IMT SCHOOL FOR ADVANCED STUDIES LUCCA

Programming Collective Adaptive Systems by relying on Attribute-based Communication

Rocco De Nicola

Joint work with
M. Loreti, Y. Abd Alrahman and Tan Duong

IFIP W.G. 2.2 - Wien - September 2019

IMT- School for Advanced Studies - Lucca

Outline

Languages for supporting the engineering of different classes of modern distributed systems

- network-aware programming
- service-oriented computing
- autonomic computing.

Programming collective adaptive systems

AbC: A calculus for Attribute based Programming

- Syntax
- Semantics
- Implementations
- Verification

Why a Languages based Approach

Languages

Languages play a key role in the engineering of systems

- Systems must be specified as naturally as possible
- Distinctive aspects of the domain need to be first-class citizens
- Intuitive/concise specifications are possible and encodings can be avoided

Models

Models strictly related to languages are at least as important for effective analysis

- high-level abstract models guarantee feasible investigations
- the scrutiny of results (e.g., counterexample) based on system features, rather than on their low-level representation, guarantees better feedbacks.

Language-based methodology

Major challenge

The big challenge for language designers is to devise appropriate abstractions and linguistic primitives to deal with the specificities of the systems under consideration while relying on an appropriate semantic model.

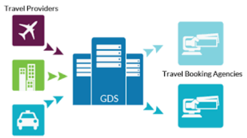
A possible approach

Combined use of formal methods with model-driven software engineering. Key ingredients are

1. A specification language equipped with a formal semantics
2. A programming framework with associated runtime environment
3. A number of verification techniques and associated tools

Our Contributions: A timeline

Service-oriented computing
- services composition
- heterogeneous components
- code reuse
- interoperability



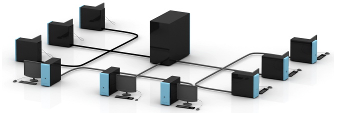
Collective adaptive systems progr.
- large number of components
- decentralised control
- unpredictable environment
- emergent behaviour

2006-2009 SCC - COWS - CASPIS

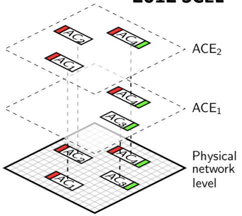
2016 AbC

1998 Klaim

2012 SCEL



Network-aware programming
- awareness of the network infrastructure
- asynchronous interactions
- open-ended non-determ. environment
- computation mobility

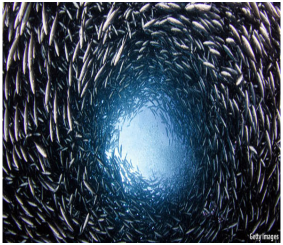


Autonomic computing
- reduced maintenance cost
- no human intervention
- continuous monitoring
- adaptation

Collective Adaptive Systems

We are surrounded by examples of collective systems, in the natural world ....

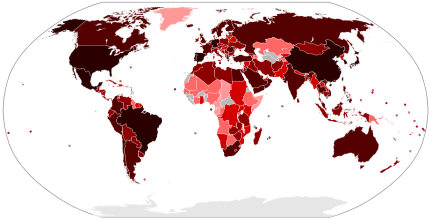
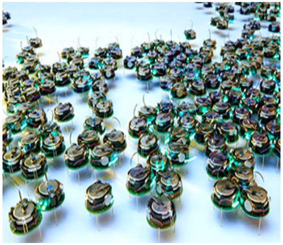
- Bees, Fishes, Birds, . . .



Collective Adaptive Systems

... and in the man-made world

- Traffic, Epidemics, Robots, . . .

Collective Adaptive Systems

Many components

From a computer science perspective, collective adaptive systems can be viewed as consisting of a large number of interacting entities.

Local behaviour

Each entity may have its own properties, objectives and actions and at the system level the entities combine to create the collective - emergent - behaviour.

Mutual Influence

The behaviour of the system is dependent on that of the individual entities and the behaviour of the individuals will be influenced by the state of the overall system.

No Central Control

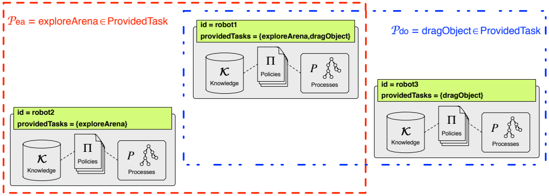
CAS need to operate without centralised control or direction. When conditions within the system change it may not be feasible to have human intervention to adjust behaviour appropriately and systems must autonomously adapt.

IMT SCHOOL FOR ADVANCED STUDIES LUCCA

The SCEL language: ensembles

Ensembles formation

- Attributes are used by the system components to dynamically organize themselves into ensembles
- Predicates $\mathcal{P}$ over attributes are used by components to specify the targets of communication actions.



$\mathcal{P}_{ea}$ = exploreArena ∈ ProvidedTask
$\mathcal{P}_{do}$ = dragObject ∈ ProvidedTask
id = robot1
providedTasks = {exploreArena,dragObject}
id = robot2
providedTasks = {exploreArena}
id = robot3
providedTasks = {dragObject}
$\mathcal{K}$ Knowledge
Π Policies
P Processes

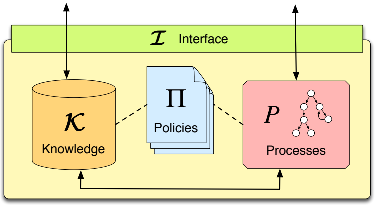
- Ensembles are determined by the predicates validated by each component
- There is no coordinator, hence no bottleneck or critical point of failure
- A component might be part of more than one ensemble

IMT SCHOOL FOR ADVANCED STUDIES LUCCA

The SCEL language

Introduced to deal with the challenges posed by the design of ensembles of autonomic components

An autonomic component in SCEL:



- Knowledge repositories where components store and retrieve information about their working environment and to use it for determining and adapting their behaviour
- Policies regulating the inter- and intra-components interaction
- Interfaces consisting of a collection of attributes, like provided functionalities, spatial coordinates, group memberships, trust level, response time, ...

A Calculus for Attribute based Communication

Yehia Abd Alrahman, Rocco De Nicola, Michele Loreti:
On the Power of Attribute-Based Communication
FORTE 2016 and Information and Computation 2019

...

...

Rocco De Nicola, Tan Duong, and Michele Loreti:
ABEL - A DSL for programming with attribute-based communication
Coordination 2019

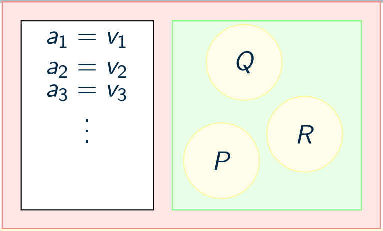
AbC: a calculus distilled from SCEL

- Systems are represented as sets of parallel components, each equipped with a set of attributes whose values can be modified by internal actions.
- Communication actions (send and receive) are decorated with predicates over attributes that partners have to satisfy to make the interaction possible.
- Communication takes place in an implicit multicast fashion: partners are selected via predicates over the attributes exposed in their interfaces.
- Components are unaware of the existence of each other and receive messages only if they satisfy senders requirements.
- Components can offer different views of themselves and can communicate with different partners according to different criteria.
- Semantics for output actions is non-blocking while input actions are blocking: they can take place through synchronization with an available sent message.

AbC: basic ingredients.

An AbC system consists of a set components that contain

- a behaviour - a set of running processes
- an environment - a map from attributes names to values



Processes can:

- send a message to all the components satisfying a given predicate;
- receive a message from a component satisfying a given predicate;
- change the environment;
- wait until a given predicate is locally satisfied.

SCHOOL FOR ADVANCED STUDIES LUCCA

AbC Syntax

Components	$C ::= \Gamma :_I P \quad \mid \quad C_1 \| C_2 \quad \mid \quad [\ C\ ]^{\triangleleft f} \quad \mid \quad [\ C\ ]^{\triangleright f}$
Processes	$P ::= 0 \quad \mid \quad \Pi(\tilde{x}).U \quad \mid \quad (\tilde{E})@\Pi.U \quad \mid \quad \langle \Pi \rangle P \quad \mid$ $P_1 + P_2 \quad \mid \quad P_1 \mid P_2 \quad \mid \quad K(x_1, \ldots, x_n)$
Updates	$U ::= [a := E]U \quad \mid \quad P$
Predicates	$\Pi ::= \mathsf{tt} \mid \mathsf{ff} \mid p_k(E_1, \ldots, E_k) \mid \Pi_1 \wedge \Pi_2 \mid \Pi_1 \vee \Pi_2 \mid \neg \Pi$
Expressions	$E ::= v \quad \mid \quad x \quad \mid \quad a \quad \mid \quad this.a \quad \mid \quad o_k(E_1, \ldots, E_k)$

IMT SCHOOL FOR ADVANCED STUDIES LUCCA

AbC: Interfaces

A basic component, $\Gamma :_I P$, is a process P associated with an attribute environment Γ, and an interface I.

- The attribute environment $\Gamma : \mathcal{A} \rightharpoonup \mathcal{V}$ is a partial map from attribute identifiers with $a \in \mathcal{A}$ to values $v \in \mathcal{V}$ where $\mathcal{A} \cap \mathcal{V} = \emptyset$. A value could be a number, a name (string), a tuple, etc.
- The interface $I \subseteq \mathcal{A}$ consists of a *finite* set of attributes names that are exposed by a component to control the interactions with other components.
- Attributes in I are public, and to those in $dom(\Gamma) - I$ are private.

AbC: Controlling Interaction

Two operators $[\ C\ ]^{\triangleleft f}$ and $[\ C\ ]^{\triangleright f}$ are introduced to restrict information flow. Function f associates a predicate Π to each tuple of values $\tilde{v} \in \mathcal{V}^*$ and attribute environment Γ.

- $[\ C\ ]^{\triangleright f}$ is used to restrict the messages that component C can send.
  - When the message outgoes $[\ C\ ]^{\triangleright f}$, the target predicate is updated to consider also predicate $\Pi' = f(\Gamma, \tilde{v})$
  - Only components satisfying $\Pi \wedge \Pi'$ will receive the message.
  - To prevent a specific *secret* s from being spread outside C, one can use $f_s(\Gamma, \tilde{v}) = \mathrm{tt}$ if $s \notin \tilde{v}$ and $f_s(\Gamma, \tilde{v}) = \mathrm{ff}$ otherwise.
- $[\ C\ ]^{\triangleleft f}$ is used to restrict the messages that component C can receive.
  - If a component with public attribute environment Γ sends a message $\tilde{v}$ to components C satisfying Π, only those components in C that satisfy $\Pi \wedge f(\Gamma, \tilde{v})$ are eligible to receive the message.

AbC: Processes

A process P can be the:

- inactive process 0,
- action-prefixed process, $act.U$, where act is a communication action and U is a process possibly preceded by an attribute update,
- self-aware process $\langle \Pi \rangle P$, blocks the execution of P until predicate Π is satisfied within the attribute environment where the process is executing and triggers execution of P when the environment changes and $\Gamma \models \Pi$
- nondeterministic choice between two processes $P_1 + P_2$,
- interleaving composition of two processes $P_1|P_2$, processes can only communicate indirectly through the attribute environment they share
- parametrised process call with a unique identifier K and a sequence of formal parameters $(x_1, \ldots, x_n)$ used in the process definition $K(x_1, \ldots, x_n) \triangleq P$.

Predicate based communication

Using attributes

- attribute-based output $(\tilde{E})@\Pi$ is used to send the evaluation of the sequence of expressions $\tilde{E}$ to the components whose attributes satisfy the predicate Π.
- attribute-based input $\Pi(\tilde{x})$ is used to receive messages from any component whose attributes (and possibly transmitted values) satisfy the predicate Π; the sequence $\tilde{x}$ acts as a placeholder for received values.
- attribute update $[a := E]$ is used to assign the result of the evaluation of E to the attribute identifier a. Updates are only possible after communication actions: they can be viewed as side effects of interactions. Execution of a communication action and the following update(s) is atomic.

Predicates can refer to public and private attributes of components.

$$(\text{“Req”}, 1, 3)@(\mathrm{i} \geq \texttt{this}.\mathrm{i})$$

can be used to send the message $(\text{“Req”}, 1, 3)$ to all components whose attribute i is not less than $\texttt{this}$.i.

Semantics rules: Potential Communications

$$\frac{[\![\tilde{E}]\!]_\Gamma = \tilde{v} \quad \{\Pi_1\}_\Gamma = \Pi}{\Gamma :_I (\tilde{E})@\Pi_1.U \xmapsto{\Gamma \downarrow I \triangleright \overline{\Pi}(\tilde{v})} \{\!|\Gamma :_I U|\!\}} \text{ BRD}$$

Expressions in $\tilde{E}$ are evaluated to $\tilde{v}$, and the *closure* Π of predicate Π_1 under Γ is computed then $\tilde{v}$, $\{\Pi_1\}_\Gamma$ and $\Gamma \downarrow I$. Environment updates may be applied.

$$\frac{\Gamma' \models \{\Pi_1[\tilde{v}/\tilde{x}]\}_{\Gamma_1} \quad \Gamma_1 \downarrow I \models \Pi}{\Gamma_1 :_I \Pi_1(\tilde{x}).U \xmapsto{\Gamma' \triangleright \Pi(\tilde{v})} \{\!|\Gamma_1 :_I U[\tilde{v}/\tilde{x}]|\!\}} \text{ RCV}$$

A message can be received when $\Gamma_1 \downarrow I$ satisfies sender's predicate Π, and the environment of the sender Γ' satisfies the receiving predicate $\{\Pi_1[\tilde{v}/\tilde{x}]\}_{\Gamma_1}$. Updates U under substitution $[\tilde{v}/\tilde{x}]$ may be applied.

Atomicity of Communications and Updates

$$\{\!|C|\!\} = \begin{cases} \{\!|\Gamma[a \mapsto [\![E]\!]_\Gamma] :_I U\ |\!\} & C = \Gamma :_I [a := E]U \\ \Gamma :_I P & C = \Gamma :_I P \end{cases}$$

Semantics rules: Actual Interactions

$$\frac{C_1 \xrightarrow{\Gamma \triangleright \Pi(\tilde{v})} C_1' \quad C_2 \xrightarrow{\Gamma \triangleright \Pi(\tilde{v})} C_2'}{C_1 \parallel C_2 \xrightarrow{\Gamma \triangleright \Pi(\tilde{v})} C_1' \parallel C_2'} \textsc{Sync} \qquad \frac{C_1 \xrightarrow{\Gamma \triangleright \overline{\Pi}(\tilde{v})} C_1' \quad C_2 \xrightarrow{\Gamma \triangleright \Pi(\tilde{v})} C_2'}{C_1 \parallel C_2 \xrightarrow{\Gamma \triangleright \overline{\Pi}(\tilde{v})} C_1' \parallel C_2'} \textsc{ComL}$$

- Sync states that C_1 and C_2 can receive the same message.
- ComL governs communication between components C_1 and C_2.

$$\frac{C \xrightarrow{\Gamma \triangleright \overline{\Pi}(\tilde{v})} C' \quad f(\Gamma, \tilde{v}) = \Pi'}{[\ C\ ]^{\triangleright f} \xrightarrow{\Gamma \triangleright \overline{\Pi \wedge \Pi'}(\tilde{v})} [\ C'\ ]^{\triangleright f}} \textsc{ResO} \qquad \frac{C \xrightarrow{\Gamma \triangleright \Pi \wedge \Pi'(\tilde{v})} C' \quad f(\Gamma, \tilde{v}) = \Pi'}{[\ C\ ]^{\triangleleft f} \xrightarrow{\Gamma \triangleright \Pi(\tilde{v})} [\ C'\ ]^{\triangleleft f}} \textsc{ResI}$$

- ResO: if C evolves to C' via $\Gamma \triangleright \overline{\Pi}(\tilde{v})$ and $f(\Gamma, \tilde{v}) = \Pi'$ then $[\ C\ ]^{\triangleright f}$ evolves via $\Gamma \triangleright \overline{\Pi \wedge \Pi'}(\tilde{v})$ to $[\ C'\ ]^{\triangleright f}$.
- ResI: $[\ C\ ]^{\triangleleft f}$ will receive $\tilde{v}$ and evolve to $[\ C'\ ]^{\triangleleft f}$ with a label $\Gamma \triangleright \Pi(\tilde{v})$ only when $C \xrightarrow{\Gamma \triangleright \Pi \wedge \Pi'(\tilde{v})} C'$ where $f(\Gamma, \tilde{v}) = \Pi'$.

SCHOOL FOR ADVANCED STUDIES LUCCA

Behavioural Theory for AbC

Observable Barbs

Let $C{\downarrow_\Pi}$ mean that component C can send a message with a predicate $\Pi' \doteq \Pi$ (i.e., $C \xrightarrow{\nu\tilde{x}\overline{\Pi'}\tilde{v}}$ where $\Pi' \doteq \Pi$ and $\Pi' \not\simeq \text{ff}$). We write $C \Downarrow_\Pi$ if $C \rightarrow\!\!\!\triangleright^* C' \downarrow_\Pi$.

Barb Preservation

$\mathcal{R}$ is barb-preserving iff for every $(C_1, C_2) \in \mathcal{R}$, $C_1{\downarrow_\Pi}$ implies $C_2 \Downarrow_\Pi$

Weak Reduction Barbed Congruence Relations

A Weak Reduction Barbed Relation is a symmetric relation $\mathcal{R}$ over the set of AbC-components which is barb-preserving, reduction-closed, and context-closed.

Barbed Bisimilarity

Two components are weakly reduction barbed congruent, written $C_1 \cong C_2$, if $(C_1, C_2) \in \mathcal{R}$ for some weak reduction barbed congruent relation $\mathcal{R}$.

Full Abstraction

Weak Bisimulation

A symmetric binary relation $\mathcal{R}$ over the set of AbC-components is a *weak bisimulation* if and only if for any $(C_1, C_2) \in \mathcal{R}$ and for any λ_1

$$C_1 \xrightarrow{\lambda_1} C'_1 \text{ implies } \exists \lambda_2 : \lambda_1 \doteqdot \lambda_2 \text{ such that } C_2 \overset{\widehat{\lambda_2}}{\Longrightarrow} C'_2 \text{ and } (C'_1, C'_2) \in \mathcal{R}$$

Two components C_1 and C_2 are weakly bisimilar, written $C_1 \approx C_2$ if there exists a weak bisimulation $\mathcal{R}$ relating them.

Theorem (Soundness)

$C_1 \approx C_2$ implies $C_1 \cong C_2$, for any two components C_1 and C_2.

Theorem (Completeness)

$C_1 \cong C_2$ implies $C_1 \approx C_2$, for any two components C_1 and C_2.

Encoding other paradigms

A number of alternative communication paradigms can be easily modelled by relying on AbC primitives.

Explicit Message Passing

A $b\pi$-calculus process P is rendered as an AbC component $\Gamma{:}P$ where $\Gamma = \emptyset$ and the communication channel is sent as a part of the transmitted values with the receiver checking its compatibility.

Group based Communications

The group name is encoded as an attribute in AbC. The constructs for joining or leaving a given group can be encoded as attribute updates.

Publish-Subscribe

A Publisher sends tagged messages for all subscribers by exposing from his environment only the current topic while subscribers check compatibility of messages according to their subscriptions.

Implementations issues

Many challenges:

- Which kind of Middleware?
  - Centralized?
  - Distributed?
- Whom checks the predicates?
  - the sender?
  - the receiver?
  - a central entities?
- For the moment: four implementations
  - one in Java
  - two in Erlang
  - one in Go

AbC implementations

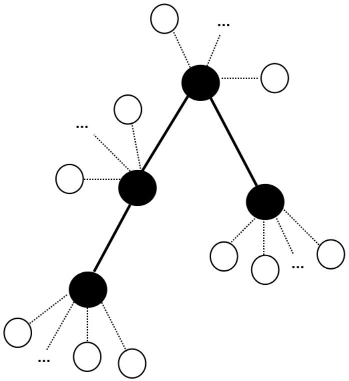
- AbaCus - Java: a centralized broker, broadcast, missed performance evaluation [ISOLA'16]
- AErlang - Erlang: a centralized broker with different dispatching policies [COORD'17]
  - Broadcast: Receivers checks both sending and receiving predicates
  - Push: broker checks sending predicates, receivers check receiving predicates
  - Pull: broker checks receiving predicates, receivers check sending predicates
  - Push-pull: broker checks both sending and receiving predicates

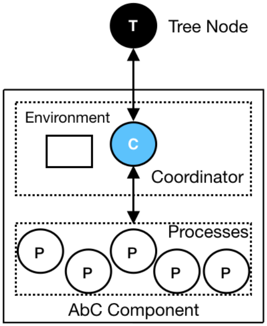
  dynamically handling messages, good performance, **deviated** semantics
- GoAt - Go: a set of broker connected in different shapes [ISOLA'18]
  - Semantics preserving implementation
  - Performance evaluation showed a tree-based structure performs best
  - However, deriving Goat code from AbC code is not immediate

ABEL - A programming framework for AbC

ABEL - Erlang) is a recent implementation of AbC combining previous experience [COORD'19]

- Providing Inter-coordinators (tree-based) and intra-coordinators interaction
- Supporting total-ordering and relaxed ordering of message delivery



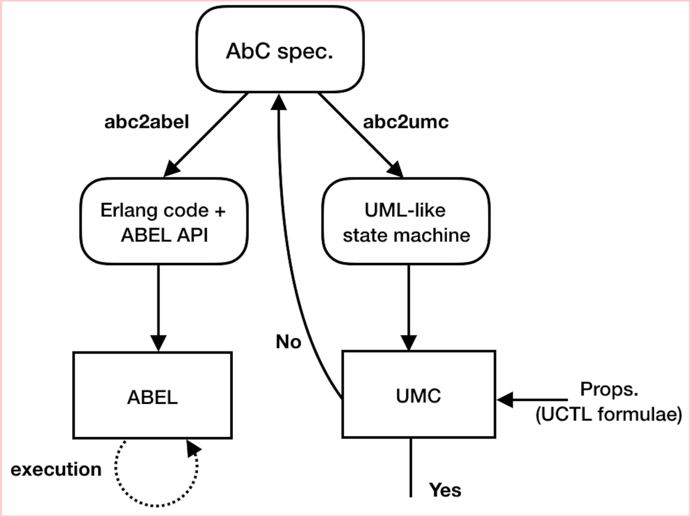


ABEL - A programming framework for AbC

ABEL API offers a one-to-one correspondence with AbC constructs

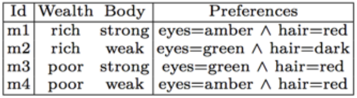
C	$::=$	$\textbf{new\_component}(\mathit{comp\_name}, \mathit{Env}, I)$	Create
		$\textbf{start\_component}(C, \mathit{BRef})$	Start
BDef	$::=$	$\mathit{proc}(\mathrm{C}, \langle \mathit{vars} \rangle) \to \mathit{Com}.$	Definition
BRef	$::=$	$\textbf{fun}(\langle \mathit{vars} \rangle) \to \mathit{proc}(C, \langle \mathit{vars} \rangle)\ \textbf{end}$	Reference
		$\mid \textbf{nil}$	
Act	$::=$	$\{\langle g \rangle, m, s, \langle u \rangle\}$	Output
		$\mid \{\langle g \rangle, r, \langle u \rangle\}$	Input
Com	$::=$	$\textbf{prefix}(\mathrm{C}, \{\mathit{Act}, \mathit{BRef}\})$	Prefix
		$\mid \textbf{choice}(\mathrm{C}, [\{\mathit{Act}, \mathit{BRef}\}])$	Choice
		$\mid \textbf{parallel}(\mathrm{C}, [\mathit{BRef}])$	Parallel
		$\mid \textbf{call}(\mathrm{C}, \mathit{BRef})$	Call

A model-driven approach to AbC programming

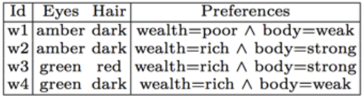


An example: Stable Marriage with Attributes

- Match men and women based on their preferences on partner's attributes
  - attributes: agents characteristics
  - preferences: interested values of partners attributes
  - An examples with 2 attributes and 2 preferences



Id	Wealth	Body	Preferences
m1	rich	strong	eyes=amber ∧ hair=red
m2	rich	weak	eyes=green ∧ hair=dark
m3	poor	strong	eyes=green ∧ hair=red
m4	poor	weak	eyes=amber ∧ hair=red



Id	Eyes	Hair	Preferences
w1	amber	dark	wealth=poor ∧ body=weak
w2	amber	dark	wealth=rich ∧ body=strong
w3	green	red	wealth=rich ∧ body=strong
w4	green	dark	wealth=rich ∧ body=weak

- Man: iteratively proposes while gradually relaxing expectations (predicates)
- Woman: performs "select and swaps"

Stable Marriage with Attributes in AbC

Two types of components M and W:

- $M_i \triangleq \Gamma_{mi} :_{\{id,w,b,pe,ph,\ldots\}} P_M$
- $W_j \triangleq \Gamma_{wj} :_{\{id,e,h,pw,pb,\ldots\}} P_W$

Specification for M: $P_M \triangleq Q \mid P \mid A \mid R$

$$\text{(Proposing) } P \triangleq \langle partner = 0 \wedge send = 1 \wedge \ldots \rangle \\ (\text{`}propose\text{'}, this.id, \widetilde{msg})@(\Pi).[\mathrm{send} := 0]P + \ldots$$

$$\text{(Positive answer) } A \triangleq (x = \text{`}yes\text{'})(x,y).(H(y) \mid A)$$

$$H(y) \triangleq (\langle partner = 0 \rangle(\text{`}confirm\text{'})@(id = y).[partner := y]0 \\ + \langle partner > 0 \rangle(\text{`}toolate\text{'})@(id = y).0$$

$$\text{(Rejection answer) } R \triangleq (x = \text{`}split\text{'})(x,y).[send := 1, partner := 0, \ldots]R \\ + (x = \text{`}no\text{'})(x,y).[send := 1, \ldots]R$$

Writing AbC in ABEL

$$
\begin{aligned}
R \triangleq\ & (x = \text{`}split\text{'})(x,y).[\ldots, send := 1, partner := 0]R \\
& + (x = \text{`}no\text{'})(x,y).[\ldots, send := 1]R
\end{aligned}
$$

$$
\begin{aligned}
&\mathbf{r}(\mathbf{C}) \rightarrow \\
&\qquad \ldots \textit{defining actions and continuations} \\
&\qquad Ref\ = fun() \rightarrow r(C)\ end, \\
&\qquad \mathbf{choice}(\mathbf{C}, [\{\mathbf{Act1}, \mathbf{Ref}\}, \{\mathbf{Act2}, \mathbf{Ref}\}]).
\end{aligned}
$$

$$
\begin{aligned}
RP1 =& fun(L, M, R) \rightarrow \\
&\qquad size(M) == 2\ andalso\ msg(1, M) == \text{'}split\text{'} \\
&\ end \\
U1 =& [\{send, 1\}, \{partner, 0\}, \ldots] \\
Act1 =& \{RP1, U1\}
\end{aligned}
$$

SMA properties checking with UMC

We verified for all input spaces of problems of size of 2

- Termination - `True`
- Soundness of outcomes:
  - completeness - `True`
  - symmetry - `True`
  - uniqueness - `False`
- Liveness properties:
  - If a woman sends ‘yes’ she will eventually receive a ‘toolate’ or ‘confirm’ message - `True`
  - If a man receives a ‘split’, he will eventually send a new proposal - `False` (he may immediately receive another ’yes’, and settle down)

Thank you!