Protocol Dialects as Formal Patterns

Víctor García José Meseguer
Enrique Gallifa Santiago Escobar Catherine Meadows

Outline

1. Lingos
   - What is a Lingo?
   - f-Checkable Lingos
   - Comp-Malleable Lingos
   - Horizontal Composition
   - Functional Composition
   - Other Ongoing Work
2. Dialects
   - Protocols as Rewrite Theories
   - What is a Dialect?
   - Vertical Composition
   - Time Dependent Dialects
3. Maude Library of Parameterized Modules and Examples

What is a Lingo?

A lingo Λ is a data transformation between data types D_1 and D_2, with a one-sided inverse. The transformation is parametric on a parameter value a belonging to a parameter set A. For each parameter a, data from D_1 is transformed into data of D_2, which, using the same parameter a, can be transformed back into the original data from D_1.

We can formalize a *lingo* Λ as a 6-tuple $\Lambda = (D_1, D_2, A, f, g, comp)$, where

$$
\begin{aligned}
& D_1, D_2 \text{ and } A \text{ are non-empty sets,} \\
& f : D_1 \times A \to D_2, \\
& g : D_2 \times A \to D_1 \\
& \text{such that } \forall d_1 \in D_1, \forall a \in A, \\
& \qquad\qquad g(f(d_1, a), a) = d_1, \\
& comp : D_2 \times A \to Bool \text{ such that } \forall d_1 \in D_1, \forall a \in A, comp(f(d_1, a), a) = true.
\end{aligned}
$$

What are the Desirable Properties of a Lingo?

The purpose of a lingo $\Lambda = (D_1, D_2, A, f, g, comp)$, is to obfuscate the communication between a sender, *Alice*, and a receiver, *Bob*, who use a common, dynamically changing, **secret parameter** $a \in A$. To make it hard for an intruder to interfere with their communication it would be desirable that, for example,

1. The probability that the attacker can guess the secret parameter a before it is used is low.
2. Non-compliant messages from an attacker, i.e., messages that do not use the current secret parameter a, can be detected and discarded (Λ is f-**checkable**).
3. The attacker cannot modifiy a legitimate message to make it compliant with the current secret parameter a (Λ is not **compliance-malleable**).
4. It is hard for an attacker to impersonate a legitimate protocol participant.

We have been developing *compositional methods* to build lingos enjoying these and other desirable properties.

f-Checkable Lingos

Definition. A lingo $\Lambda = (D_1, D_2, A, f, g, comp)$ is *f-checkable* iff: (i) $\exists d_2 \in D_2,\ \exists a \in A,\ \ s.t.\ \nexists d_1 \in D_1,\ (d_2 = f(d_1, a))$; and (ii) $\forall d_2 \in D_2, \forall a \in A,\ comp(d_2, a) = true \Leftrightarrow \exists d_1 \in D_1,\ (d_2 = f(d_1, a))$

Example 1 (Lingo D&C)

$\Lambda_{d\&c} = (\mathbb{N}, \mathbb{N} \times \mathbb{N}, \mathbb{N}, f, g, comp)$, with:

- $f(n, a) = (quot(n + (a + 2), a + 2), rem(n + (a + 2), a + 2))$
- $g((x, y), a) = (x \cdot (a + 2)) + y - (a + 2)$
- $comp((x, y), a) = x \equiv quot((x \cdot (a + 2)) + y, a + 2))\ and\ y \equiv rem((x \cdot (a + 2)) + y, a + 2))$

where $\equiv$ is the equality predicate on $\mathbb{N}$.

Theorem. There is an automatic transformation $\Lambda \mapsto \Lambda^{\sharp}$ transforming any lingo Λ into an f-checkable lingo.

Comp-Malleable Lingos

Call a lingo Λ *comp-malleable* if an intruder can disrupt the communication between an honest sender *Alice* and an honest receiver *Bob* in a protocol $\mathcal{P}$ whose messages are obfuscated with lingo Λ by producing a *compliant* message supposedly sent from *Alice* to *Bob* using a secret parameter $a \in A$ of Λ, but actually sent by the intruder.

Definition. Λ is called *comp-malleable* iff there exists a Σ_Λ-term $t(x, y)$ of sort D_2 with free variables x, y of respective sorts D_2 and A and a computable function $r : A \to A$ such that $\forall d_1 \in D_1,\ \forall a, a' \in A$,

1. $f(d_1, a) \neq t(f(d_1, a), r(a'))$
2. $comp(t(f(d_1, a), r(a')), a) = true$.

$t(x, y)$ is a *recipe* that an intruder can use to replace $f(d_1, a)$ by $t(f(d_1, a), r(a'))$ to pass the compliance test $comp(t(f(d_1, a), r(a')), a)$. The function r ensures that the randomly chosen parameter $r(a')$ meets condition (1).

Examples of Comp-Malleable Lingos

Example 2 (Lingo XOR)

Let $\Lambda_{xor} = (\{0,1\}^n, \{0,1\}^n, \{0,1\}^n, \oplus, \oplus, \mathit{true})$, with $\oplus : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ bitwise exclusive or and $\mathit{true} : \{0,1\}^n \times \{0,1\}^n \to \mathit{Bool}$ the always true predicate.

Λ_{xor} is a comp-malleable lingo. The recipe $t(x,y)$ is $x \oplus y$ and r_{oplus} is the computable function $r_{oplus} = \lambda y \in \{0,1\}^n.\ \textbf{if}\ y. = .\vec{0}\ \textbf{then}\ \vec{1}\ \textbf{else}\ y\ \textbf{fi} \in \{0,1\}^n$.

There are f-checkable lingos that are comp-malleable.

Example 3 (Lingo XOR#)

The f-checkable lingo $\Lambda^\sharp_{xor}$ obtained from Λ_{xor} by applying the $\Lambda \mapsto \Lambda^\sharp$ transformation is comp-malleable. with recipe $t(x,y) = [p_1(x) \oplus p_1(y), p_2(x) \oplus p_1(y)]$ and suitable r.

Horizontal Composition

The horizontal composition of two lingos with the same D_1 and D_2 is a lingo that behaves as the first lingo or as the second one depending on the result of a random throw of a biased coin. The coin bias β should be chosen so as to favor the lingo that is deemed stronger.

Definition. Given lingos $\Lambda = (D_1, D_2, A, f, g)$ and $\Lambda' = (D_1, D'_2, A', f', g')$, a bias $\beta \in (0, 1)$, functions *param* $: \mathbb{N} \to A$ and *param*$' : \mathbb{N} \to A'$ from Λ and Λ'' respectively, and a pseudo-random function $throw_\beta : \mathbb{N} \to \{0, 1\}$ simulating a sequence of throws of a coin with bias β, their *horizontal composition with bias* β is the lingo $\Lambda \oplus_\beta \Lambda' =_{def} (D_1, D_2 \cup D'_2, A \cup A', f \oplus_\beta f', g \oplus_\beta g')$, where for each $d_1 \in D_1$, $d_2 \in D_2 \cup D'_2$ and $n \in \mathbb{N}$,

- $f \oplus_\beta f'(d_1, n) =_{def}$ **if** $throw_\beta(n) = 1$ **then** $f(d_1, param(n))$ **else** $f'(d_1, param'(n))$ **fi**
- $g \oplus_\beta g'(d_2, n) =_{def}$ **if** $throw_\beta(n) = 1$ **then** $g(d_2, param(n))$ **else** $g'(d_2, param'(n))$ **fi**
- $comp \oplus_\beta comp'(d_2, n) =_{def}$ **if** $throw_\beta(n) =$ 1 **then** $comp(d_2, param(n))$ **else** $comp'(d_2, param'(n))$ **fi**

Example of Horizontal Composition

Example 4 (Horizontal composition of XOR and D&C)

Let Λ_{xor} be the exclusive or lingo from Example 2. Let $\Lambda_{D\&C}$ be the divide and check lingo from Example 1. Then, the horizontal composition of Λ_{xor} and $\Lambda_{D\&C}$ is the lingo $\Lambda_{xor} \oplus_{\beta} \Lambda_{D\&C} = (\{0,1\}^n, \{0,1\}^n \cup (\{0,1\}^n \times \{0,1\}^n), \{0,1\}^n \cup (\{0,1\}^n \times \{0,1\}^n), xor \oplus_{\beta} divide, xor \oplus_{\beta} check, true_{\Lambda_{xor}} \oplus_{\beta} comp_{\Lambda_{d\&c}})$.

Functional Composition

The functional composition $\Lambda \odot \Lambda'$ of lingos Λ and Λ', first alters a message with Λ, and then further alters the result with Λ'. This lingo composition can greatly increases the choice of parameters and can make life much harder for attackers. It seems a promising method to construct non-comp-malleable lingos.

Definition. Given lingos $\Lambda = (D_1, D_2, A, f, g, comp)$ and $\Lambda' = (D_2, D_3, A', f', g', comp')$, their *functional composition* is the lingo,

$\Lambda \odot \Lambda' =_{def} (D_1, D_3, A \times A', f \cdot f', g * g', comp \odot comp')$, where for each $d_1 \in D_1$, $d_3 \in D_3$, and $(a, a') \in A \times A'$,

- $f \cdot f'(d_1, (a, a')) =_{def} f'(f(d_1, a), a')$
- $g * g'(d_3, (a, a')) =_{def} g(g'(d_3, a'), a)$
- $comp \odot comp'(d_3, (a, a')) =_{def} comp'(d_3, a') \text{ and } comp(g'(d_3, a'), a)$

Functional Composition

Example 5 (Functional Composition of XOR and D&C)

Consider lingos Λ_{xor} and $\Lambda_{d\&c}$, given in Examples 2 and 1 respectively. We can functionally composed them as $\Lambda_{xor \odot d\&c}$ where for each $d_1 \in \{0,1\}^n$, $d_3 \in \{0,1\}^n \times \{0,1\}^n$, and $(a, a') \in \{0,1\}^n \times \{0,1\}^n$,

- $xor \cdot divide(d_1, (a, a')) = divide(xor(d_1, a), a')$
- $xor * check(d_3, (a, a')) = xor(check(d_3, a'), a)$
- $true_{\Lambda_{xor}} \odot true_{\Lambda_{d\&c}}(d_3, (a, a')) = true_{\Lambda_{d\&c}}(d_3, a') \text{ and } true_{\Lambda_{xor}}(g'(d_3, a'), a)$

Other Ongoing Work

The main purpose of lingos is to provide a *compositional framework* to design increasingly harder break obfuscation mechanism. Besides the methods described above, other ongoing work on lingos includes:

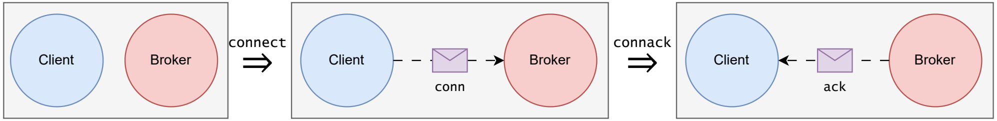
1. **Data Adaptors**, to reuse a lingo Λ with different input and/or output payload formats.
2. **Additional Lingo Composition Operators**, such as **product** $\Lambda \times \Lambda'$ and **tupling** $[\Lambda, \Lambda']$ of lingos, to further harden lingos.
3. **Authenticating Lingos**, to preclude attackers from impersonating existing honest participants.

Outline

1. Lingos
   - What is a Lingo?
   - f-Checkable Lingos
   - Comp-Malleable Lingos
   - Horizontal Composition
   - Functional Composition
   - Other Ongoing Work
2. Dialects
   - Protocols as Rewrite Theories
   - What is a Dialect?
   - Vertical Composition
   - Time Dependent Dialects
3. Maude Library of Parameterized Modules and Examples

Protocols as Rewrite Theories

A protocol $\mathcal{P}$ is modeled as a collection of objects communicating by message passing. Message send and receive actions are specified by rewrite rules.

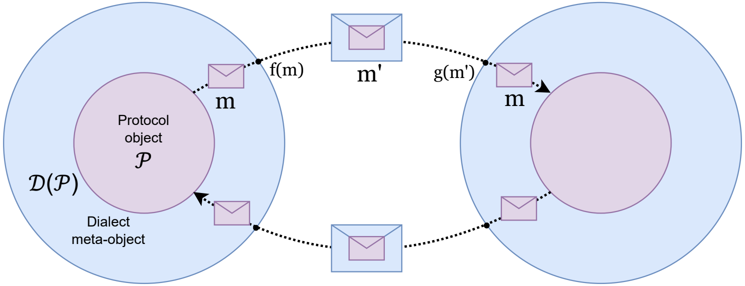


$\texttt{rl [connect] :}$
$\quad < C : \mathit{Client} \mid \mathit{brk} : B,\ \mathit{Atts} >$
$\texttt{=>} < C : \mathit{Client} \mid \mathit{brk} : B,\ \mathit{Atts} >$
$\quad (\mathit{from}\ C\ \mathit{to}\ B\ :\ \mathit{conn})\ .$

$\texttt{rl [connack] :}$
$\quad < B : \mathit{Broker} \mid \mathit{peer} : \mathit{none},\ \mathit{Atts} >$
$\quad (\mathit{from}\ C\ \mathit{to}\ B\ :\ \mathit{conn})$
$\texttt{=>} < B : \mathit{Broker} \mid \mathit{peer} : C,\ \mathit{Atts} >$
$\quad (\mathit{from}\ B\ \mathit{to}\ C\ :\ \mathit{ack})\ .$

What is a Dialect?

Dialects are protocol transformations of the form $\mathcal{P}(D) \mapsto \mathcal{D}(\Lambda(D), \mathcal{P}(D))$, where $\mathcal{D}(\Lambda(D), \mathcal{P}(D))$, is a protocol that wraps each object of $\mathcal{P}(D)$ inside a *dialect meta-actor* that uses lingo $\Lambda(D)$ to obfuscate the communication between the honest protocol actors of $\mathcal{P}(D)$.



What is a Dialect?

Meta-Object actions are specified by rewrite rules and *protocol-generic*. Thus, the meta-object rules of a Dialect can be applied to a wide range of underlying protocos, including to protocols transformed by other dialects. The rule below shows how the meta-object *obfuscates* a message before being sent.

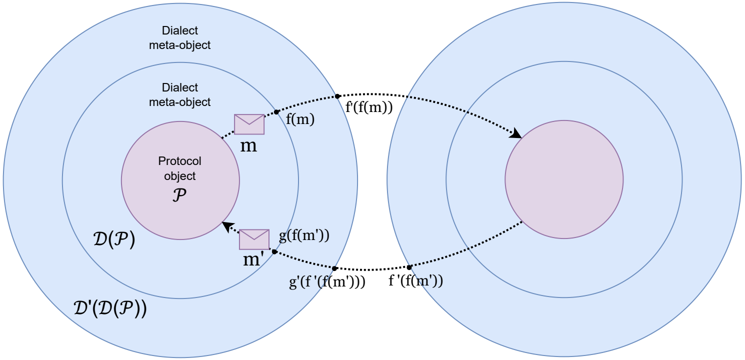
$$\begin{aligned}
\texttt{rl [out]} : < O_1 : DC \mid \texttt{conf} : (< O_1 : C \mid atts > \ (from\ O_1\ to\ O_2\ :\ P) \cup\ M), \\
\texttt{peer-counters} : R,\ atts' > \\
\texttt{=>}\ < O_1 : DC \mid \texttt{conf} : (< O_1 : C \mid atts > \ M \setminus (from\ O_1\ to\ O_2\ :\ P)), \\
\mathrm{update}(\texttt{peer-counters} : R, atts') > \\
(from\ O_1\ to\ O_2\ :\ f(P, param(R[O_2])))\ .
\end{aligned}$$

Note that *the underlying protocol is not changed at all*, making dialects highly modular.

Dialects

Vertical Composition

For $\mathcal{D}(\Lambda, \mathcal{P})$ and $\mathcal{D}'(\Lambda', \mathcal{P}')$ two dialects, their vertical composition, denoted $\mathcal{D}'(\Lambda', \mathcal{P}') \circ \mathcal{D}(\Lambda, \mathcal{P})$, is a new dialect of the form $\mathcal{D}'(\Lambda', \mathcal{D}(\Lambda, \mathcal{P}))$.

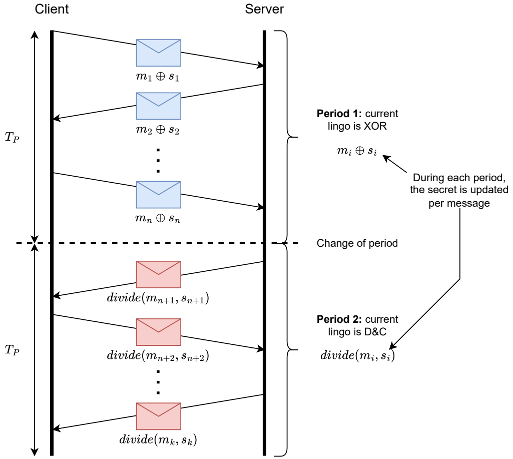


Dialect meta-object
Dialect meta-object
Protocol object $\mathcal{P}$
m
f(m)
f'(f(m))
g(f(m'))
m'
g'(f '(f(m')))
f '(f(m'))
$\mathcal{D}(\mathcal{P})$
$\mathcal{D}'(\mathcal{D}(\mathcal{P}))$

Note the close relationship with the functional lingo composition $\Lambda \odot \Lambda'$.

Time Dependent Dialects

A dialect is a moving-target because its parameters change all the time. Faster moving targets can be achieved by means of *periodic* dialects, where the lingo being used changes *periodically* and *unpredictably*.


Client
Server
T_P
$m_1 \oplus s_1$
$m_2 \oplus s_2$
$m_n \oplus s_n$
Period 1: current lingo is XOR
$m_i \oplus s_i$
During each period, the secret is updated per message
Change of period
T_P
$divide(m_{n+1}, s_{n+1})$
$divide(m_{n+2}, s_{n+2})$
$divide(m_k, s_k)$
Period 2: current lingo is D&C
$divide(m_i, s_i)$

Outline

1. Lingos
   What is a Lingo?
   f-Checkable Lingos
   Comp-Malleable Lingos
   Horizontal Composition
   Functional Composition
   Other Ongoing Work

2. Dialects
   Protocols as Rewrite Theories
   What is a Dialect?
   Vertical Composition
   Time Dependent Dialects

3. Maude Library of Parameterized Modules and Examples

Maude Library of Parameterized Modules and Examples

All the examples used for this presentation are specified in Maude and are available at the project's GitHub repository (waiting for a pub-release from NRL).

The following slides contain an overview of all the components of the Dialects as Formal Patterns specified in Maude as parameterized (i.e., generic) modules, as well as many examples instantiating them.

For example, a *dialect* is a protocol- and lingo-generic parameterized module that can be applied to a wide variety of protocols whose communication can be altered by a wide variety of lingos.

The examples then illustrate how parameterized lingos and parameterized modules can be instantiated in a variety of concrete instances.

Lingos

- Parameterized Lingos
  - Identity
  - Split
  - XOR
  - Divide and check (D&C)
- Lingo Composition Operations and Examples
  - Horizontal Composition of Lingos
    - sin-dilingo-mqtt-biased
    - sin-dilingo-mqtt-fair
    - sin-dilingo'-mqtt-fair
    - sin-trilingo-nspk-fair
  - Functional Composition of Lingos
    - sin-xor&split-nspk
    - sin-xor&xor'&split-nspk
    - sin-xor&dc-mqtt
  - Data Adaptors
    - sin-laxor-mqtt

Dialects

- Parameterized Singleton Dialect Examples
  - sin-id-mqtt
  - sin-xor-nspk
  - sin-xor-mqtt
  - sin-edxor-mqtt
  - sin-dc-mqtt
  - sin-split-nspk
  - sin-xor2K-mqtt
- Parameterized Periodic Dialect Examples
  - per-xor-nspk
  - per-xor-mqtt
  - per-dilingo-mqtt
- Examples of Vertical Composition of Dialects
  - sin-xor1+sin-xor2-mqtt
  - sin-xor1+sin-xor2-nspk
  - sin-xor+sin-split-nspk
  - sin-xor+sin-dc-mqtt
  - sin-dilingo-fair+sin-edxor-mqtt (vertical composition of an horizontal composition)