Forcing,
Transition Algebras,
and Calculi

Go Hashimoto and Daniel Găină (IMI, Japan)
Ionuț Țuțu (IMAR, Romania)

2nd Workshop on Logic, Algebra and Category Theory
Fukuoka, 2025

In this talk

1. Short presentation of transition algebra
2. Applicability to process calculi
3. Proof system, soundness, completeness via forcing
4. Tool support and introduction to SpeX

Transition algebra (TA)

* at a glance, yet another logic used to reason about labelled transition systems

* deserves further examination thanks to a blend of special features…

Transition algebra (TA)

* at a glance, yet another logic used to reason about labelled transition systems

* deserves further examination thanks to a blend of special features:

  - provides support both for the static, structural aspects of systems, via equations, and for the dynamic aspects of systems, via transitions

  - uniform treatment of states and transition labels (in particular, quantification over labels)

  - unrestricted use of equations and transitions

  - increased expressivity by employing actions similar to those found in dynamic logics

  - operational semantics (more to follow)

TA signatures

* ordinary algebraic signatures
* pairs (S, F), where:
  - S is a set of so-called sorts
  - F is an $S^* \times S$-indexed family of sets $F_{w \to s}$ of operation symbols of arity w and sort s
* as usual, we also write

$$\sigma : w \to s \in F$$

in place of $\sigma \in F_{w \to s}$

Models

* (S, F)-algebras A interpreting:
  - every sort $s \in S$ as a set A_s
  - every operation symbol $\sigma : w \to s \in F$ as a function $\sigma^A : A_w \to A_s$
  - for any sort $t \in S$, every element $e \in A_t$ as a binary S-sorted relation $(e_s \subseteq A_s \times A_s)_{s \in S}$

Sentences

* built using standard Boolean connectives and quantifiers from two kinds of atoms:

  equations $t = t'$

  transitions $t \, a \, t'$

  where t and t' are terms having the same sort and a is an action

* actions are built from terms using

  sequential composition $a \mathbin{;} b$

  choice $a \cup b$

  iteration a^*

Semantics

* $A \vDash t = t'$ when $t^A = t'^A$
* $A \vDash t\ a\ t'$ when $(t^A, t'^A) \in a^A$

and so on, where

* $(a \mathbin{⨾} b)^A = a^A \mathbin{⨾} b^A$
* $(a \cup b)^A = a^A \cup b^A$
* $(a^*)^A = (a^A)^* = \bigcup\{(a^A)^n \mid n \in \mathbb{N}\}$

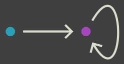
Quiz time

Which of the following models satisfies $\forall y \cdot \exists! z \cdot y \to z$?

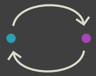
Quiz time

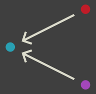
Which of the following models satisfies
$\forall y \cdot \exists! z \cdot y \rightarrow z$?

a) 

b) 

c) 

d) 

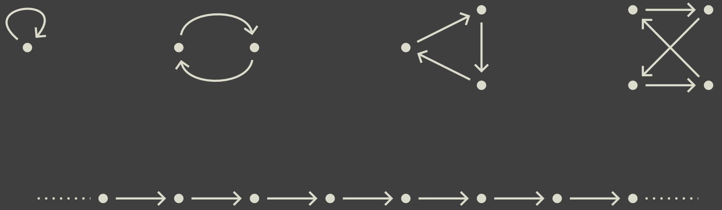
e) 

Quiz time

How do the models of $\forall y \cdot \exists! z \cdot y \to z \land \exists! x \cdot x \to y$ look like?

Quiz time

How do the models of $\forall y \cdot \exists! z \cdot y \to z \land \exists! x \cdot x \to y$ look like?



Quiz time

What if we add one of the following constraints?

* $\forall x, x' \cdot x \to^* x'$
* $\forall x, x' \cdot x \to^* x' \vee x' \to^* x$

Application to process calculi

Demo in

```
 ___          __  __
/ __|_ __  ___\ \/ /
\__ \ '_ \/ -_)>  <
|___/ .__/\___/_/\_\
    |_|
```

Syntactic entailment

* get to $\Gamma \vDash \varphi$ via $\Gamma \vdash \varphi$
* where $\vdash$ is defined by proof rules of the following form:

$$\frac{\Gamma \vdash t = t'}{\Gamma \vdash t' = t} \qquad \frac{\Gamma \vdash t\ a\ t', \quad \Gamma \vdash t'\ b\ t''}{\Gamma \vdash t\ (a \mathbin{;} b)\ t''} \qquad \frac{\Gamma \vdash t\ a\ t'}{\Gamma \vdash t\ (a \cup b)\ t'}$$

$$\frac{\Gamma \vdash t\ (a \mathbin{;} b)\ t'', \quad \Gamma \cup \{t\ a\ x, x\ b\ t'\} \vdash \varphi}{\Gamma \vdash \varphi}$$

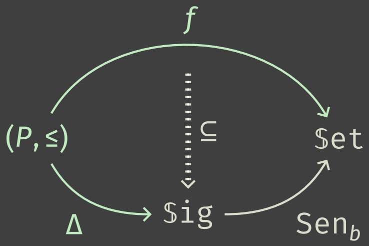
$$\frac{\Gamma \vdash \neg\neg\varphi}{\Gamma \vdash \varphi} \qquad \frac{\Gamma \cup \{\varphi\} \vdash \bot}{\Gamma \vdash \neg\varphi} \qquad \dots$$

Nota bene

* $\vdash$ is ω_1-compact whereas $\models$ is not so for uncountable signatures
* therefore, we cannot hope for a general completeness result for TA

* for at most countable signatures, neither $\vdash$ nor $\models$ is compact
* so we cannot tackle completeness using the classical Henkin method
* which leads us to forcing…

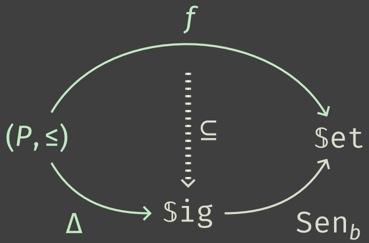
Forcing properties



where

* $(P, \leq)$ is a poset of so-called conditions
* Δ maps $p \leq q$ to $\Delta_p \subseteq \Delta_q$, and similarly for f
* $f(p) \subseteq \mathrm{Sen}_b(\Delta_p)$
* $f(p) \vDash \varphi$ implies $\varphi \in f(q)$ for some $q \geq p$

Syntactic forcing



where

* $p = (\Delta_p, \Gamma_p)$ where $\Delta_p = \Sigma \cup C_p$ and $\Gamma_p \subseteq \mathrm{Sen}(\Delta_p)$ consistent
* $p \leq q$ iff $\Delta_p \subseteq \Delta_q$ and $\Gamma_p \subseteq \Gamma_q$
* $\Delta(p) = \Delta_p$
* $f(p) = \Gamma_p \cap \mathrm{Sen}_b(\Delta_p)$

Forcing relation

* $p \Vdash \varphi$ when $\varphi \in f(p)$ for atomic sentences
* $p \Vdash t\,(a \fatsemi b)\,t'$ when $p \Vdash t\,a\,\tau$ and $p \Vdash \tau\,b\,t'$
  for some Δ_p-term τ
* $p \Vdash t\,(a \cup b)\,t'$ when $p \Vdash t\,a\,t'$ or $p \Vdash t\,b\,t'$
* $p \Vdash t\,a^*\,t'$ when $p \Vdash t\,a^n\,t'$ for some $n \in \mathbb{N}$
* $p \Vdash \neg\varphi$ when $q \nVdash \varphi$ for all $q \geq p$
* $p \Vdash \bigvee \Phi$ when $p \Vdash \varphi$ for some $\varphi \in \Phi$
* $p \Vdash \exists x \cdot \varphi$ when $p \Vdash \theta(\varphi)$ for some substitution θ

Forcing properties

* $p \Vdash \neg\neg\varphi$ iff for all $q \geq p$
  there is $r \geq q$ such that $r \Vdash \varphi$
* if $p \leq q$ and $p \Vdash \varphi$ then $q \Vdash \varphi$
* if $p \Vdash \varphi$ then $p \Vdash \neg\neg\varphi$
* we cannot have both $p \Vdash \varphi$ and $p \Vdash \neg\varphi$
* $\Gamma_p \vdash \varphi$ iff $p \Vdash \neg\neg\varphi$

Generic sets and models

Theorem

1. Every $p \in P$ belongs to a generic set $G \subseteq P$.

* G is an ideal
* for all $q \in G$ and sentences φ there is $r \in G$ such that $r \geq q$ and either $r \Vdash \varphi$ or $r \Vdash \neg\varphi$

Generic sets and models

Theorem

1. Every $p \in P$ belongs to a generic set $G \subseteq P$.

* G is an ideal
* for all $q \in G$ and sentences φ there is $r \in G$ such that $r \geq q$ and either $r \Vdash \varphi$ or $r \Vdash \neg\varphi$

2. If $p \in G$ and G is generic, then $\bigcup\{\Gamma_q \mid q \in G\}$ is a maximally consistent set that includes Γ_p.

Generic sets and models

Theorem

1. Every $p \in P$ belongs to a generic set $G \subseteq P$.

* G is an ideal
* for all $q \in G$ and sentences φ there is $r \in G$ such that $r \geq q$ and either $r \Vdash \varphi$ or $r \Vdash \neg\varphi$

2. If $p \in G$ and G is generic, then $\bigcup\{\Gamma_q \mid q \in G\}$ is a maximally consistent set that includes Γ_p.

3. G admits a countable and reachable generic model A.

* $A \vDash \varphi$ iff $q \Vdash \varphi$ for some $q \in G$

Completeness

Theorem
1. Every consistent set of sentences has a countable model.

Completeness

Theorem

1. Every consistent set of sentences has a countable model.

2. $\Gamma \vdash \varphi$ iff $\Gamma \vDash \varphi$.

Completeness

Theorem

1. Every consistent set of sentences has a countable model.

2. $\Gamma \vdash \varphi$ iff $\Gamma \vDash \varphi$.

* suppose ad absurdum $\Gamma \nvdash \varphi$

Completeness

Theorem

1. Every consistent set of sentences has a countable model.

2. $\Gamma \vdash \varphi$ iff $\Gamma \vDash \varphi$.

* suppose ad absurdum $\Gamma \nvdash \varphi$
* we get $\Gamma \nvdash \neg\neg\varphi$, hence $\Gamma \cup \{\neg\varphi\} \nvdash \bot$

Completeness

Theorem
1. Every consistent set of sentences has a countable model.

2. $\Gamma \vdash \varphi$ iff $\Gamma \vDash \varphi$.

* suppose ad absurdum $\Gamma \nvdash \varphi$
* we get $\Gamma \nvdash \neg\neg\varphi$, hence $\Gamma \cup \{\neg\varphi\} \nvdash \bot$
* it follows that $\Gamma \cup \{\neg\varphi\}$ is consistent, so it has a model

Completeness

Theorem

1. Every consistent set of sentences has a countable model.

2. $\Gamma \vdash \varphi$ iff $\Gamma \vDash \varphi$.

* suppose ad absurdum $\Gamma \nvdash \varphi$
* we get $\Gamma \nvdash \neg\neg\varphi$, hence $\Gamma \cup \{\neg\varphi\} \nvdash \bot$
* it follows that $\Gamma \cup \{\neg\varphi\}$ is consistent, so it has a model
* thus contradicting $\Gamma \vDash \varphi$

```
         ___          __  __
        / __|_ __  ___\ \/ /
        \__ \ '_ \/ -_)>  <
Back to |___/ .__/\___/_/\_\
            |_|
```

Term rewriting as a substructure for specification-language interpreters

* solid mathematical foundation
* algebraic, close to standard notation used by working theoretical computer scientists
* great for rapid prototyping

but

* still somewhat rigid as a meta-language
* limited support for modularization

Object-based programming

* we rewrite configurations
  multisets of ↲

  1. objects   < Id : Class | Attributes >
  2. messages   message(To, From, Arguments)

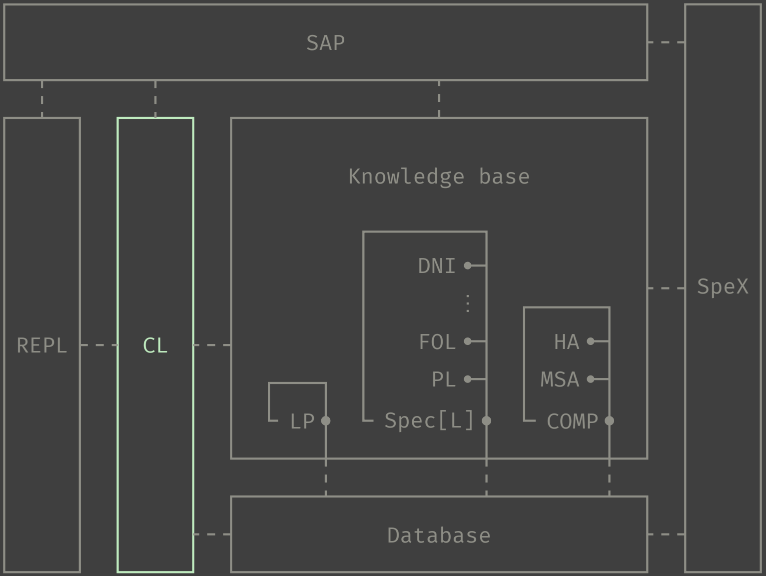
* using rules of the form

```
rl < Id : Class | ... >
   message(Id, ... )
 ⇒ ...
```

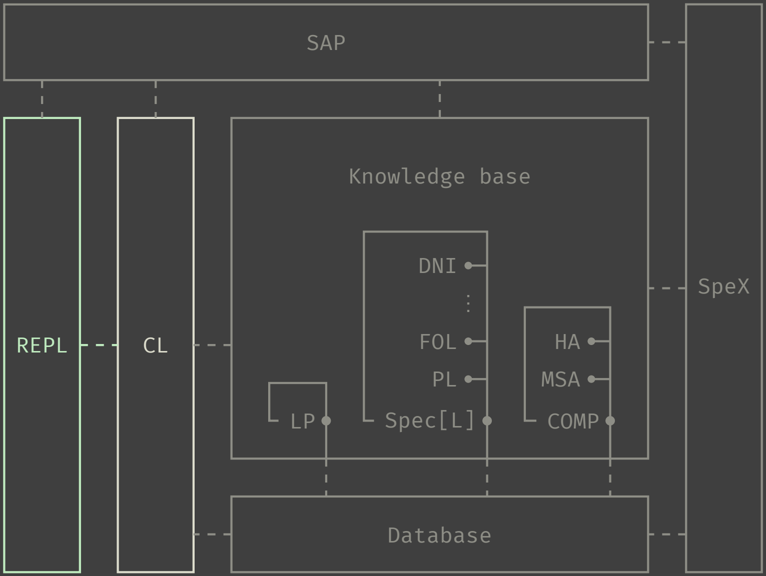
SpeX

* not a plain interpreter, but an ‘environment’
* integrates specification-language processors
* language agnostic
* offers a basic system UI ‘for free’
* based on Maude 3 (OBP with external rewrites)

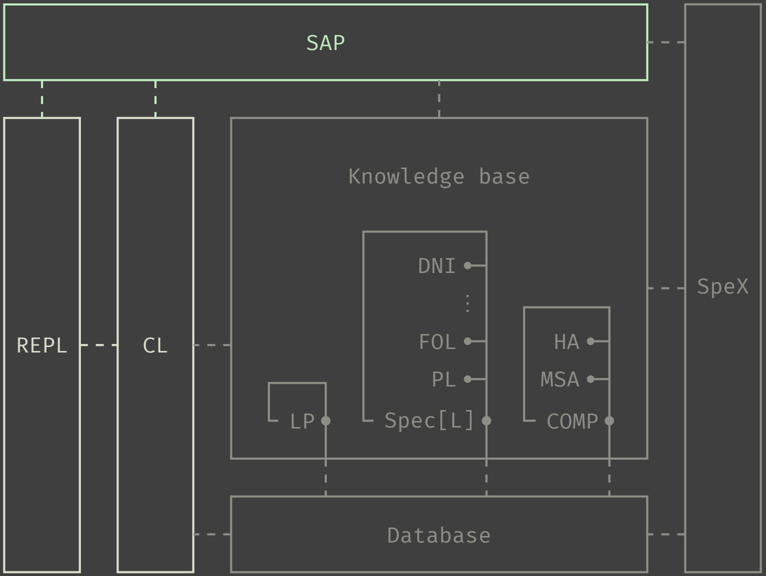
System overview (overly simplified)



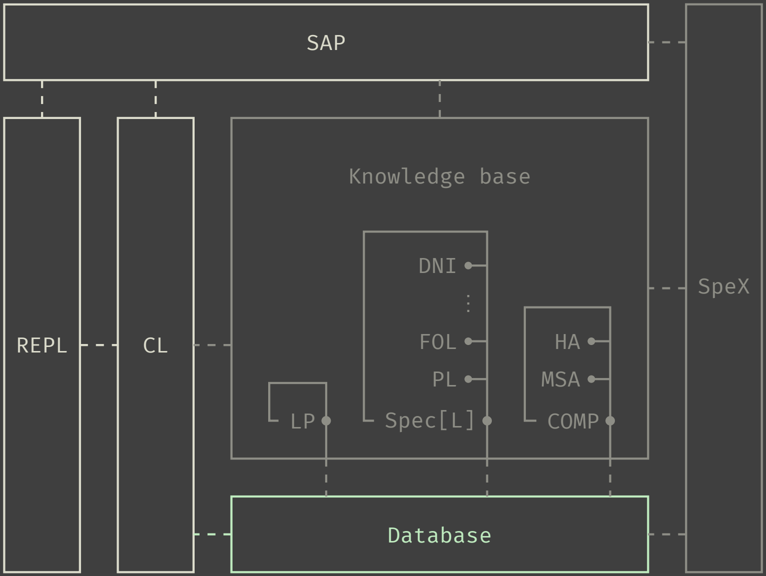
System overview (overly simplified)



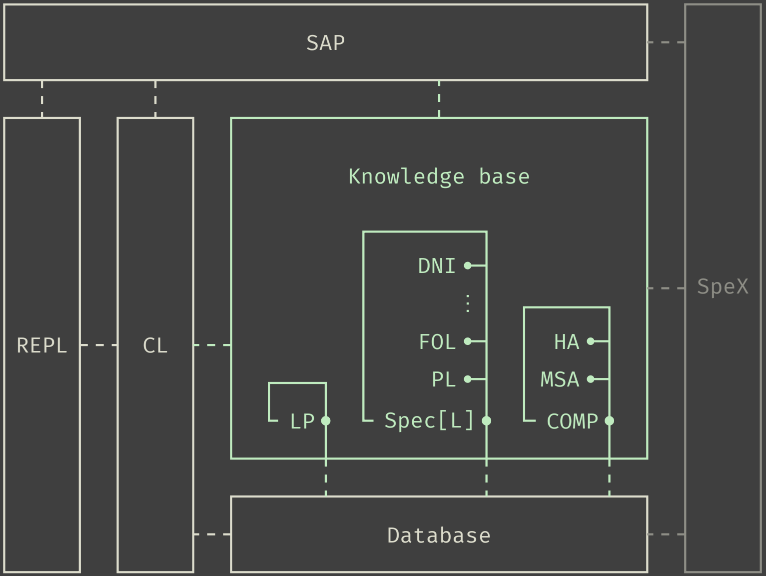
System overview (overly simplified)



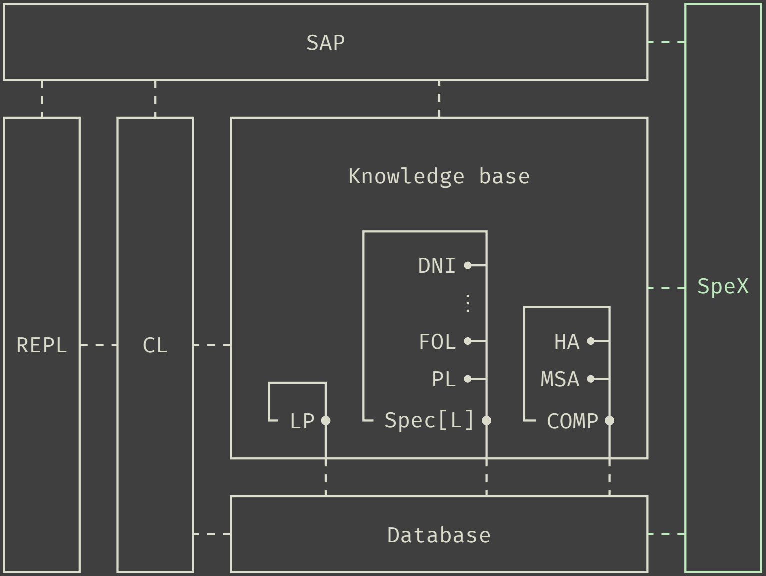
System overview (overly simplified)



System overview (overly simplified)



System overview (overly simplified)



Integrating new languages into SpeX

* by means of processors
  - objects of class PROC
  - interact with SpeX (the object)

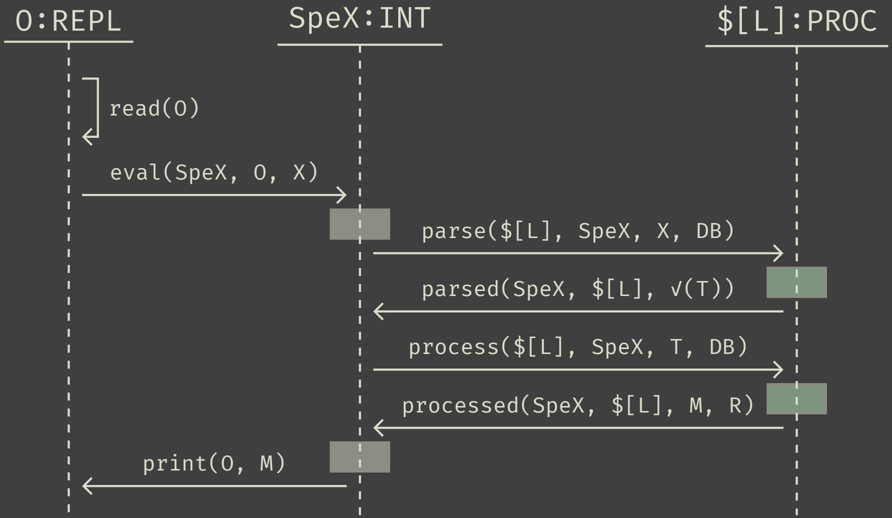
* receive messages of the form

```
parse($[L], SpeX, Input, DB)
process($[L], SpeX, AnnotatedTerm, DB)
```

* reply with messages of the form

```
parsed(SpeX, $[L], ParsingOutcome)
processed(SpeX, $[L], Text, Record)
```

A basic execution scenario



Example: Spec[DNI] (codev'd J. Fiadeiro and C. Chiriță)

```
spec Bind is
  including Base .
  mod __bind_ : Protein Organelle ⋋ Coat .
  ...
  ax store k:Nominal
     forall-local {p:Protein, o:Organelle}
     [ p o bind z:Coat ]
     (forall-local {o':Organelle}
       brane(o') = @(k) brane(o))
     and
     (forall-local {c':Coat}
       c' = z implies brane(c') = @(k) brane(o))
     [label: bind-effect] .
endspec
```

Example: Spec[DNI] (codev'd J. Fiadeiro and C. Chiriță)

```
spec Bind is
  including Base .
  mod __bind_ : Protein Organelle ⋊ Coat .
  …
  ax store k:Nominal
     forall-local {p:Protein, o:Organelle}
     [ p o bind z:Coat ]
     (forall-local {o':Organelle}
       brane(o') = @(k) brane(o))
     and
     (forall-local {c':Coat}
       c' = z implies brane(c') = @(k) brane(o))
     [label: bind-effect] .
endspec
```

Example: Spec[DNI] (codev'd J. Fiadeiro and C. Chiriță)

```
spec Bind is
  including Base .
  mod __bind_ : Protein Organelle ⋉ Coat .
  ...
  ax store k:Nominal
     forall-local {p:Protein, o:Organelle}
     [ p o bind z:Coat ]
     (forall-local {o':Organelle}
       brane(o') = @(k) brane(o))
     and
     (forall-local {c':Coat}
       c' = z implies brane(c') = @(k) brane(o))
     [label: bind-effect] .
endspec
```

Example: Spec[DNI] (codev'd J. Fiadeiro and C. Chiriță)

```
spec Bind is
  including Base .
  mod __bind_ : Protein Organelle ⋊ Coat .
  ...
  ax store k:Nominal
     forall-local {p:Protein, o:Organelle}
     [ p o bind z:Coat ]
     (forall-local {o':Organelle}
       brane(o') = @(k) brane(o))
     and
     (forall-local {c':Coat}
       c' = z implies brane(c') = @(k) brane(o))
     [label: bind-effect] .
endspec
```

Example: COMP (codev'd R. Diaconescu)

```
bobj WATCH is
  syncing (UP-TO-24-COUNTER as HOUR)
      and (UP-TO-60-COUNTER as MINUTE)
      and (UP-TO-60-COUNTER as SECOND) .
  op _:_:_ : Nat Nat Nat → State .
  act tick_ : State → State .
  act inc-min_ : State → State .
  …
endbo

open WATCH
  check tick inc-min (H:Nat : M:Nat : S:Nat)
      ~ inc-min tick (H:Nat : M:Nat : S:Nat)
  forall M:Nat < 60 = true
     and S:Nat < 60 = true .
close
```

Example: COMP (codev'd R.Diaconescu)

```
bobj WATCH is
  syncing (UP-TO-24-COUNTER as HOUR)
      and (UP-TO-60-COUNTER as MINUTE)
      and (UP-TO-60-COUNTER as SECOND) .
  op _:_:_ : Nat Nat Nat → State .
  act tick_ : State → State .
  act inc-min_ : State → State .
  …
endbo

open WATCH
  check tick inc-min (H:Nat : M:Nat : S:Nat)
      ~ inc-min tick (H:Nat : M:Nat : S:Nat)
  forall M:Nat < 60 = true
     and S:Nat < 60 = true .
close
```

Example: IPDL (dev'd K. Sojakova,
M. Codescu,
and J. Gancher)

```
protocol real =
  newfamily SendInShare[bound N + 2 bound N + 2
                        dependentBound I]
            indices: m, n, i ... : bool in
  newfamily OTMsg-0[bound N + 2 bound N + 2
                    bound K ]
            indices: n, m, k ... : bool in
   ...
  parties || 1OutOf4OTReal
  where parties = ...
    and 1OutOf4OTReal = ...
```

See https://arxiv.org/abs/2507.22705

Example: Spec[TA] (codev'd D. Găină and A. Riesco)

```
spec Institute is
  including CCS .

  ops theorem, coffee, coin ...
  ops Institute, Mathematician, CoffeeVM ...

  ax Institute
     = (Mathematician | CoffeeVM) \ coin \ coffee .
  ax Mathematician
     =(tau)> (snd(coin) . rcv(coffee) .
              snd(theorem) . Mathematician) .
  ax CoffeeVM
     =(tau)> (rcv(coin) . snd(coffee) . CoffeeVM) .
endspec
```

Obtaining SpeX and TATP

* from the git repositories:

  https://gitlab.com/ittutu/spex

  https://github.com/Transition-Algebra/TATP

* then, provided Maude 3(>.2) is installed:

```
./configure
make
[sudo] make install
```

Happy hacking!