Ping-Pong Protocols and Turchin Relation

Antonina Nepeivoda
Program Systems Institute of RAS

Pereslavl–Zalessky

Introduction

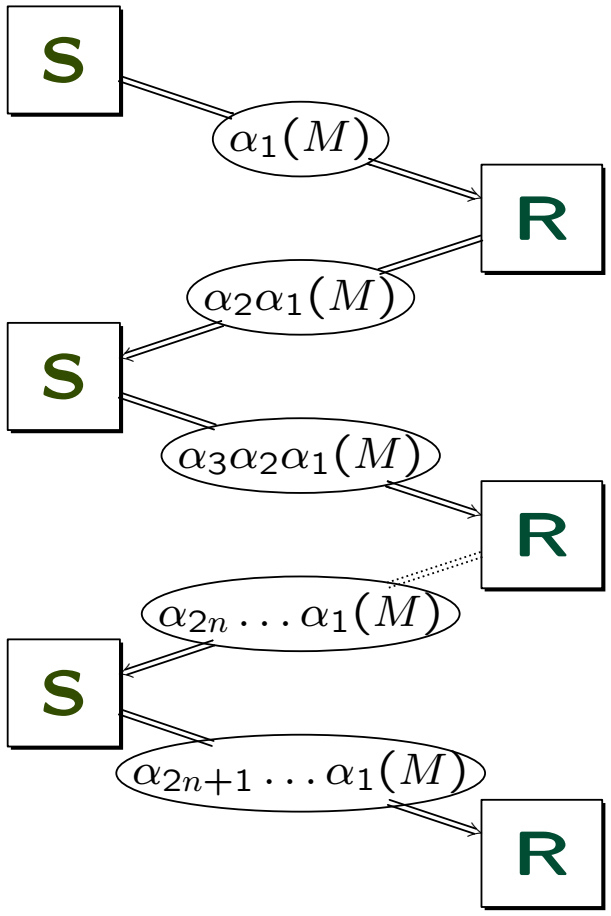
The talk is organized as follows:

1. Introduction of the class of protocols and intruder behavior model.
2. Modeling the protocols in terms of prefix rewriting grammars.
3. Testing classical termination criteria on being capable to solve the verification task.
4. Refining the Turchin relation and showing that pure unfolding with the refined relation always solves the verification task.

The approach of verification via unfolding is not new

1. Program verification via fold/unfold techniques (Pettorossi, Proietti)
2. Cryptographic protocol verification via supercompilation (Nemytykh, Lisitsa)
3. Using prefix grammars in the task of concurrent protocol verification (Delzanno, Esparza, Srba)

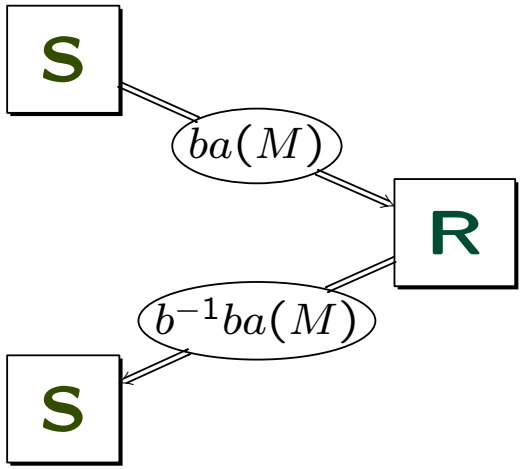
Consider an interaction of two legal participants $\mathbf{S}$ and $\mathbf{R}$ of sending pieces of data in course, as while playing ping-pong, by an open channel. The initial message is denoted as M.



(Dolev, Yao) A ping-pong protocol $\mathbf{P(S,R)}$ ($\mathbf{S}$ and $\mathbf{R}$ are legal participants) — a tuple $\Gamma = \langle \alpha_1, \ldots, \alpha_n \rangle$, where $\alpha_i \in \Sigma^*_{\mathbf{S}}$, iff i is odd, and $\alpha_i \in \Sigma^*_{\mathbf{R}}$, iff i is even.

Λ is the empty word. $x_1 x_2 \ldots x_n \to \Lambda$ denotes the fact of intercancellation of the sequence $x_1 x_2 \ldots x_n$ (the word collapses to Λ).

Let $\mathbf{P_a(S,R)} = \langle ba, b^{-1} \rangle$, where $bb^{-1} \to \Lambda$ and $b^{-1}b \to \Lambda$.



Dolev–Yao Intruder Model

An intruder $\mathbf{Z}$ can tackle any message x sent from $\mathbf{S}$ to $\mathbf{R}$ or vice versa and replace it by $\beta(x)$, $\beta \in \Sigma_{\mathbf{Z}}$ (β is a single action). A protocol $\mathbf{P}(\mathbf{S}, \mathbf{R})$ is insecure iff there exists such a sequence of intruder actions that the intruder can get M. An intruder can initiate multiple interactions by $\mathbf{P}(\mathbf{S}, \mathbf{R})$ (playing a role of $\mathbf{S}$ or $\mathbf{R}$ in each of them).

But what if $\Sigma_{\mathbf{Z}}$ contains some actions indivisible from the intruder's point of view, though composite from the point of view of some principal?

Example

Suppose a user is allowed to append their name to an encrypted message M (optionally). Then the action

M_1	M_2	...	M_n	+	A	N	T	O	N	I	N	A

is available to the user, but the subactions

M_1	M_2	...	M_n	+	A	N	T	O	N

and

M_1	M_2	...	M_n	+	N	I	N	A

are forbidden.

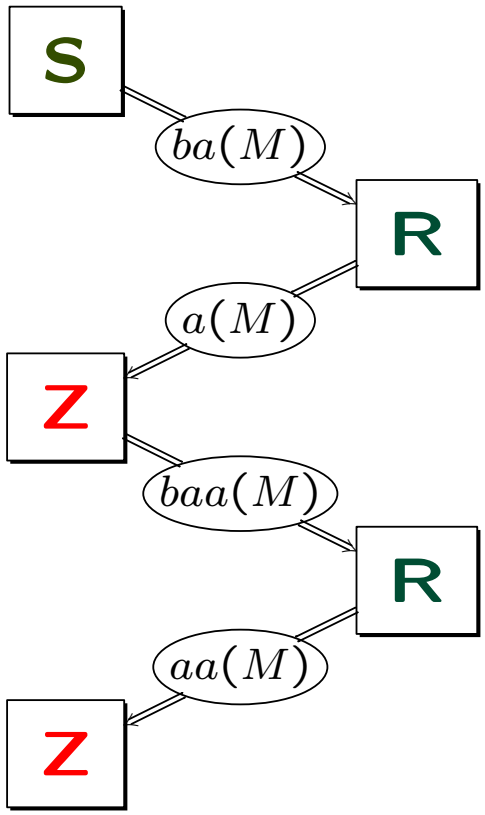
Consider the protocol $\mathbf{P_a(S,R)}$ with $\Sigma_{\mathbf{Z}} = \{c, d\}$.

Let $d = ba$ and $c = a^{-1}a^{-1}$ such that $a^{-1}a \to \Lambda$, but the single action a^{-1} is unavailable to $\mathbf{Z}$. Note that in the Dolev–Yao intruder model $a^{-1}a^{-1}$ cannot appear in $\Sigma_{\mathbf{Z}}$, since it is a word, not a single letter.

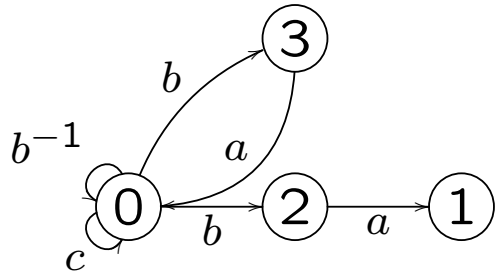
This protocol is insecure. An example of an attack is below:

From S to R:	$ba(M)$	
From R to S:	$b^{-1}ba(M)$	tackled by Z
From Z to R:	$ba b^{-1}ba(M)$	
From R to Z:	$b^{-1}ba b^{-1}ba(M)$	

At last the intruder applies c and gets the initial M unencrypted.


S
$ba(M)$
R
$a(M)$
Z
$baa(M)$
R
$aa(M)$
Z

From the intruder alphabet $\Sigma_{\mathbf{Z}}$ and $\mathbf{P_a(S,R)}$ the following finite automaton $FA_{PA} = \langle S_{FA}, \Sigma_{\mathbf{R}} \cup \Sigma_{\mathbf{S}} \cup \Sigma_{\mathbf{Z}}, R, 0, 1 \rangle$ is constructed.



Every action α_i from $\mathbf{P_a(S,R)}$ corresponds to a loop from the state 0 to itself. Every action from $\Sigma_{\mathbf{Z}}$ corresponds to a self-loop from 0 to 0. The action $\alpha_1 \in \mathbf{P_a(S,R)}$ ($\alpha_1 = ba$) corresponds to a path from the initial state 0 to the accepting state 1.

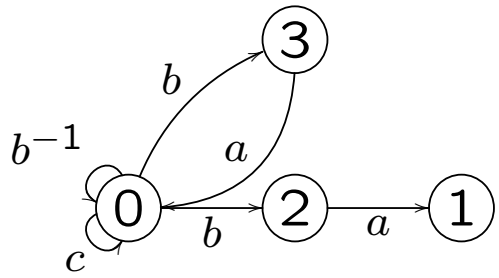
The cancellation rules are

$$b^{-1}b \to \Lambda \qquad bb^{-1} \to \Lambda \qquad caa \to \Lambda$$

Dolev–Even–Karp verification procedure uses only the automaton model and the cancellation rules to find whether an attack exists.

The verification procedure consists of the following steps:

1. $C \leftarrow \{\langle i, i\rangle\}$, $i \in S_{FA}$. $Q \leftarrow C$.
2. While $Q \neq \emptyset$ do
   (a) $Q \leftarrow Q \setminus \{\langle i, j\rangle\}$.
   (b) For all k if $\langle j, k\rangle \in C$, $\langle i, k\rangle \notin C$, then $C \leftarrow C \cup \{\langle i, k\rangle\}$, $Q \leftarrow Q \cup \{\langle i, k\rangle\}$.
   (c) For all k if $\langle k, i\rangle \in C$, $\langle k, j\rangle \notin C$, then $C \leftarrow C \cup \{\langle k, j\rangle\}$, $Q \leftarrow Q \cup \{\langle k, j\rangle\}$.
   (d) For all k, l if $k \rightarrow^{\tau} i \in R$ and $j \rightarrow^{\sigma} l \in R$, $\tau\sigma \rightarrow \Lambda$ and $\langle k, l\rangle \notin C$, then $C \leftarrow C \cup \{\langle k, l\rangle\}$, $Q \leftarrow Q \cup \{\langle k, l\rangle\}$.

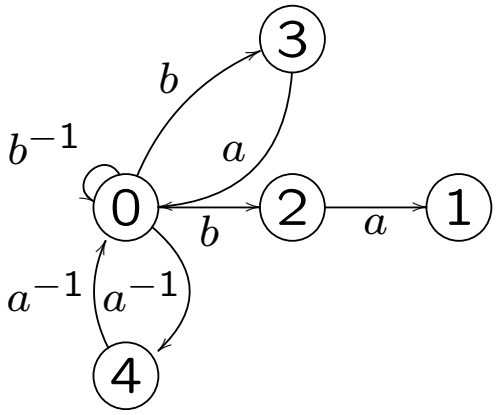


The attack on $\mathbf{P_a}$ cannot be found since there are no rules in the algorithm that operate with triple cancellations.

Extended Dolev–Yao Intruder Model

Let us define $\Sigma_{\mathbf{X}}$ as follows. Elementary actions from $\Sigma_{\mathbf{X}}$ (which are indivisible for everyone) are denoted by single letters. Actions that are indivisible only for $\mathbf{X}$ but can be decomposed to elementary steps by someone else are denoted by words.

Now in $\mathbf{P_a(S,R)}$ model $\Sigma_{\mathbf{Z}}$ can look like $\{a^{-1}a^{-1}, ba\}$, s.t. $a^{-1}a \to \Lambda$.



Now the attack on $\mathbf{P_a}$ is found by the Dolev–Even–Karp procedure.

A tuple $\langle \Sigma, \mathbf{R}, \Gamma_0 \rangle$, where Σ is an alphabet, $\Gamma_0 \in \Sigma^+$ is the initial word and $\mathbf{R}$ is the set of rewrite rules, is *a prefix grammar (PG)*, iff any rule $R : R_l \to R_r$ is applicable only to words of the form $R_l\Phi$ and generates the words $R_r\Phi$.

Example

If the rule $a \to bb$ is applied to $aabab$ in PG, then the result is neither $abbbab$ nor $aabbbb$ but $bbabab$.

A PG hierarchy (Caucal):

Grammar type	Rule form
Type 0	$\Phi \to^\sigma \Psi$
Type $1\frac{1}{2}$	$pa \to^\sigma q\Psi$
Type 2	$a \to^\sigma \Psi$
Type 3	$a \to^\sigma b \vee a \to^\sigma \Lambda$

Ping-pong protocols can be modeled in terms of PG but the question of modeling must be solved carefully.

Example

The naive PG representation of $\mathbf{P_a(S,R)}$ is incorrect.

$\mathbf{G_{PA}}$:

$R_1 : \Lambda \to a^{-1}a^{-1}$ $\quad$ $R_3 : bb^{-1} \to \Lambda$ $\quad$ $R_5 : \Lambda \to ba$

$R_2 : a^{-1}a \to \Lambda$ $\quad$ $R_4 : b^{-1}b \to \Lambda$ $\quad$ $R_6 : \Lambda \to b^{-1}$

$a^{-1}a^{-1}aa$ cannot be transformed to Λ.

So the ability to apply a cancellation as early as possible must be retained.

Example

$\mathbf{P_a(S, R)}$ can be represented as

$\mathbf{G_{PA'}}$:

$R_1 : [0]x \to [0]bax$
$R_2 : [0]x \to [0]a^{-1}a^{-1}x$
$R_3 : [0]x \to [0]b^{-1}x$
$R_4 : [0]b \to [0]$
$R_5 : [0]a \to [1]$
$R_6 : [1]x \to [0]a^{-1}x$
$R_7 : [1]a \to [0]$

or as

$\mathbf{G_{PA0}}$:

$R_1 : \Lambda \to ba$
$R_2 : \Lambda \to a^{-1}a^{-1}$
$R_3 : \Lambda \to b^{-1}$
$R_4 : b \to \Lambda$
$R_5 : a \to a^{-1}$
$R_6 : aa \to \Lambda$

Consider the full tree of derivations from the initial word Γ_0 by rules $\mathbf{R}$ of a PG. This tree contains branches that correspond to protocol attacks. Such branches start from Γ_0 and end with Λ.

The problem: in general the tree is infinite.

If an attack exists, it will be found. But if not, then the unfolding must be halted when it becomes clear that there is no possibility to find a branch ending with Λ in the tree. The only question remains what is a halting criterion.

A similar problem is solved in program transformation theory.

A semantic tree of a program — a tree of program runs on all possible data.

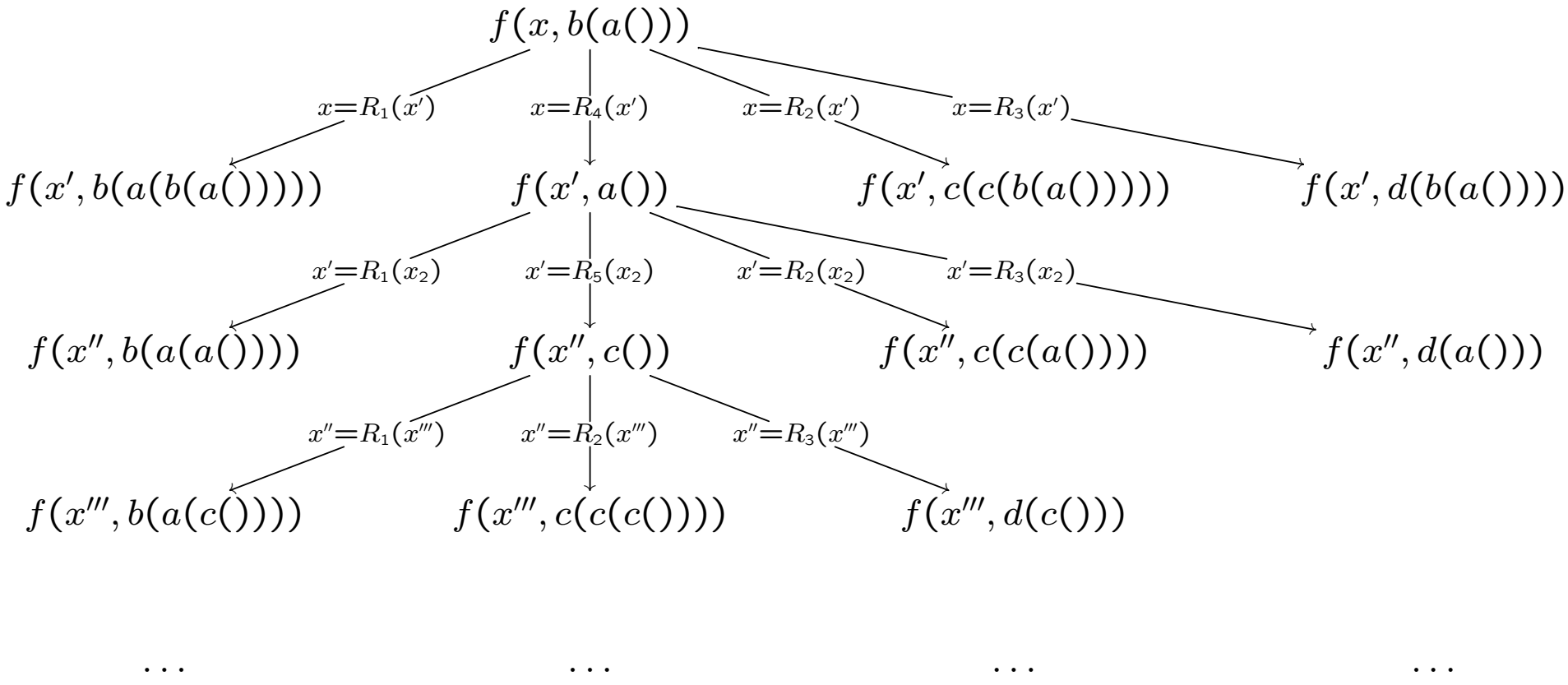
Example

This program describes the grammar $\mathbf{G_{PA}}$

$$\begin{array}{ll} \mathtt{f(R_1(x), y) = f(x, b(a(y)));} & \mathtt{f(R_4(x), b(y)) = f(x, y);} \\ \mathtt{f(R_2(x), y) = f(x, c(c(y)));} & \mathtt{f(R_5(x), a(y)) = f(x, c(y));} \\ \mathtt{f(R_3(x), y) = f(x, d(y));} & \mathtt{f(R_6(x), a(a(y))) = f(x, y);} \end{array}$$

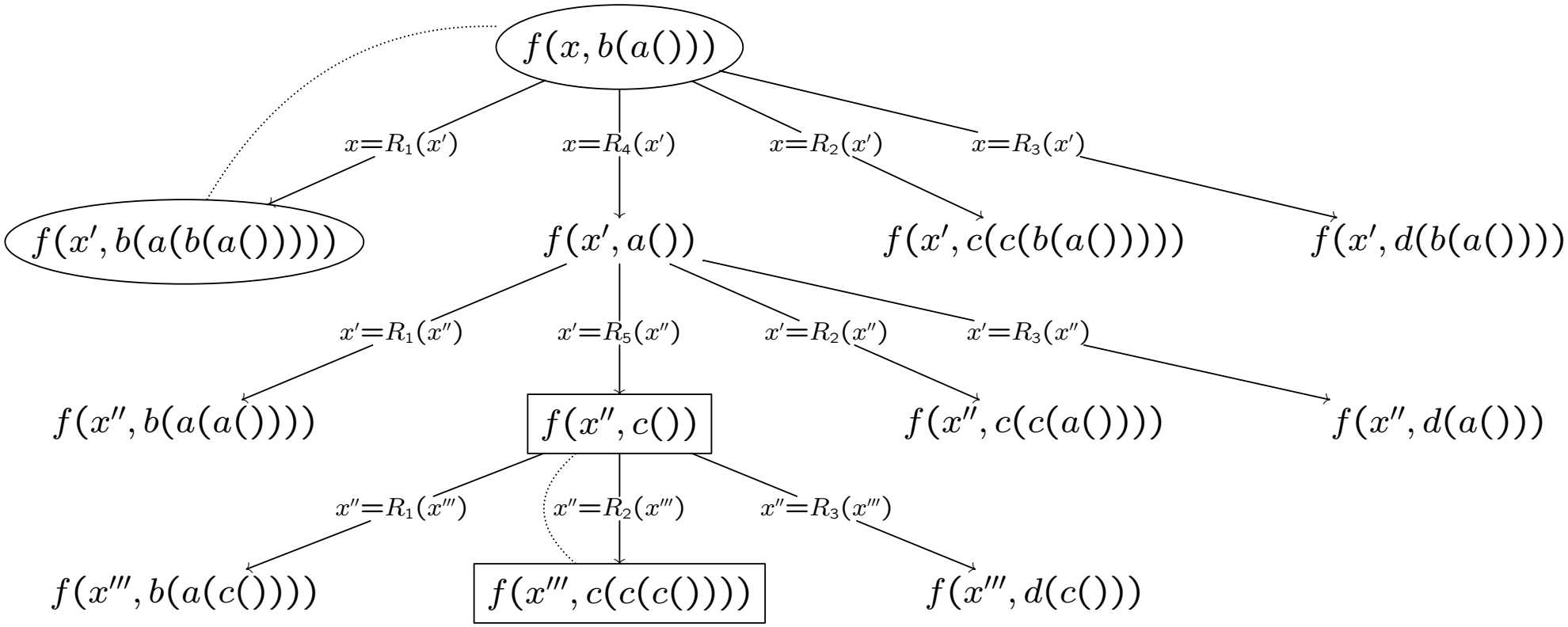
The argument x contains a history of the trace; the argument y contains a word in Σ. Introduction of x makes the program primitive recursive and also removes intersections of rules' domains.

Starting unfolding of `f(x,b(a()))`



This tree can contain branches of any length (depending on x). In the sense of semantics these ways correspond to loops.

Example



The terms in the pairs $f(x', c(c(c())))$ and $f(x, c())$ and $f(x''', c(c(c())))$ and $f(x'', c())$ are similar. This observation can imply halting runs from the corresponding child nodes.

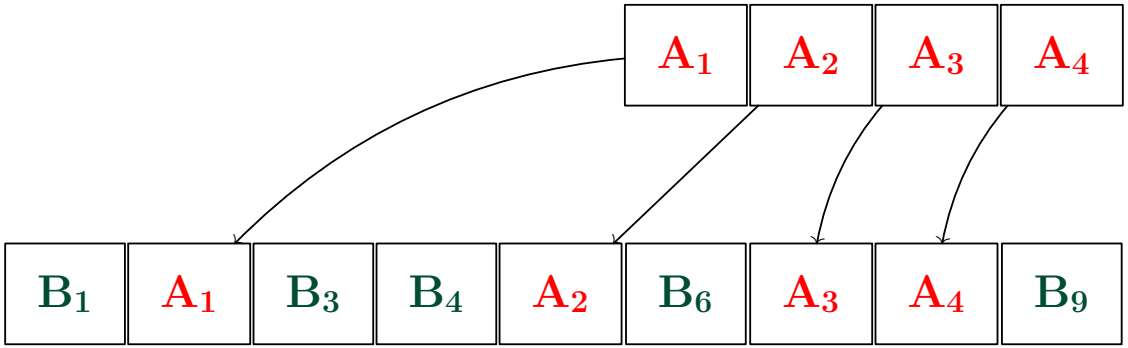
A relation R is *almost well* on a set of sequences from $S \subset \Sigma^*$ in an alphabet Σ iff every infinite sequence $\{a_n\}$ contains a_i, a_j, such that $i < j$ and $\langle a_i, a_j \rangle \in R$.

This property guarantees that all infinite branches of semantic tree must be terminated.

A sequence $\{a_n\}$ with the property $\forall i, j (i < j \Rightarrow \langle a_i, a_j \rangle \notin R)$ is called *a bad sequence*.

Classical Example

Let Σ be a finite alphabet. $A = a_1 a_2 \ldots a_m$, $B = b_1 b_2 \ldots b_n$, $\forall i, j (i \geq 1 \;\&\; i \leq m \;\&\; j \geq 1 \;\&\; j \leq n \Rightarrow a_i \in \Sigma \;\&\; b_j \in \Sigma)$. If $\forall i (i < m \Rightarrow \exists j, k (b_j = a_i \;\&\; b_k = a_{i+1} \;\&\; k > j))$ then A is embedded in B in the sense of Higman relation ($A \trianglelefteq B$).

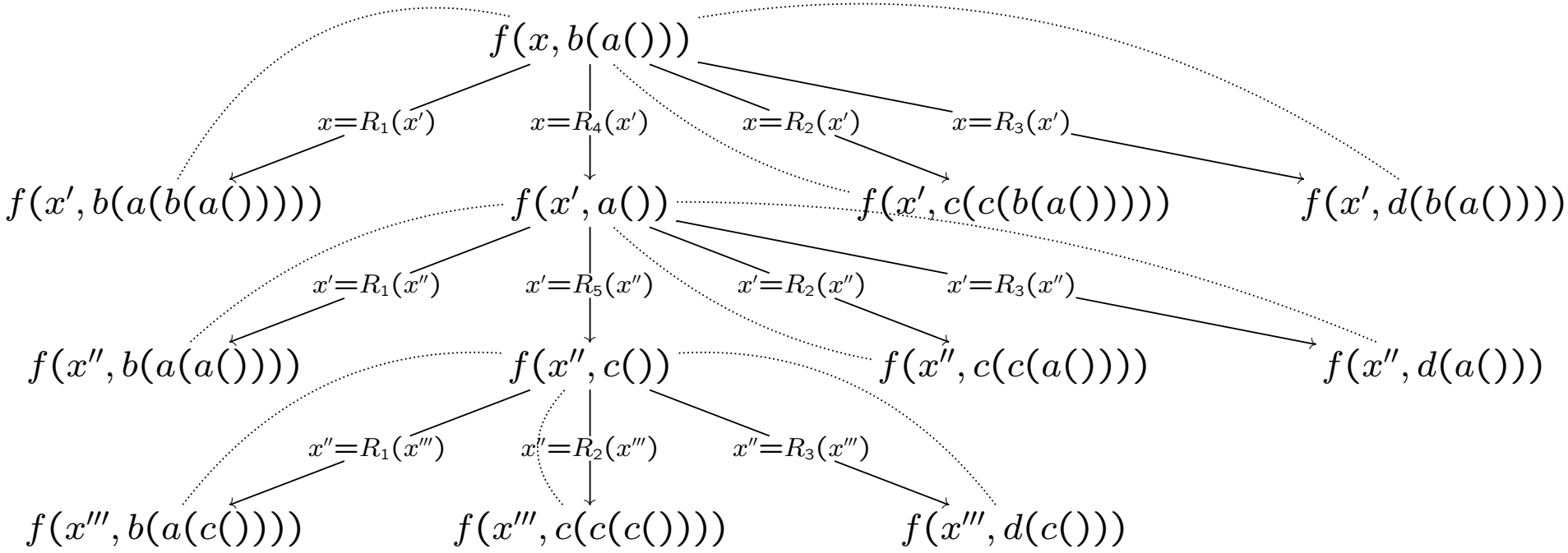


$\trianglelefteq$ is almost well on all sequences in a finite alphabet (Higman, 1952).

Is it possible to use the relation to verify ping-pong protocols?

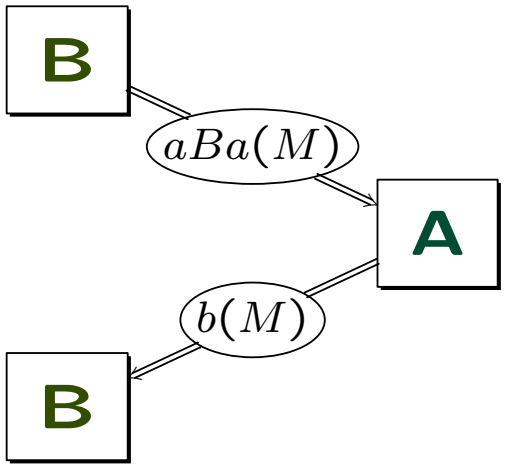
Example

When unfolding the full tree of $\mathbf{G_{PA0}}$ derivations until a pair that satisfies Higman relation is found on a branch, we do not reach Λ.



Classical Example

Consider the following protocol.



X is an appending of $\mathbf{X}$ name, x is an encryption by some open key of $\mathbf{X}$. The corresponding grammar is

$\mathbf{G_{RR}}$:

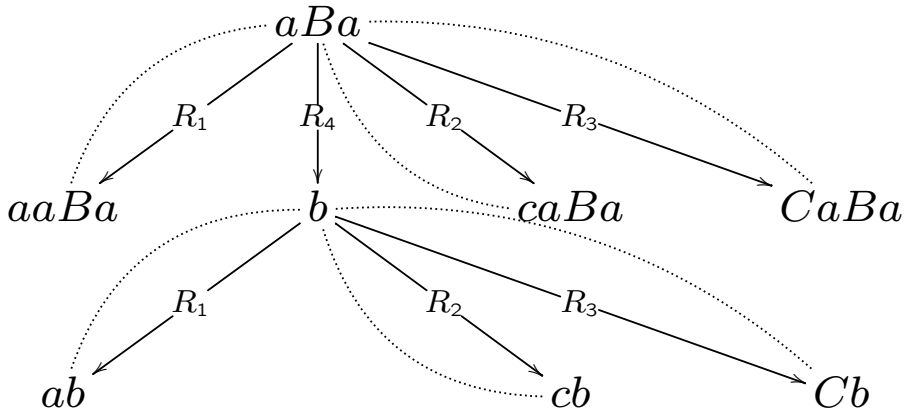
$R_1 : \Lambda \to a$ $\quad$ $R_4 : aBa \to b$ $\quad$ $R_7 : B \to \Lambda$

$R_2 : \Lambda \to c$ $\quad$ $R_5 : aCa \to c$ $\quad$ $R_8 : C \to \Lambda$

$R_3 : \Lambda \to C$ $\quad$ $R_6 : c \to \Lambda$

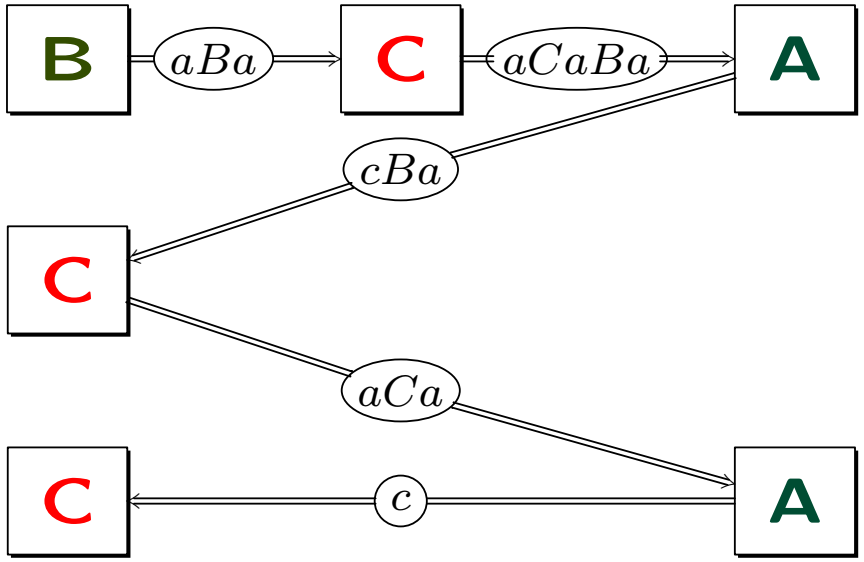
An intruder is $\mathbf{C}$.

$\Gamma_0 = aBa$. Let the branches of the tree be unfolded until finding a Higman pair:



There is no branch corresponding to an attack. But it exists:

$$aBa \to CaBa \to aCaBa \to cBa \to Ba \to a \to Ca \to aCa \to c \to \Lambda.$$



To use the unfolding technique in verification of ping-pong protocols we must construct a stronger relation.

Let a sequence $\{\Phi_i\}_{i=1}^n$ be generated by a PG $\mathbf{G} = \langle \Sigma, \mathbf{R}, \Gamma_0 \rangle$. Let us write down every Φ_i (including Γ_0) letter by letter, from the right side to the left (as Arabians do) and mark every letter by a natural number pointing the first time when the letter appeared in the sequence. This notation is called *time indexing*.

Example

$$\Gamma_0 \leftarrow \begin{cases} a_{(0)} \\ B_{(1)}a_{(0)} \\ a_{(2)}B_{(1)}a_{(0)} \end{cases}$$

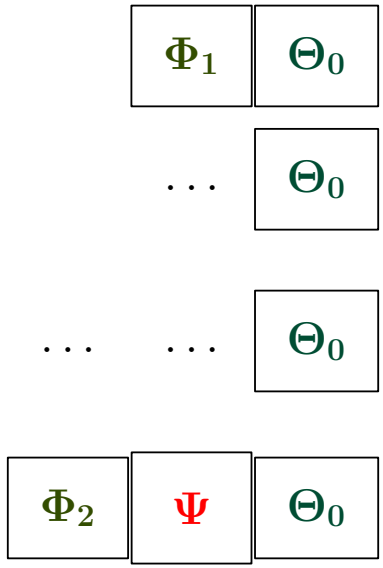
$$R_3(\Gamma_0) \leftarrow C_{(3)}a_{(2)}B_{(1)}a_{(0)}$$

$$R_1(R_3(\Gamma_0)) \leftarrow a_{(4)}C_{(3)}a_{(2)}B_{(1)}a_{(0)}$$

$$R_5(R_1(R_3(\Gamma_0))) \leftarrow c_{(5)}B_{(1)}a_{(0)}$$

Definition of Turchin relation

$$\Gamma \preceq \Delta \Leftrightarrow \Gamma = \Phi_1\Theta_0 \;\&\; \Delta = \Phi_2\Psi\Theta_0 \;\&\; \Phi_1 \approx \Phi_2$$



Generalized Turchin theorem

$\preceq$ is almost full on sequences generated by a finite PG.

$\preceq$ is surely stronger than $\trianglelefteq$. Is it enough for the task of verification?

Example

Let us try time-indexing notation on the attack built in $\mathbf{G_{RR}}$:

$$a_{(2)}B_{(1)}a_{(0)} \to C_{(3)}a_{(2)}B_{(1)}a_{(0)} \to$$

$$\to a_{(4)}C_{(3)}a_{(2)}B_{(1)}a_{(0)} \to c_{(5)}B_{(1)}a_{(0)} \to$$

$$\to B_{(1)}a_{(0)} \to a_{(0)} \to C_{(6)}a_{(0)} \to a_{(7)}C_{(6)}a_{(0)} \to c_{(8)} \to \Lambda.$$

$a_{(2)}B_{(1)}a_{(0)} \preceq a_{(4)}C_{(3)}a_{(2)}B_{(1)}a_{(0)}$ and $a_{(0)} \preceq a_{(7)}C_{(6)}a_{(0)}$.

This means that the branch is also terminated before it reaches Λ.

Consider the following modification of $\mathbf{G_{PA0}}$.

$\mathbf{G_{PAC}}$:

$R_1 : \Lambda \to ba$ $\quad$ $R_4 : b \to \Lambda$ $\quad$ $R_7 : \Lambda \to bac$

$R_2 : \Lambda \to a^{-1}a^{-1}$ $\quad$ $R_5 : a \to a^{-1}$

$R_3 : \Lambda \to b^{-1}$ $\quad$ $R_6 : aa \to \Lambda$

The rules R_1 and R_7 are different in essence: application of the latter makes Λ impossible in the further derivation. But $\preceq$ does not make difference between them because of the same prefix ba.

It is desirable to forbid occasional overlaps of right-hand sides' parts to get rid of the unwanted embeddings.

PG $\mathbf{G}{=}\langle \mathbf{\Sigma}, \mathbf{R}, \Gamma_0 \rangle$ is *annotated*, iff every two rules either have the same right-hand side or share no letters in the right-hand sides.

Every grammar can be transformed to annotated so that all bad sequences constructed by the initial grammar are transformed to bad sequences constructed by the annotated.

For example, the same letters in distinct right-hand sides can be colored differently. Left-hand sides of rules ignore colouring.

Example

Let us annotate $\mathbf{G_{RR}}$ (Γ_0 is R_0).

$\mathbf{G_{ARR}}$:

$R_1 : \Lambda \to a$ $\quad$ $R_4 : aBa \to b$ $\quad$ $R_7 : B \to \Lambda$

$R_2 : \Lambda \to c$ $\quad$ $R_5 : aCa \to c$ $\quad$ $R_8 : C \to \Lambda$

$R_3 : \Lambda \to C$ $\quad$ $R_6 : c \to \Lambda$ $\quad$ $R_0 : \Lambda \to aBa$

Now the attack sequence is

$$a_{(2)}B_{(1)}a_{(0)} \to C_{(3)}a_{(2)}B_{(1)}a_{(0)} \to$$

$$\to a_{(4)}C_{(3)}a_{(2)}B_{(1)}a_{(0)} \to c_{(5)}B_{(1)}a_{(0)} \to$$

$$\to B_{(1)}a_{(0)} \to a_{(0)} \to C_{(6)}a_{(0)} \to a_{(7)}C_{(6)}a_{(0)} \to c_{(8)} \to \Lambda.$$

No words in this sequence form a Turchin pair, and the sequence is found by unfolding.

Consider a relation $\preceq^T$ such that $\Gamma \preceq^T \Delta$ iff $\Gamma = \Phi\Theta_0$, $\Delta = \Phi'\Psi\Theta_0$ and there exists such $R : R_l \to R_r$ ($R_r \neq \Lambda$) that $\Phi \approx R_r$ and $\Phi' \approx R_r$. Every infinite sequence generated by a PG contains an infinite chain of words comparable with respect to $\preceq^T$.

Usage of $\preceq^T$ instead of $\preceq$ is equivalent to the annotating.

Let $\mathbf{G}$ be an annotated PG of the type 2. If it generates a sequence ending with Λ, then there exists a sequence generated by $\mathbf{G}$ and ending with Λ with no Turchin pairs.

Thus the problem of verification of ping-pong protocols via unfolding is solved for a restricted class of the protocols.

Example

Consider the attack generated by $\mathbf{G_{PA0}}$.

$$b_{(1)}a_{(0)} \to a_{(0)} \to b_{(2)}a_{(1)}a_{(0)} \to a_{(1)}a_{(0)} \to \Lambda$$

It cannot be found by even $\preceq^T$, for $b_{(1)}a_{(0)}$ and $b_{(2)}a_{(1)}a_{(0)}$ are directly generated by the same rule.

But the ways of *erasing* of $a_{(0)}$ and $a_{(1)}$ are different.

Let x be a letter in the right-hand side of a rule. Its *erasing counter* is the number of occurrences of x in all left-hand sides.

For example, a from $\mathbf{G_{PA0}}$ has erasing counter 3, and b has erasing counter 1.

Every letter in a rule of a type 2 PG has erasing counter 0 (if it cannot be erased) or 1.

$\Gamma \preceq^{!} \Delta$ (Γ forms with Δ *a Turchin pair with erasing distinction*) iff $\Gamma \preceq^{T} \Delta$ and for some $x \in \Sigma$ the number of occurrences of x in Δ is greater than its erasing counter.

Theorem 1

Let $\mathbf{G}$ be a 0-type finite PG. Every infinite sequence generated by $\mathbf{G}$ contains such Γ and Δ that either $\Gamma \approx \Delta$ or $\Gamma \preceq^{!} \Delta$.

Theorem 2

If $\mathbf{G}$ generates Λ-ending sequences, then it generates also a Λ-ending sequence that contains no Turchin pairs with erasing distinction.

These two propositions give us a sound criterion of branch termination while verifying the protocols in the form of PG grammars via unfolding.

Conclusion

The unfolding technique which is used in program transformation can be used in ping-pong protocol verification if the additional annotation of erasings is introduced.

1. The ping-pong protocols can be modeled as PGs with the slight generalization of intruder's behaviour model (now it admits composite actions that are indivisible for an intruder).

2. This model can be used by program transformation tools with unfolding to verify the protocols (in the sense of finding Λ-ending traces) using an appropriate termination criterion (for the generic model the criterion is unfolding until $\preceq^!$-pair).

Thanks!

The Initial Program Model of G_{PA0}

```
$ENTRY Go {e.1 = <Check ('I')('I')()()(e.1)(<Const 'b1'>)(<Const 'a0'>)>;}


Check {(e.1)(e.2)(e.3)(e.4)() = True;

       (e.1)(e.2)(e.3)(e.4)() (s.A s.Ind) e.y = False;

       (e.1'II')(e.2)(e.3)(e.4)((R1) e.x) e.y =

                <Shorten (e.1'III')(e.2'I')(e.3)(e.4)(e.x)('b3')('a2')<Uncst e.y> >;

       (e.1)(e.2'I')(e.3)(e.4)((R1) e.x) e.y =

                <Shorten (e.1'I')(e.2'II')(e.3)(e.4)(e.x)('b3')('a2') <Uncst e.y> >;

       (e.1)(e.2)(e.3)(e.4)((R1) e.x) e.y =

                <Shorten (e.1'I')(e.2'I')(e.3)(e.4)(e.x)(<Const 'b3'>)(<Const 'a2'>) e.y>;

       (e.1)(e.2)(e.3'I')(e.4)((R2) e.x) e.y =

                <Shorten (e.1)(e.2)(e.3'III')(e.4)(e.x)('c4')('c6') <Uncst e.y> >;

       (e.1)(e.2)(e.3)(e.4)((R2) e.x) e.y =

                <Shorten (e.1)(e.2)(e.3'II')(e.4)(e.x)(<Const 'c4'>)(<Const 'c6'>) e.y>;

       (e.1)(e.2)(e.3)(e.4'I')((R3) e.x) e.y =

                <Shorten (e.1)(e.2)(e.3)(e.4'II')(e.x)('d5') <Uncst e.y»;

       (e.1)(e.2)(e.3)(e.4)((R3) e.x) e.y =

                <Shorten (e.1)(e.2)(e.3)(e.4'I')(e.x)(<Const 'd5'>) e.y>;}
```

```
Shorten { (e.1)(e.2'I')(e.3)(e.4'I')(e.x) e.y ('b' s.1)('d' s.2) e.z =
                <Shorten (e.1)(e.2)(e.3)(e.4)(e.x)e.y e.z>;
          (e.1)(e.2'I')(e.3)(e.4'I')(e.x) e.y ('d' s.1)('b' s.2) e.z =
                <Shorten (e.1)(e.2)(e.3)(e.4)(e.x)e.y e.z>;
          (e.1'I')(e.2)(e.3'I')(e.4)(e.x) e.y ('c' s.1)('a' s.2) e.z =
                <Shorten (e.1)(e.2)(e.3)(e.4)(e.x)e.y e.z>;
          (e.1'I')(e.2)(e.3'I')(e.4)(e.x) e.xx ('a' s.1)('c' s.2) e.xxx =
                <Shorten (e.1)(e.2)(e.3)(e.4)(e.x)e.y e.z>;
           e.z = <Check e.z>;}
```

The Residual Program (refined)

```
$ENTRY Go {(R3)(R1)(R3)(R2) e.1 = <F e.1>;
            e.2 = False;}


F {   = True;
   (R3)(R1)(R3)(R2) e.1 = <F e.1>;
   e.2 = False;}
```