From Symbolic Constraint Automata to Promela

Hui Feng, Marcello Bonsangue, and Benjamin Lion

Areas of concern

Formal protocols

Letter: element $s \in \Sigma$.
Words: sequence $w \in \Sigma^*$.
Formal languages: set of words $L \subseteq \Sigma^*$.
Application: semantics of computer programs (termination).

Areas of concern
Formal protocols

Letter: element $s \in \Sigma$.
Words: sequence $w \in \Sigma^*$.
Formal languages: set of words $L \subseteq \Sigma^*$.
Application: semantics of computer programs (termination).

Letter: element $s \in \Sigma$.
Run: stream $\sigma \in \Sigma^\omega$.
Formal protocols: set of streams $L \subseteq \Sigma^\omega$
Application: semantics of computer networks (liveness properties).

Protocol Design and verification

Compositional design:

- set of protocol primitives $p_1, ..., p_n \subseteq \Sigma^\omega$;
- composite protocol $P = p_1 \cap ... \cap p_n \subseteq \Sigma^\omega$.

Protocol Design and verification

Compositional design:

- set of protocol primitives $p_1, ..., p_n \subseteq \Sigma^\omega$;
- composite protocol $P = p_1 \cap ... \cap p_n \subseteq \Sigma^\omega$.

Verification:

- given a property $Prop \subseteq \Sigma^\omega$;
- check that $p_1 \cap ... \cap p_n \subseteq Prop$.

Protocol Design and verification

Compositional design:

- set of protocol primitives $p_1, ..., p_n \subseteq \Sigma^\omega$;
- composite protocol $P = p_1 \cap ... \cap p_n \subseteq \Sigma^\omega$.

Verification:

- given a property $Prop \subseteq \Sigma^\omega$;
- check that $p_1 \cap ... \cap p_n \subseteq Prop$.

References:

- Behaviors of Processes and Synchronized Systems of Processes (Nivat, 1982),
- A Co-inductive Calculus of Component Connector (Arbab and Rutten, 2002).

Plan

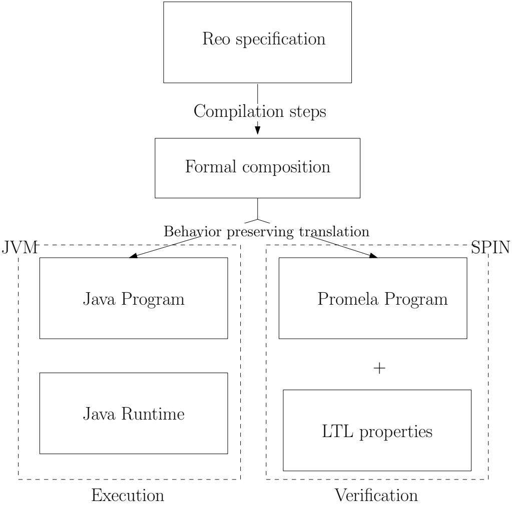
We present Reo, a connector-based language for compositional design of protocols.

We introduce a state-based specification for Reo connectors.

We provide a behavior preserving translation to Promela...

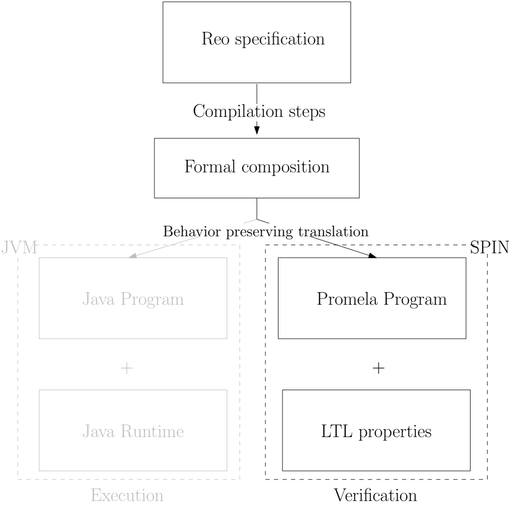
that we use to apply model checking with Spin.

Highlights



Reo specification
Compilation steps
Formal composition
Behavior preserving translation
JVM
SPIN
Java Program
Java Runtime
Promela Program
+
LTL properties
Execution
Verification

Highlights



Reo specification
Compilation steps
Formal composition
Behavior preserving translation
JVM
Java Program
+
Java Runtime
Execution
SPIN
Promela Program
+
LTL properties
Verification

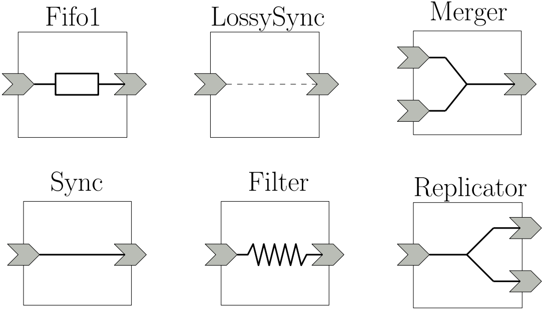
Reo

We fix the alphabet to a set of port assignments, i.e., $\Sigma = \mathbb{P} \to \mathbb{D}_\star$.

A Reo primitive over a set of ports $P \subseteq \mathbb{P}$ denotes a subset $R(P) \subseteq \Sigma^\omega$ that may constrain ports in P only.

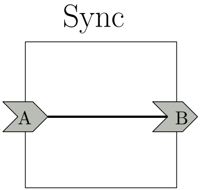
Composition is intersection, i.e., $R_1(P_1) \cap R_2(P_2)$ for two primitives $R_1(P_1)$ and $R_2(P_2)$.

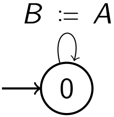
Primitives of interaction



Sync channel

Graphical and syntax





Sync channel

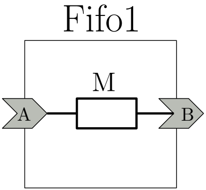
Semantics

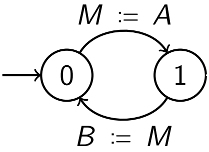
Set of behavior of the $Sync(A, B)$ channel:

$$\left\{ \begin{array}{c|c} A & B \\ \hline d_1 & d_1 \\ * & * \\ d_2 & d_2 \\ \dots & \dots \end{array} \quad , \; \dots \; , \quad \begin{array}{c|c} A & B \\ \hline * & * \\ * & * \\ d_1 & d_1 \\ \dots & \dots \end{array} \right\}$$

Fifo channel

Graphical and syntax





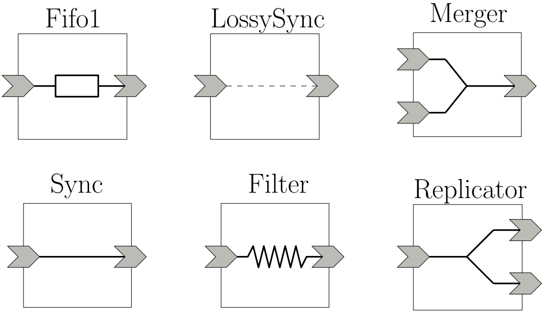
Fifo channel

Semantics

Set of behavior of the $Fifo1(A, B, M)$ channel:

$$\left\{ \begin{array}{c|c|c} A & M & B \\ \hline d_1 & * & * \\ * & d_1 & * \\ * & d_1 & * \\ * & d_1 & d_1 \\ \dots & \dots & \dots \end{array} \quad , \dots , \quad \begin{array}{c|c|c} A & M & B \\ \hline d_1 & * & * \\ * & d_1 & d_1 \\ d_2 & * & * \\ * & d_2 & * \\ \dots & \dots & \dots \end{array} \right\}$$

Primitives of interaction



Syntax:

- Ports are variables;
- Assignments are solutions to guarded commands;
- Channels are within a fragment of constraint automata

Symbolic Constraint Automata

Guarded Actions

We fix a set of terms:

$$t ::= d \mid x \mid f(\bar{t})$$

with d data constants, x port and memory variables, and f function symbols.
A guarded action is a formula of the form:

$$P(\bar{x}) \rightarrow \bar{y} := t$$

where

- P is a guard on $\bar{x}$;
- $\bar{x}$ is a vector of input ports or memory variables;
- $\bar{y}$ is a vector of output ports or memory variables;
- free variables in t are either input or memory variables.

Symbolic Constraint Automata

Guarded Actions

The set of guarded actions over inputs I, outputs O, and memories V is written $Act(I, O, V)$.

Examples:

- $(i \leq v \rightarrow (o, v) \coloneqq [i, i])$ is a guarded action in $Act(\{i\}, \{o\}, \{v\})$;
- $(i \leq v \rightarrow (o, i) \coloneqq [i, o])$ is not a guarded action in $Act(\{i\}, \{o\}, \{v\})$, as i and o appear both on the left-hand side and on the right-hand side of the action, respectively.

Symbolic Constraint Automata

Syntax and Semantics

A symbolic constraint automaton $A = (Q, q_0, I, O, V, \rightarrow)$ is such that

- $q_0 \in Q$,
- $I \cap O = I \cap V = O \cap V = \emptyset$,
- $\rightarrow \subseteq Q \times Act(I, O, V) \times Q$.

Symbolic Constraint Automata

Syntax and Semantics

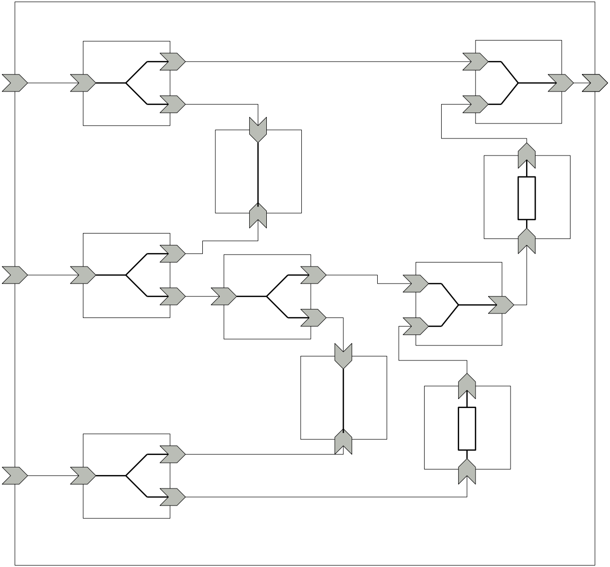
A symbolic constraint automaton $A = (Q, q_0, I, O, V, \rightarrow)$ is such that

- $q_0 \in Q$,
- $I \cap O = I \cap V = O \cap V = \emptyset$,
- $\rightarrow \subseteq Q \times Act(I, O, V) \times Q$.

The semantics $[\![A]\!]$ of A is the set of streams of assignements $\sigma \in (\mathbb{P} \rightarrow \mathbb{D}_{\star})^{\omega}$ where, for all $n \in \mathbb{N}$, there is a transition $q_n \xrightarrow{P(\bar{x}) \rightarrow \bar{y} := t} q_{n+1}$ with:

1. the interpretation of the guard of $P(\bar{x})$ holds in σ_n,
2. for all output and memory variables $x \in O(\alpha) \cup V(\alpha)$ occuring in $\bar{y}$, the assignment σ_{n+1} inherits the values assigned by the command $\bar{y} := t$ evaluated with σ_n;
3. for all variables x not involved in the guarded action α, the value does not change, that is, $\sigma_{n+1}(x) = \sigma_n(x)$.

Alternator: a composite connector



Composition

Given two automata A_1 and A_2, we form their product $A_1 \bowtie A_2$, such that $[\![A_1 \bowtie A_2]\!] = [\![A_1]\!] \cap [\![A_2]\!]$.

Additionally, we assume that:

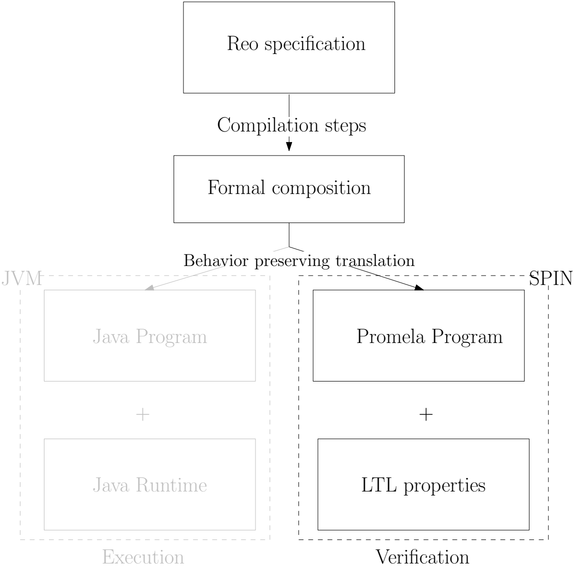
- variables in A_1 and A_2 are disjoint;
- every pair of guarded actions (a_1, a_2) in A_1 and A_2 synchronize on some inputs or some outputs but not both, i.e., $I(a_1) \cap O(a_2) \neq \emptyset \implies I(a_2) \cap O(a_1) = \emptyset$.

We hide X in A with $\exists X.A$ where $[\![\exists X.A]\!] = \mathrm{pr}_{\mathbb{P}\setminus X}[\![A]\!]$.

Composite protocol

$$
\begin{aligned}
A_{alternator}&(A, E, M, K) = \\
&\exists B, C, J, L, E, D, F, I, O, L, G, H, N, \\
&A_{rep}(A, B, C) \bowtie A_{fifo1}(J, L) \bowtie A_{merger}(B, J, K) \bowtie \\
&A_{replicator}(E, D, F) \bowtie A_{fifo1}(I, O) \bowtie A_{merger}(G, I, L) \bowtie \\
&A_{replicator}(F, G, H) \bowtie A_{sync}(H, N) \bowtie A_{replicator}(M, N, O)
\end{aligned}
$$

Compilation to Promela



Hint for correctness

<table>
<tr><th>Logic</th><th>Promela</th></tr>
<tr><td>port variables</td><td><pre>   typedef port {
    chan data = [1] of {Data};
    chan sync = [1] of {int}; }</pre></td></tr>
<tr><td>$q_i \xrightarrow{P_i(\bar{x}) \to \bar{y}:=t_i} q_{i+1}$</td><td><pre>(guard_i /\ s=i) ->
        atomic{command_i /\ s= i+1}</pre></td></tr>
<tr><td>$Protocol = (Q, q_0, I, O, V, \to)$</td><td><pre>proctype Protocol(port p1; ...){
∀p ∈ I ∪ O, Data _p;
∀v ∈ V, Data v;
int state = 0;
/* Guarded commands */
do
:: transition_1

:: ...

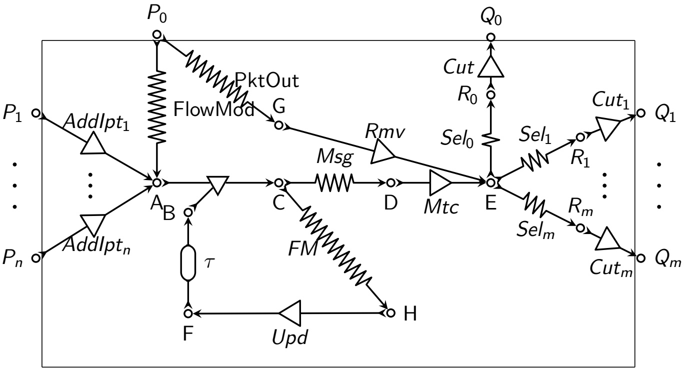
:: transition_n
od}</pre></td></tr>
</table>

Properties

Properties	Temporal formulas
p_fires	len(p.data) != 0 && len(p.sync)!=0 && X(len(p.data)==0 $\|$ *len*(*p.sync*) == 0)
p_silent	!(p_fires)
m_full	len(m.data)!=0
m_empty	len(m.data)==0
m_fires	m_full && X(m_empty)) $\|$ m_empty && X(m_full)
m_silent	!(m_fires)
p1_before_p2	(p1_fires $\rightarrow \Diamond$ (p2_fires))
p1_then_p2	(p1_fires $\rightarrow$ X(silent U p2_fires))

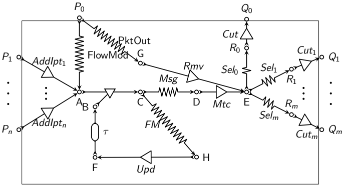
Example

Software Defined Network



Example

Verification in Spin



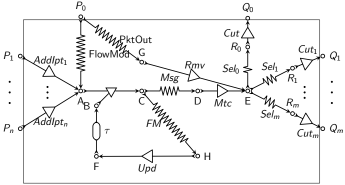
Temporal properties:

$\phi_1 = \Box$((A.data=1)$\rightarrow \Diamond$(B.data==1))

$\phi_2 = \Box$((A.data==2)$\rightarrow$($\Diamond$(B.data==2)$\wedge\Diamond$(C.data==2)))

Example

Verification in Spin



Temporal properties:

$\phi_1 = \Box$((A.data=1)$\rightarrow \Diamond$(B.data==1))

$\phi_2 = \Box$((A.data==2)$\rightarrow$($\Diamond$(B.data==2)$\wedge\Diamond$(C.data==2)))

Results from Spin:

	Errors found	Time usage	Depth reached	States stored	States matched	Transi -tions
ϕ_1	0	10.5s	479	214292	660445	1058424
ϕ_2	0	8.64s	479	152057	373096	767167

Conclusion

Theory: Description of compilation steps from Reo protocol into Promela programs.

Practice: Implementation of the theoretical results as extension of Reo compiler. Application on a case study.