)\\
\mathcal{H}[\![0^l]\!]_{me} &= C(0, l)\\
\mathcal{H}[\![suc(M^{l_0})^l]\!]_{me} &= \mathcal{H}[\![M^{l_0}]\!]_{me} \wedge C(suc(l_0), l)\\
\mathcal{H}[\![\{M_1^{l_1}, \cdots, M_k^{l_k}\}^l_{N^{l_e}}]\!]_{me} &= \wedge_{i=1}^k \mathcal{H}[\![M_i^{l_i}]\!]_{me} \wedge \mathcal{H}[\![N^{l_e}]\!]_{me} \wedge C(enc(\{l_1, \cdots, l_k\}_{l_e}), l)
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{H}[\![\mathbf{0}]\!]_{me} &= \mathbf{1}\\
\mathcal{H}[\![\overline{M^l}\langle N_1^{l_1}, \cdots, N_k^{l_k}\rangle.P]\!]_{me} &= \begin{array}{l}\mathcal{H}[\![M^l]\!]_{me} \wedge \wedge_{i=1}^k \mathcal{H}[\![N_i^{l_i}]\!]_{me} \wedge (NC(l) \Rightarrow \mathcal{H}[\![P]\!]_{me}) \wedge\\ ((\wedge_{i=1}^k NC(l_i)) \Rightarrow \wedge_{i=1}^k \forall v : (\mathcal{C}(v) \wedge C(v, l) \Rightarrow\\ \qquad \forall u_i : C(u_i, l_i) \Rightarrow K(u_i, v, i, k)))\end{array}\\
\mathcal{H}[\![M^l(x_1^{\beta_1}, \cdots, x_k^{\beta_k}).P]\!]_{me} &= \begin{array}{l}\mathcal{H}[\![M^l]\!]_{me} \wedge (NC(l) \Rightarrow \mathcal{H}[\![P]\!]_{me[\tilde{x} \mapsto \tilde{\beta}]}) \wedge\\ \wedge_{i=1}^k \forall v : (\mathcal{C}(v) \wedge C(v, l) \Rightarrow\\ \qquad \forall u_i : K(u_i, v, i, k) \Rightarrow R(u_i, \beta_i))\end{array}\\
\mathcal{H}[\![P_1|P_2]\!]_{me} &= \mathcal{H}[\![P_1]\!]_{me} \wedge \mathcal{H}[\![P_2]\!]_{me}\\
\mathcal{H}[\![(\nu n^\chi)P]\!]_{me} &= \mathcal{H}[\![P]\!]_{me[n \mapsto \chi]}\\
\mathcal{H}[\![!P]\!]_{me} &= \mathcal{H}[\![P]\!]_{me}\\
\mathcal{H}[\![[M_1^{l_1} \ is \ M_2^{l_2}]P]\!]_{me} &= \mathcal{H}[\![M_1^{l_1}]\!]_{me} \wedge \mathcal{H}[\![M_2^{l_2}]\!]_{me} \wedge (NCC(l_1, l_2) \Rightarrow \mathcal{H}[\![P]\!]_{me})\\
\mathcal{H}[\![\begin{array}{l}let\ (x^{\beta_x}, y^{\beta_y}) = M^l\\ in\ P\end{array}]\!]_{me} &= \begin{array}{l}\mathcal{H}[\![M^l]\!]_{me} \wedge (NC(l) \Rightarrow \mathcal{H}[\![P]\!]_{me[x \mapsto \beta_x, y \mapsto \beta_y]}) \wedge\\ \forall\ pair(l_1, l_2) : (C(pair(l_1, l_2), l) \Rightarrow\\ \qquad (\forall u : C(u, l_1) \Rightarrow R(u, \beta_x)) \wedge\\ \qquad (\forall u : C(u, l_2) \Rightarrow R(u, \beta_y)))\end{array}\\
\mathcal{H}[\![\begin{array}{l}case\ M^l\ of\\ 0 : P\ suc(x^\beta) : Q\end{array}]\!]_{me} &= \begin{array}{l}\mathcal{H}[\![M^l]\!]_{me} \wedge (C(0, l) \Rightarrow \mathcal{H}[\![P]\!]_{me}) \wedge\\ ((\exists\ suc(l_0) : C(suc(l_0), l) \wedge NC(l_0)) \Rightarrow \mathcal{H}[\![Q]\!]_{me[x \mapsto \beta]}) \wedge\\ \forall\ suc(l_0) : C(suc(l_0), l) \Rightarrow (\forall u : C(u, l_0) \Rightarrow R(u, \beta))\end{array}\\
\mathcal{H}[\![\begin{array}{l}case\ M^l\ of\\ \{x_1^{\beta_1}, \cdots, x_k^{\beta_k}\}_{N^{l_d}}\\ in\ P\end{array}]\!]_{me} &= \begin{array}{l}\mathcal{H}[\![M^l]\!]_{me} \wedge \mathcal{H}[\![N^{l_d}]\!]_{me} \wedge\\ ((\exists\ enc\{l_1, \cdots, l_k\}_{l_e} : C(enc\{l_1, \cdots, l_k\}_{l_e}, l) \wedge NCC(l_e, l_d) \wedge\\ \qquad \wedge_{i=1}^k NC(l_i)) \Rightarrow \mathcal{H}[\![P]\!]_{me[\tilde{x} \mapsto \tilde{\beta}]}) \wedge\\ \forall\ enc\{l_1, \cdots, l_k\}_{l_e} : (C(enc(\{l_1, \cdots, l_k\}_{l_e}), l) \wedge NCC(l_e, l_d))\\ \qquad \Rightarrow \wedge_{i=1}^k \forall u : C(u, l_i) \Rightarrow R(u, \beta_i)\end{array}
\end{aligned}
$$

Table 3
Horn Clauses with Sharing for the spi-calculus.

symbols, in the clause clause, we call an interpretation ρ of the relational symbols $\mathcal{R}$ a *solution* provided $(\rho, \sigma) \models \mathsf{clause}$.

Transformation to Horn Clauses with Sharing. Table 3 contains the constraint generation function corresponding to Table 2. Set inclusions have been expanded to set memberships using an additional universal quantifier and a set membership of the form $u \in R(v)$ has been written using a binary predicate

$$\forall l_1, l_2 : (\exists\, u : \mathcal{C}(u) \wedge C(u, l_1) \wedge C(u, l_2)) \Rightarrow NCC(l_1, l_2)$$

$$\forall l_1, l_2 : (C(0, l_1) \wedge C(0, l_2)) \Rightarrow NCC(l_1, l_2)$$

$$\forall l_1, l_2 : (\exists\, suc(u_1), suc(u_2) :$$
$$C(suc(u_1), l_1) \wedge C(suc(u_2), l_2) \wedge NCC(u_1, u_2)) \Rightarrow NCC(l_1, l_2)$$

$$\forall l_1, l_2 : (\exists\, pair(u_{11}, u_{12}), pair(u_{21}, u_{22}) :$$
$$C(pair(u_{11}, u_{12}), l_1) \wedge C(pair(u_{21}, u_{22}), l_2) \wedge$$
$$NCC(u_{11}, u_{21}) \wedge NCC(u_{12}, u_{22})) \Rightarrow NCC(l_1, l_2)$$

$$\forall l_1, l_2 : (\exists\, enc\{u_{11}, \cdots, u_{1k}\}_{u_1}, enc\{u_{21}, \cdots, u_{2k}\}_{u_2} :$$
$$C(enc\{u_{11}, \cdots, u_{1k}\}_{u_1}, l_1) \wedge C(enc\{u_{21}, \cdots, u_{2k}\}_{u_2}, l_2) \wedge$$
$$NCC(u_{11}, u_{21}) \wedge \cdots \wedge NCC(u_{1k}, u_{2k}) \wedge NCC(u_1, u_2)) \Rightarrow NCC(l_1, l_2)$$

Table 4
Axiomatization of NCC.

of the form $R(u, v)$. Also we are now explicit about variables such as χ that were only supposed to range over channels in $\mathcal{C}$. (The predicate $\mathcal{C}$ is assumed to be predefined; alternatively it could be updated in the clause for restriction where a new channel is introduced.)

For readability, as well as for ease of complexity estimation, we have retained the use of function symbols like *pair*; this could be avoided by *(i)* using an additional ternary relation R_{pair}, *(ii)* defining $R_{pair}(l_1, l_2, pair(l_1, l_2))$ whenever the pair $pair(l_1, l_2)$ is constructed, and *(iii)* extracting the components of all pairs in $C(\cdot, l)$ by quantifying over all l_1, l_2, l_{pair} such that $C(l_{pair}, l) \wedge R_{pair}(l_1, l_2, l_{pair})$.

Finally we have had to code the tests $[\![C]\!](l) \neq \emptyset$ and $[\![C]\!](l_e) \cap [\![C]\!](l_d) \neq \emptyset$ using the primitives allowed. We therefore introduce the auxiliary predicates $NC(l)$ and $NCC(l_e, l_d)$ and axiomatize them inductively by considering each of the five diffferent formations of values in $DVal$; this encoding is global and should not be repeated for each syntactic component. The axiomatization of NCC can be found in Table 4; the axiomatization of NC can be obtained by merely removing the variables occurring in the second parameter of NCC.

Proposition 5.1

$$\sqcap\{(R, K'', C) \mid (R, K'', C) \models''_{me} P\} = \sqcap\{(R, K'', C) \mid (R, K'', C) \textit{ fulfills } \mathcal{H}[\![P]\!]_{me}\}$$

Proof. It suffices to prove that

$$(R, K'', C) \models''_{me} P \quad \Leftrightarrow \quad (R, K'', C) \text{ fulfills } \mathcal{H}[\![P]\!]_{me}$$
$$(R, K'', C) \models''_{me} M \quad \Leftrightarrow \quad (R, K'', C) \text{ fulfills } \mathcal{H}[\![M]\!]_{me}$$

by induction in P and M; for this we use that $[\![C]\!](l) \neq \emptyset$ is equivalent to $NC(l)$ and $[\![C]\!](l_e) \cap [\![C]\!](l_d) \neq \emptyset$ is equivalent to $NCC(l_e, l_d)$. □

Theoretical Complexity. The complexity estimate is based on Horn Clauses with Sharing [12]. For this to work we view the analysis as not quantifying wildly over sequences of variables such as $l_1, \cdots, l_k, l_e$ but only over occurrences of $enc\{\cdot^{l_1}, \cdots, \cdot^{l_k}\}_{\cdot^{l_e}}$ as they occur in the program since there are only linearly many such candidates.

Also we need to modify the axiomatization of the predicate NCC in Table 4 to avoid having quantifiers nested to depth 4. It turns out that the notion of *tiling* developed in [12] is useful for this; more specifically the axiomatization of Table 5 is obtained by applying the second variant of tiling developed in [12]. Clearly their least solutions define the same predicates NCC.

Proposition 5.2 $\sqcap\{(R, K'', C) \mid (R, K'', C)$ *fulfills* $\mathcal{H}[\![P]\!]_{me}\}$ *can be computed in cubic time.*

Proof. The clause $\mathcal{H}[\![P]\!]_{me}$ has size $\mathcal{O}(n)$ for a process of size n and the quantifiers have nesting depth at most 2. The global axiomatization of NC and NCC have size $\mathcal{O}(1)$ and nesting depth at most 3 (when axiomatized as in Table 5). For a universe of size $\mathcal{O}(n)$ the resulting constraints can be solved in time $\mathcal{O}(n^3)$ according to Proposition 2 of [12]. □

Empirical Validation. A slightly optimized version of the analysis has been implemented using our Succinct Solver [13]. To explain the main modification note that the computation of NCC is needlessly expensive in that it computes the entire relation even though typically only a small fraction of it is needed.

We deal with this using a general transformation akin to the magic set transformation for Prolog: whenever $NCC(x, y)$ is wanted in a precondition (being reachable after *pre* has succesfully been passed) we replace it by $NCC_!(x, y)$, we add the clause $pre \Rightarrow NCC_?(x, y)$, and we modify the axiomatization of NCC to yield an axiomatization of $NCC_!$ that only computes the result for values set by $NCC_?$.

We have tested the analysis on a scalable version of the Wide-Mouth-Frog communication protocol as described in [1]. The empirical measurements indicate good practical performance. In fact a system with 15 producers and 15 consumers can be analysed in about a minute.

6 Conclusion

We have shown that the versatility of Horn Clauses with Sharing suffice for obtaining an implementation of the context-independent control flow analysis [4,6] in cubic time despite the fact that the original specification operates over an infinite space of structured data values — without sacrificing any precision. Two key transformations allowed this to be accomplished: *(i)* an

$\forall l_1, l_2, u : \mathcal{C}(u) \wedge C(u, l_1) \wedge C(u, l_2) \Rightarrow NCC(l_1, l_2)$

$\forall l_1, l_2 : C(0, l_1) \wedge C(0, l_2) \Rightarrow NCC(l_1, l_2)$

$\forall l_1, l_2, suc(u_1) : C(suc(u_1), l_1) \wedge NCC'(suc(u_1), l_2) \Rightarrow NCC(l_1, l_2)$

$\forall l_2, suc(u_1), suc(u_2) : C(suc(u_2), l_2) \wedge NCC(u_1, u_2) \Rightarrow NCC'(suc(u_1), l_2)$

$\forall l_1, l_2, pair(u_{11}, u_{12}) :$
$\quad C(pair(u_{11}, u_{12}), l_1) \wedge NCC'(pair(u_{11}, u_{12}), l_2) \Rightarrow NCC(l_1, l_2)$

$\forall l_2, pair(u_{11}, u_{12}), pair(u_{21}, u_{22}) :$
$\quad C(pair(u_{21}, u_{22}), l_2) \wedge NCC(u_{11}, u_{21}) \wedge NCC(u_{12}, u_{22})$
$\quad \Rightarrow NCC'(pair(u_{11}, u_{12}), l_2)$

$\forall l_1, l_2, enc\{u_{11}, \cdots, u_{1k}\}_{u_1} :$
$\quad C(enc\{u_{11}, \cdots, u_{1k}\}_{u_1}, l_1) \wedge NCC'(enc\{u_{11}, \cdots, u_{1k}\}_{u_1}, l_2) \Rightarrow NCC(l_1, l_2)$

$\forall l_2, enc\{u_{11}, \cdots, u_{1k}\}_{u_1}, enc\{u_{21}, \cdots, u_{2k}\}_{u_2} :$
$\quad C(enc\{u_{21}, \cdots, u_{2k}\}_{u_2}, l_2) \wedge NCC(u_{11}, u_{21}) \wedge \cdots \wedge NCC(u_{1k}, u_{2k}) \wedge NCC(u_1, u_2)$
$\quad\quad \Rightarrow NCC'(enc\{u_{11}, \cdots, u_{1k}\}_{u_1}, l_2)$

Table 5
Tiled axiomatization of NCC.

explicit concretization function linking the infinite universe to a finite universe of tree-grammars (thus adapting the treatment of set-constraints to the more general logical formulae used here), and *(ii)* an explicit Galois connection encoding the apparent "relational formulation" of the polyadic constructs for communciation in terms of a cheaper "independent attribute formulation". The transformations are relevant also for other analyses of polyadic calculi operating over infinite universes.

A Proof of Proposition 3.1

$$\gamma\left(\sqcap\{(R', K', C') \mid (R', K', C') \models'_{me} P\}\right) = \sqcap\{(R, K, C) \mid (R, K, C) \models_{me} P\}$$

Proof. *Part 1: "$\sqsupseteq$" holds.* For this it suffices to show that the analysis in Table 1 is approximated by the one in Table 2 in the sense of [7]; this means that

$$(R', K', C') \models'_{me} P \quad \Rightarrow \quad (\gamma(R', K', C')) \models_{me} P$$

$$(R', K', C') \models'_{me} M \quad \Rightarrow \quad (\gamma(R', K', C')) \models_{me} M$$

and may be proved by structural induction in P and M. (We note that the converse implications do not necessarily hold due to the lack of unique representations in $DVal$ of values in Val.) We consider a few illustrative

cases and write (R, K, C) for $\gamma(R', K', C')$.

In the case of the constant zero we use that $[\![C']\!]^{\#\dagger}$ is monotonic so that $\{0\} \subseteq C'(l)$ implies that $[\![C']\!]^{\#\dagger}\{0\} \subseteq [\![C']\!]^{\#\dagger}(C'(l))$; since $[\![C']\!]^{\#\dagger}\{0\} = \{0\}$ and $[\![C']\!]^{\#\dagger}(C'(l)) = [\![C']\!](l) = C(l)$ the result follows. In the case of successor we proceed as for constants and additionally use that $[\![C']\!]^{\#\dagger}(\{suc(l_0)\}) = \textsc{suc}([\![C']\!](l_0))$.

In the case of output the reachability test $[\![C']\!](l) \neq \emptyset$ in Table 2 is equivalent to the test $C(l) \neq \emptyset$ in Table 1. Similarly in the case of matching where the reachability test $[\![C']\!](l_1) \cap [\![C']\!](l_2) \neq \emptyset$ in Table 2 is equivalent to the test $C(l_1) \cap C(l_2) \neq \emptyset$ in Table 1.

In the case of branching upon the value of an integer we use that $0 \in C'(l)$ if and only if $0 \in [\![C']\!](l)$ so that the test $0 \in C(l)$ in Table 1 can be written as $0 \in C'(l)$ also in Table 2. Similarly the test $(\exists w : suc(w) \in C(l))$ in Table 1 is equivalent to $(\exists w : suc(w) \in [\![C']\!](l))$ which is equivalent to $(\exists l_0 : suc(l_0) \in C'(l) \wedge \exists w : w \in [\![C']\!](l_0))$ and to the formulation actually used in Table 2.

Finally in the case of decryption we use that the reachability condition

$$(\exists enc\{w_1, \cdots, w_k\}_{w_e} \in [\![C']\!](l) : w_e \in [\![C']\!](l_d))$$

that could have been used in Table 2 is in fact equivalent to the condition

$$((\exists enc\{l_1, \cdots, l_k\}_{l_e} \in C'(l) : [\![C']\!](l_e) \cap [\![C']\!](l_d) \neq \emptyset \ \wedge \wedge_{i=1}^{k} [\![C']\!](l_i) \neq \emptyset)$$

actually used; similarly that the condition $w_e \in [\![C']\!](l_d)$ is in fact equivalent (given the assumptions on w_e) to the condition $[\![C']\!](l_e) \cap [\![C']\!](l_d) \neq \emptyset$ actually used.

Part 2: "$\sqsubseteq$" holds. For this we recall that it is completely standard, e.g. [11, Subsection 3.1.2], to view Tables 1 and 2 as defining monotone functions F (or $F_{me,P}$) and F' (or $F'_{me,P}$) operating over complete lattices such that

$$(R, K, C) \models_{me} P \quad \Leftrightarrow \quad F_{me,P}(R, K, C) \sqsubseteq (R, K, C)$$

and similarly for F'. Subsequently we shall write $\mathsf{lfp}(F) = (R_{\mathsf{lfp}}, K_{\mathsf{lfp}}, C_{\mathsf{lfp}})$ and $\mathsf{lfp}(F') = (R'_{\mathsf{lfp}}, K'_{\mathsf{lfp}}, C'_{\mathsf{lfp}})$. From Tarski's Theorem it follows that the *desired* result amounts to

$$\gamma(R'_{\mathsf{lfp}}, K'_{\mathsf{lfp}}, C'_{\mathsf{lfp}}) \sqsubseteq (R_{\mathsf{lfp}}, K_{\mathsf{lfp}}, C_{\mathsf{lfp}}).$$

Restricting [6] the sets $\mathcal{B}$, $\mathcal{C}$ and $\mathcal{L}$ to be finite sets containing all relevant entities in P and me and restricting the arity of arguments to enc to the maximal one found in P it is clear that F' operates over a complete lattice of finite size. It follows that $(R'_{\mathsf{lfp}}, K'_{\mathsf{lfp}}, C'_{\mathsf{lfp}}) = F'^{n}(\bot, \bot, \bot)$ for some natural number n. Writing

$$\gamma_{\mathsf{lfp}}(R', K', C') = (C^{\#\dagger}_{\mathsf{lfp}} \circ R', \ C^{\overline{\#}\dagger}_{\mathsf{lfp}} \circ K', \ C^{\#\dagger}_{\mathsf{lfp}} \circ C')$$

[6] Otherwise we could establish continuity and resort to transfinite induction.

we then show that $\gamma_{\mathsf{lfp}}(F'^n(\bot,\bot,\bot)) \sqsubseteq (R_{\mathsf{lfp}}, K_{\mathsf{lfp}}, C_{\mathsf{lfp}})$ by induction in n.

The base case is immediate since $\gamma_{\mathsf{lfp}}(\bot,\bot,\bot) = (\bot,\bot,\bot)$. For the inductive step it suffices to assume that $\gamma_{\mathsf{lfp}}(R',K',C') \sqsubseteq (R_{\mathsf{lfp}}, K_{\mathsf{lfp}}, C_{\mathsf{lfp}})$ and to show that $\gamma_{\mathsf{lfp}}(R'',K'',C'') \sqsubseteq (R_{\mathsf{lfp}}, K_{\mathsf{lfp}}, C_{\mathsf{lfp}})$ where $(R'',K'',C'') = F'(R,K,C)$; this amounts to a straightforward structural induction on P (with an auxiliary induction on M) using that $(R_{\mathsf{lfp}}, K_{\mathsf{lfp}}, C_{\mathsf{lfp}})$ is a fixed point of F. We have now established the *intermediate* result that

$$\gamma_{\mathsf{lfp}}(R'_{\mathsf{lfp}}, K'_{\mathsf{lfp}}, C'_{\mathsf{lfp}}) \sqsubseteq (R_{\mathsf{lfp}}, K_{\mathsf{lfp}}, C_{\mathsf{lfp}}).$$

To obtain the desired result, with γ_{lfp} replaced by γ, we define an operator $\langle\cdots\rangle_k$ for "truncating" functions producing sets of elements of Val to include only elements of depth at most k. It then suffices to prove

$$\forall k : (\langle[\![C'_{\mathsf{lfp}}]\!]^{\#\dagger}\circ R'_{\mathsf{lfp}}\rangle_k, \langle[\![C'_{\mathsf{lfp}}]\!]^{\overline{\#}\dagger}\circ K'_{\mathsf{lfp}}\rangle_k, \langle[\![C'_{\mathsf{lfp}}]\!]^{\#\dagger}\circ C'_{\mathsf{lfp}}\rangle_k) \sqsubseteq (\langle R_{\mathsf{lfp}}\rangle_k, \langle K_{\mathsf{lfp}}\rangle_k, \langle C_{\mathsf{lfp}}\rangle_k)$$

because all structures in Val are finite.

We proceed by induction in k. The base case has $k=1$ and is immediate because $\langle[\![C'_{\mathsf{lfp}}]\!]^{\#\dagger} \circ C'_{\mathsf{lfp}}\rangle_1 = \langle C_{\mathsf{lfp}}^{\#\dagger} \circ C'_{\mathsf{lfp}}\rangle_1 \sqsubseteq \langle C_{\mathsf{lfp}}\rangle_1$ using the intermediate result, and similarly for the other two components. For the inductive step we calculate:

$$\begin{aligned}
\langle[\![C'_{\mathsf{lfp}}]\!]^{\#\dagger} \circ C'_{\mathsf{lfp}}\rangle_{k+1} &= \langle[\![C'_{\mathsf{lfp}}]\!]\rangle_k^{\#\dagger} \circ C'_{\mathsf{lfp}} && \text{(property of } \langle\cdots\rangle_{k+1}) \\
&= \langle[\![C'_{\mathsf{lfp}}]\!]^{\#\dagger} \circ C'_{\mathsf{lfp}}\rangle_k^{\#\dagger} \circ C'_{\mathsf{lfp}} && \text{(definition of } [\![\cdots]\!]) \\
&\sqsubseteq \langle C_{\mathsf{lfp}}\rangle_k^{\#\dagger} \circ C'_{\mathsf{lfp}} && \text{(induction hypothesis)} \\
&= \langle C_{\mathsf{lfp}}^{\#\dagger} \circ C'_{\mathsf{lfp}}\rangle_{k+1} && \text{(property of } \langle\cdots\rangle_{k+1}) \\
&\sqsubseteq \langle C_{\mathsf{lfp}}\rangle_{k+1} && \text{(the intermediate result)}
\end{aligned}$$

and similarly for the other two components; this completes the proof. □

References

[1] M. Abadi. Secrecy by typing in security protocols. *Journal of the ACM*, 46:749–786, 1999. A preliminary version appeared in *Proceedings of Theoretical Aspects of Computer Software*, *LNCS*, Springer-Verlag, 1997.

[2] M. Abadi and A. D. Gordon. A calculus for cryptographic protocols - The Spi calculus. *Information and Computation*, 148:1–70, January 1999.

[3] A. Aiken. Introduction to set constraint-based program analysis. *Science of Computer Programming (SCP)*, 35(2):79–111, 1999.

[4] C. Bodei. *Security Issues in Process Calculi.* PhD thesis, Department of Computer Science, University of Pisa, 2000.

[5] C. Bodei, P. Degano, F. Nielson, and H. Riis Nielson. Static analysis for the π-calculus with applications to security. *Information and Computation*, 165:68–92, 2001.

[6] C. Bodei, P. Degano, H. Riis Nielson, and F. Nielson. Static analysis for secrecy and non-interference in networks of processes. In *Proc. PACT'01*, number 2127 in Lecture Notes in Computer Science, pages 27–41. Springer-Verlag, 2001.

[7] R. R. Hansen, J. G. Jensen, F. Nielson, and H. Riis Nielson. Abstract Interpretation of Mobile Ambients. In *Proceedings of SAS'99*, volume 1694 of *LNCS*, pages 134–148. Springer-Verlag, 1999.

[8] D. Lugiez and Ph. Schnoebelen. The regular viewpoint on pa-processes. In *9th Int. Conference on Concurrency (CONCUR)*, pages 50–66. LNCS 1466, Springer Verlag, 1998.

[9] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes (I and II). *Information and Computation*, 100(1):1–77, 1992.

[10] F. Nielson and H. Riis Nielson. Flow logics and operational semantics. *Electronic Notes of Theoretical Computer Science*, 10, 1998.

[11] F. Nielson, H. Riis Nielson, and C. L. Hankin. *Principles of Program Analysis*. Springer, 1999.

[12] F. Nielson and H. Seidl. Control flow analysis in cubic time. In *Proceedings of ESOP'01*, volume 2028 of *LNCS*, pages 252–268. Springer-Verlag, 2001.

[13] F. Nielson and H. Seidl. Succinct solvers. Technical Report 01-12, University of Trier, Germany, 2001.