EB[ASTD]: Meta-modelling framework for ASTD

Christophe Chen, Peter Rivière, Neeraj Kumar Singh,
Guillaume Dupont, Yamine Ait Ameur, Marc Frappier

Rodin WorkShop 2025
(Slides from FormaliSE2025)

27 April 2025

Outline

Motivation

ASTD is a graphical notation that combines

- Statechart : hierarchy, AND-states, OR-states..
- Process algebra : concurrency, compositionality

which allows

- **modularity**
- **expressiveness**
- adding variables
- an operational semantics

How to reason on ASTD models ?

Outline

ASTD – Algebraic State Transition Diagram

ASTD is :

- a graphical *state-based* formalism
- built *incrementaly* by applying **compositional** operator

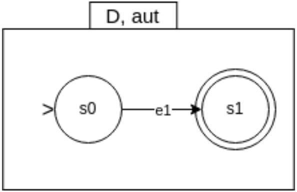


Figure – D = Automaton()

ASTD – Algebraic State Transition Diagram

ASTD is :

- a graphical *state-based* formalism
- built *incrementaly* by applying **compositional** operator
- equipped with *sequent-based* transition rules [5, 6]
- applied on intrusion detection and autonomous vehicle [14, 4, 11]

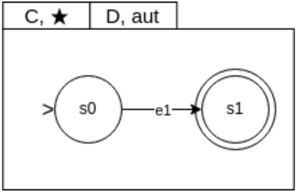


Figure – C = Closure(D)

ASTD – Algebraic State Transition Diagram

ASTD is :

- a graphical *state-based* formalism
- built *incrementaly* by applying **compositional** operator
- equipped with *sequent-based* transition rules [5, 6]
- applied on intrusion detection and autonomous vehicle [14, 4, 11]

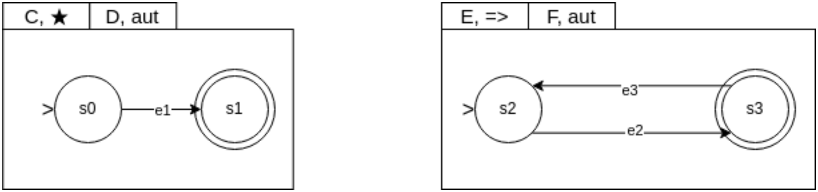


Figure – Two ASTDs C and E

ASTD – Algebraic State Transition Diagram

ASTD is :

- a graphical *state-based* formalism
- built *incrementaly* by applying **compositional** operator
- equipped with *sequent-based* transition rules [5, 6]
- applied on intrusion detection and autonomous vehicle [14, 4, 11]

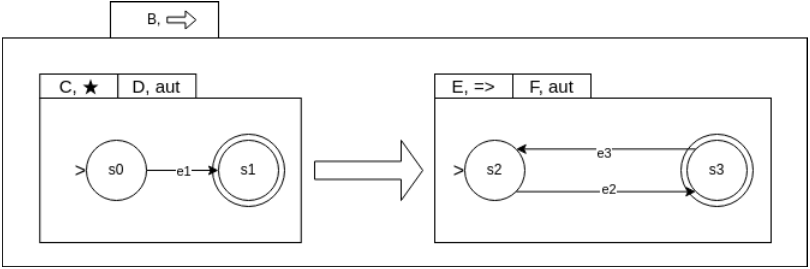


Figure – B = Sequence(C,E)

ASTD – Algebraic State Transition Diagram

ASTD is :

- a graphical *state-based* formalism
- built *incrementaly* by applying **compositional** operator
- equipped with *sequent-based* transition rules [5, 6]
- applied on intrusion detection and autonomous vehicle [14, 4, 11]

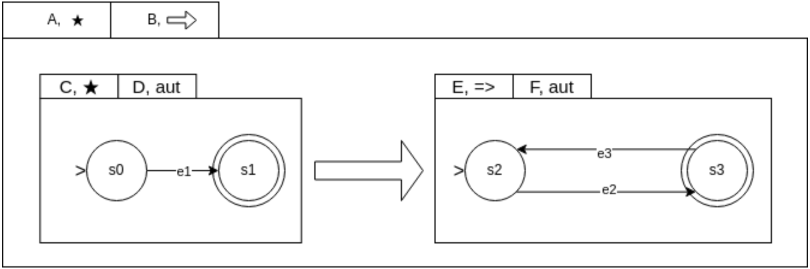


Figure – Final ASTD

Exploit an ASTD specification

Interesting **research directions** based on ASTD features

- graphical formalism $\Longrightarrow$ Visualization
- operational semantics $\Longrightarrow$ Animation
- state variables $\Longrightarrow$ Invariants
- state-based $\Longrightarrow$ Advanced properties (for eg. Liveness)

Exploit an ASTD specification

Interesting **research directions** based on ASTD features

- graphical formalism $\Longrightarrow$ Visualization
- operational semantics $\Longrightarrow$ Animation
- state variables $\Longrightarrow$ Invariants
- state-based $\Longrightarrow$ Advanced properties (for eg. Liveness)

All above need a developing environment for ASTD
but developing from scratch is cumbersome...

Exploit an ASTD specification

Interesting **research directions** based on ASTD features

- graphical formalism $\Longrightarrow$ Visualization
- operational semantics $\Longrightarrow$ Animation
- state variables $\Longrightarrow$ Invariants
- state-based $\Longrightarrow$ Advanced properties (for eg. Liveness)

All above need a developing environment for ASTD
but developing from scratch is cumbersome...
$\Longrightarrow$ Embedding ASTD in a unified modelling framework

Outline

Existing approaches for ASTDs

cASTD is a *compiler* that generates C++ code based on a given specification. [15, 3]

- + It can ***animate*** an ASTD for **validation** purpose
- - *Poor* animation features (no backtracking, traces, . . .)

ASTD2EB is a *model to text* transformation
where compositional operators become a **pattern** in Event-B. [12]

- + Rodin tool support and proof assistant
- - *Loss* of hierarchy structure

pASTD : provides definition of **invariants** and **POs**. It is a java plugin that transfer POs in Event-B [8]

- + easier to prove
- - loss of traceability
- - POs are defined (but no soundness)

Position of our work...

Despite advances in incorporating features, ASTD tools

- have primarily focused on **design** and **testing**
- based on ***ad hoc* model transformation**
- that lost **hierarchical structure** of the original ASTD

$\Longrightarrow$

- The power of ASTD is ***not* exploited** by target language
- Spec error is **difficult to trace back** to the original ASTD

Position of our work...

Despite advances in incorporating features, ASTD tools

- have primarily focused on **design** and **testing**
- based on ***ad hoc* model transformation**
- that lost **hierarchical structure** of the original ASTD

$\Longrightarrow$

- The power of ASTD is ***not* exploited** by target language
- Spec error is **difficult to trace back** to the original ASTD

Only manipulation of ASTD instances
$\Longrightarrow$ ***mechanize*** its semantics to manipulate ASTDs as first order objects for analysis (animation, ..), reasoning (proof,..)

Formalised embedding approaches

We are inspired from **meta-modelling** approaches, such as :

- MetaCoq project to build a meta-programming environment, implemented in CertiCoq [19, 2]
- Modelling HOL models in Isabelle/HOL [9]
- B models embedded in PVS [13]
- Event-B Machine encoded in CSP [10, 18]
- **EB4EB**, a reflexive framework for advanced **reasoning** and **analysis** on Event-B machine in Event-B [17, 16]

Formalised embedding approaches

We are inspired from **meta-modelling** approaches, such as :

- MetaCoq project to build a meta-programming environment, implemented in CertiCoq [19, 2]
- Modelling HOL models in Isabelle/HOL [9]
- B models embedded in PVS [13]
- Event-B Machine encoded in CSP [10, 18]
- **EB4EB**, a reflexive framework for advanced **reasoning** and **analysis** on Event-B machine in Event-B [17, 16]

In the same spirit of EB4EB, we *invite* the semantics of ASTD in Event-B as algebraic theory.

Outline

Modelling language – Event-B [1]

Correct-by-construction, state-based formal method, refinement. Two components : Contexts and Machines

CONTEXT Ctx **SETS** s **CONSTANTS** c **AXIOMS** A **THEOREMS** T_{ctx} **END**	**MACHINE** M **SEES** Ctx **VARIABLES** x **INVARIANTS** $I(x)$ **THEOREMS** $T_{mch}(x)$ **VARIANT** $V(x)$ **EVENTS** INITIALISATION $\widehat{=}$ ... $evt \ \widehat{=}$ **ANY** α **WHERE** $G(x, \alpha)$ **THEN** $x \mid BAP(\alpha, x, x')$ **END** **END**

Modelling language – Event-B [1]

Correct-by-construction, state-based formal method, refinement. Two components : Contexts and Machines

CONTEXT Ctx **SETS** s **CONSTANTS** c **AXIOMS** A **THEOREMS** T_{ctx} **END**	**MACHINE** M **SEES** Ctx **VARIABLES** x **INVARIANTS** $I(x)$ **THEOREMS** $T_{mch}(x)$ **VARIANT** $V(x)$ **EVENTS** INITIALISATION $\widehat{=}$ $\ldots$ $evt \ \widehat{=}$ **ANY** α **WHERE** $G(x, \alpha)$ **THEN** $x \mid BAP(\alpha, x, x')$ **END** **END**

- Automatic generation of POs.
- Rodin $\Longrightarrow$ Editor + POs generator + automatic/interactive provers

Modelling language – Event-B [1]

Correct-by-construction, state-based formal method, refinement. Two components : Contexts and Machines

CONTEXT Ctx **SETS** s **CONSTANTS** c **AXIOMS** A **THEOREMS** T_{ctx} **END**	**MACHINE** M **SEES** Ctx **VARIABLES** x **INVARIANTS** $I(x)$ **THEOREMS** $T_{mch}(x)$ **VARIANT** $V(x)$ **EVENTS** INITIALISATION $\hat{=}$ ... evt $\hat{=}$ **ANY** α **WHERE** $G(x, \alpha)$ **THEN** $x \mid BAP(\alpha, x, x')$ **END** **END**

- Automatic generation of POs.
- Rodin $\Longrightarrow$ Editor + POs generator + automatic/interactive provers

Theorems Context (*THM_ctx*)	$A \Rightarrow T_{ctx}$
Theorems machine (*THM_mch*)	$A \wedge I(x) \Rightarrow T_{mch}(x)$
Initialisation (INIT)	$A \wedge AP(\alpha, x') \Rightarrow I(x')$
Invariant preservation (INV)	$A \wedge I(x) \wedge G(x, \alpha) \wedge BAP(x, \alpha, x') \Rightarrow I(x')$
Event feasibility (FIS)	$A \wedge I(x) \wedge G(x, \alpha) \Rightarrow \exists x' \cdot BAP(x, \alpha, x')$
Variant progress (VAR)	$A \wedge I(x) \wedge G(x, \alpha) \wedge BAP(x, \alpha, x') \Rightarrow V(x') < V(x)$

Event-B extension [7]

Formalisation of new data-types for Event-B $\Longrightarrow$ Theories

```
THEORY Th
IMPORT Th1, ...




END
```

Event-B extension [7]

Formalisation of new data-types for Event-B $\Longrightarrow$ Theories

```
THEORY Th
IMPORT Th1, ...
TYPE PARAMETERS E, F, ...

DATATYPES
  Type2(E, ...)
  constructors
    cstr1(p1 : T1, ...)

OPERATORS
  Op1<nature> (p1 : T1, ...)

    direct definition D1

END
```

- Algebraic definition for data-types $\Longrightarrow$ Constructive definitions

Event-B extension [7]

Formalisation of new data-types for Event-B $\Longrightarrow$ Theories

```
THEORY Th
IMPORT Th1, ...
TYPE PARAMETERS E, F, ...

DATATYPES
  Type2(E, ...)
  constructors
    cstr1(p1 : T1, ...)

OPERATORS
  Op1<nature> (p1 : T1, ...)

    direct definition D1

AXIOMATIC DEFINITIONS
  TYPES A1, ...
  OPERATORS
   AOp2<nature> (p1 : T1, ...):Tr

  AXIOMS A1, ...

END
```

- Algebraic definition for data-types $\Longrightarrow$ Constructive definitions and/or axiomatic definitions.

Event-B extension [7]

Formalisation of new data-types for Event-B $\Longrightarrow$ Theories

THEORY Th
IMPORT Th1, ...
TYPE PARAMETERS E, F, ...

DATATYPES
Type2(E, ...)
constructors
cstr1($p_1 : T_1$, ...)

OPERATORS
Op1<nature> ($p_1 : T_1$, ...)
well–definedness $WD(p_1, \ldots)$
direct definition D_1

AXIOMATIC DEFINITIONS
TYPES A_1, ...
OPERATORS
AOp2<nature> ($p_1 : T_1$, ...):T_r
well–definedness $WD(p_1, \ldots)$
AXIOMS A_1, ...

END

- Algebraic definition for data-types $\Longrightarrow$ Constructive definitions and/or axiomatic definitions.
- Well-Definedness (WD) conditions (partial definitions).

Event-B extension [7]

Formalisation of new data-types for Event-B $\Longrightarrow$ Theories

```
THEORY Th
IMPORT Th1, ...
TYPE PARAMETERS E, F, ...

DATATYPES
  Type2(E, ...)
  constructors
    cstr1(p1 : T1, ...)

OPERATORS
  Op1<nature> (p1 : T1, ...)
    well–definedness WD(p1, ...)
    direct definition D1

AXIOMATIC DEFINITIONS
  TYPES A1, ...
  OPERATORS
   AOp2<nature> (p1 : T1, ...):Tr
    well–definedness WD(p1, ...)
  AXIOMS A1, ...

THEOREMS T1, ...
PROOF RULES. . .

END
```

- Algebraic definition for data-types $\Longrightarrow$ Constructive definitions and/or axiomatic definitions.
- Well-Definedness (WD) conditions (partial definitions).
- Relevant proved theorems.
- Relevant inference/rewrite rules.

Event-B extension [7]

Formalisation of new data-types for Event-B $\Longrightarrow$ Theories

THEORY Th
IMPORT Th1, ...
TYPE PARAMETERS E, F, ...

DATATYPES
Type2(E, ...)
constructors
cstr1($p_1 : T_1$, ...)

OPERATORS
Op1<nature> ($p_1 : T_1$, ...)
well–definedness $WD(p_1, \ldots)$
direct definition D_1

AXIOMATIC DEFINITIONS
TYPES A_1, ...
OPERATORS
AOp2<nature> ($p_1 : T_1$, ...)$: T_r$
well–definedness $WD(p_1, \ldots)$
AXIOMS A_1, ...

THEOREMS T_1, ...
PROOF RULES. . .

END

- Algebraic definition for data-types $\Longrightarrow$ Constructive definitions and/or axiomatic definitions.
- Well-Definedness (WD) conditions (partial definitions).
- Relevant proved theorems.
- Relevant inference/rewrite rules.
- Theory Plugin for Rodin environment.

Outline

ASTDStruct - Import the syntax

Constructive datatype.

- Statename, event, variable
- ASTD operators integrated
- Focus on Sequence

$P(x)$ denoted as $\{x|P(x)\}$
axm of comprehension

```
THEORY ASTDStruct
  TYPE PARAMETERS St, Ev, Var
  DATA TYPES
    ASTD(St, Ev, Var)
    constructors
    Elementary(...)
    Automaton(...)
    Sequence(
      SeqFirst : ASTD(St, Ev, Var),
      SeqSnd : ASTD(St, Ev, Var),
      SeqAttr : ℙ(Var),
      SeqInitAttr : ℙ(Var × ℤ) ↔ ℙ(Var × ℤ),
      SeqInv : ℙ(ℙ(Var × ℤ))
      )
    Closure(...)
    Guard(...)
```

ASTDStruct - Import the syntax

Constructive datatype.

- Statename, event, variable

```
THEORY ASTDStruct
  TYPE PARAMETERS St , Ev , Var
  DATA TYPES
    ASTD ( St , Ev , Var )
    constructors
    Elementary(...)
    Automaton(...)
    Sequence(
      SeqFirst : ASTD(St, Ev, Var) ,
      SeqSnd : ASTD(St, Ev, Var) ,
      SeqAttr : ℙ(Var) ,
      SeqInitAttr : ℙ(Var × ℤ) ↔ ℙ(Var × ℤ) ,
      SeqInv : ℙ(ℙ(Var × ℤ))
      )
    Closure(...)
    Guard(...)
```

ASTDStruct - Import the syntax

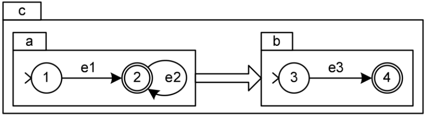
Constructive datatype.

- Statename, event, variable
- ASTD operators integrated

THEORY ASTDStruct
  TYPE PARAMETERS St , Ev , Var
  DATA TYPES
    ASTD(*St* , *Ev* , *Var*)
    constructors
    Elementary(...)
    Automaton(...)
    Sequence(
      $SeqFirst : ASTD(St, Ev, Var)$,
      $SeqSnd : ASTD(St, Ev, Var)$,
      $SeqAttr : \mathbb{P}(Var)$,
      $SeqInitAttr : \mathbb{P}(Var \times \mathbb{Z}) \leftrightarrow \mathbb{P}(Var \times \mathbb{Z})$,
      $SeqInv : \mathbb{P}(\mathbb{P}(Var \times \mathbb{Z}))$
)
    Closure(...)
    Guard(...)

ASTDStruct - Import the syntax

Constructive datatype.



- Statename, event, variable
- ASTD operators integrated
- Focus on Sequence

THEORY ASTDStruct
  TYPE PARAMETERS St , Ev , Var
  DATA TYPES
    ASTD (*St* , *Ev* , *Var*)
    constructors
    Elementary$(\ldots)$
    Automaton$(\ldots)$
    Sequence(
      SeqFirst : *ASTD*(*St*, *Ev*, *Var*) ,
      SeqSnd : *ASTD*(*St*, *Ev*, *Var*) ,
      SeqAttr : $\mathbb{P}(Var)$,
      SeqInitAttr : $\mathbb{P}(Var \times \mathbb{Z}) \leftrightarrow \mathbb{P}(Var \times \mathbb{Z})$,
      SeqInv : $\mathbb{P}(\mathbb{P}(Var \times \mathbb{Z}))$
)
    Closure$(\ldots)$
    Guard$(\ldots)$

ASTDStruct - Import the syntax

Constructive datatype.

- Statename, event, variable
- ASTD operators integrated
- Focus on Sequence

$P(x)$ denoted as $\{x|P(x)\}$
axm of comprehension

```
THEORY ASTDStruct
  TYPE PARAMETERS St, Ev, Var
  DATA TYPES
    ASTD(St, Ev, Var)
    constructors
    Elementary(...)
    Automaton(...)
    Sequence(
      SeqFirst : ASTD(St, Ev, Var),
      SeqSnd : ASTD(St, Ev, Var),
      SeqAttr : ℙ(Var),
      SeqInitAttr : ℙ(Var × ℤ) ↔ ℙ(Var × ℤ),
      SeqInv : ℙ(ℙ(Var × ℤ))
      )
    Closure(...)
    Guard(...)
```

ASTDStruct - Well definedness conditions

WD conditions are introduced in each ASTD operator

- Constraints on actions, scope, type..

- Group in a single operator

THEORY ASTDStruct
TYPE PARAMETERS St , Ev , Var
...
OPERATORS
Automaton_WellCons ***predicate*** $(a : ASTD(St, Ev, Var))$

ASTD_WellTyped ***predicate*** $(a : ASTD(St, Ev, Var))$

Scope_WellCons ***predicate*** $(a : ASTD(St, Ev, Var),$
$accVar : \mathbb{P}(Var))$

InitActionS_WellCons ***predicate*** $(a : ASTD(St, Ev, Var))$

InvariantS_WellCons ***predicate*** $(a : ASTD(St, Ev, Var),$
$accVar : \mathbb{P}(Var))$

ASTD_WellCons ***predicate***
$(a : ASTD(St, Ev, Var), accVar : \mathbb{P}(Var))$
direct definition
$Automaton\_WellCons(a)$
$\wedge ASTD\_WellTyped(a)$
$\wedge Scope\_WellCons(a, accVar)$
$\wedge InitActionS\_WellCons(a)$
$\wedge InvariantS\_WellCons(a, accVar)$

ASTDStruct - Well definedness conditions

WD conditions are introduced in each ASTD operator

- Constraints on actions, scope, type..

THEORY ASTDStruct
TYPE PARAMETERS St , Ev , Var
. . .
OPERATORS
Automaton_WellCons ***predicate*** $(a : ASTD(St, Ev, Var))$

ASTD_WellTyped ***predicate*** $(a : ASTD(St, Ev, Var))$

Scope_WellCons ***predicate*** $(a : ASTD(St, Ev, Var),$
$accVar : \mathbb{P}(Var))$

InitActionS_WellCons ***predicate*** $(a : ASTD(St, Ev, Var))$

InvariantS_WellCons ***predicate*** $(a : ASTD(St, Ev, Var),$
$accVar : \mathbb{P}(Var))$

ASTDStruct - Well definedness conditions

WD conditions are introduced in each ASTD operator

- Constraints on actions, scope, type..
- Group in a single operator

THEORY ASTDStruct
TYPE PARAMETERS St , Ev , Var
...
OPERATORS
Automaton_WellCons ***predicate*** $(a : ASTD(St, Ev, Var))$

ASTD_WellTyped ***predicate*** $(a : ASTD(St, Ev, Var))$

Scope_WellCons ***predicate*** $(a : ASTD(St, Ev, Var),$
$accVar : \mathbb{P}(Var))$

InitActionS_WellCons ***predicate*** $(a : ASTD(St, Ev, Var))$

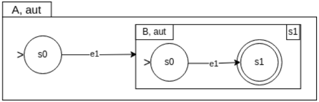
InvariantS_WellCons ***predicate*** $(a : ASTD(St, Ev, Var),$
$accVar : \mathbb{P}(Var))$

ASTD_WellCons ***predicate***
$(a : ASTD(St, Ev, Var), accVar : \mathbb{P}(Var))$
direct definition
$Automaton\_WellCons(a)$
$\land ASTD\_WellTyped(a)$
$\land Scope\_WellCons(a, accVar)$
$\land InitActionS\_WellCons(a)$
$\land InvariantS\_WellCons(a, accVar)$

ASTDBehavior - Transition rules

Inductive sequent-based operational semantics rules. (For Eg. aut_2)

NextState ***expression*** (
    astd : $ASTD(St, Ev, Var)$,
    current : $ASTDState(St, Var)$,
    $E_e : Var \nrightarrow \mathbb{Z}$)
well–defnedness condition
    $ASTD\_WellCons(astd, dom(E_e))$
recursive definition
case astd :
  Elementary(inv) => ...
  Automaton$(i, f, ..., mapping)$ =>
    $aut_1 \cup \mathbf{aut_2} \cup aut_3$
  Sequence$(fst, snd, attr, initAttr, inv)$
      => ....
  Closure(..) => ...
  Guard(...) => ...



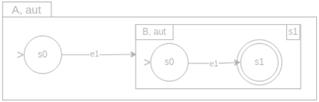
$$\frac{s \xrightarrow{\sigma, E_g, E'_g}_{a.\nu(n)} s' \qquad \begin{array}{l} E_g = E_e \lhd\!\!- E \\ E'_e = E_e \lhd\!\!- attr \lhd E'_g \\ E' = attr \lhd E'_g \end{array}}{(aut_\circ, n, E, s) \xrightarrow{\sigma, E_e, E'_e}_{a} (aut_\circ, n, E', s')}\ aut_2$$

$\{aut_\circ(n, E', s') \mapsto E'_e \mid n, E, s, s', E'_e, E_g, E'_g, E' \cdot$
2. $\wedge n = StateName(current)$
3. $\wedge s = AutSubState(current)$
4. $\wedge E = AutCache(current)$
5. $\wedge s' \mapsto E'_g \in NextState(mapping(n), \sigma, s, E_g)$
6. $\wedge E_g = E_e \lhd\!\!- E$
7. $\wedge E'_e = E_e \lhd\!\!- attr \lhd E'_g$
8. $\wedge E' = attr \lhd E'_g$

$\}$

ASTDBehavior - Transition rules

Inductive sequent-based operational semantics rules. (For Eg. aut_2)



NextState ***expression*** (
  astd : $ASTD(St, Ev, Var)$,
  current : $ASTDState(St, Var)$,
  $E_e : Var \mathbin{+\!\!\!\rightarrow} \mathbb{Z}$)
well–definedness condition
  $ASTD\_WellCons(astd, dom(E_e))$
recursive definition
case astd :
  Elementary(inv) => ...
  Automaton($i, f, \ldots, mapping$) =>
    $aut_1 \cup$ **aut₂** $\cup\, aut_3$
  Sequence($fst, snd, attr, initAttr, inv$)
      => . . . .
  Closure(..) => . . .
  Guard(...) => . . .

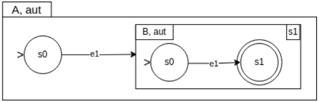
$$aut_2 \quad \frac{s \xrightarrow{\sigma, E_g, E_g'}_{a.\nu(n)} s' \qquad \begin{array}{l} E_g = E_e \mathbin{<\!\!+} E \\ E_e' = E_e \mathbin{<\!\!+} attr \mathbin{\lhd\!\!\!-} E_g' \\ E' = attr \lhd E_g' \end{array}}{(aut_\circ, n, E, s) \xrightarrow{\sigma, E_e, E_e'}_{a} (aut_\circ, n, E', s')}$$

$\{aut_\circ(n, E', s') \mapsto E_e' \mid n, E, s, s', E_e', E_g, E_g', E' \cdot$
2. $\wedge n = StateName(current)$
3. $\wedge s = AutSubState(current)$
4. $\wedge E = AutCache(current)$
5. $\wedge s' \mapsto E_g' \in NextState(mapping(n), \sigma, s, E_g)$
6. $\wedge E_g = E_e \mathbin{<\!\!+} E$
7. $\wedge E_e' = E_e \mathbin{<\!\!+} attr \mathbin{\lhd\!\!\!-} E_g'$
8. $\wedge E' = attr \lhd E_g'$

ASTDBehavior - Transition rules

Inductive sequent-based operational semantics rules. (For Eg. aut_2)

NextState ***expression*** (
astd : $ASTD(St, Ev, Var)$,
current : $ASTDState(St, Var)$,
E_e : $Var \nrightarrow \mathbb{Z}$)
well–definedness condition
$ASTD\_WellCons(astd, dom(E_e))$
recursive definition
case astd :
Elementary(*inv*) => ...
Automaton(*i*, *f*, ..., *mapping*) =>
$aut_1 \cup$ **aut₂** $\cup\ aut_3$
Sequence(*fst*, *snd*, *attr*, *initAttr*, *inv*)
=> ....
Closure(..) => ...
Guard(...) => ...



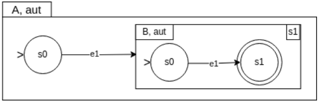
$$aut_2 \quad \frac{s \xrightarrow{\sigma, E_g, E'_g}_{a.\nu(n)} s' \qquad E_g = E_e \Leftarrow E \quad E'_e = E_e \Leftarrow attr \lhd E'_g \quad E' = attr \lhd E'_g}{(aut_\circ, n, E, s) \xrightarrow{\sigma, E_e, E'_e}_a (aut_\circ, n, E', s')}$$

$\{aut_\circ(n, E', s') \mapsto E'_e \mid n, E, s, s', E'_e, E_g, E'_g, E' \cdot$
2. $\wedge n = StateName(current)$
3. $\wedge s = AutSubState(current)$
4. $\wedge E = AutCache(current)$
5. $\wedge s' \mapsto E'_g \in NextState(mapping(n), \sigma, s, E_g)$
6. $\wedge E_g = E_e \Leftarrow E$
7. $\wedge E'_e = E_e \Leftarrow attr \lhd E'_g$
8. $\wedge E' = attr \lhd E'_g$
$\}$

ASTDBehavior - Transition rules

Inductive sequent-based operational semantics rules. (For Eg. aut_2)

NextState ***expression*** (
  $astd$: $ASTD(St, Ev, Var)$,
  $current$: $ASTDState(St, Var)$,
  E_e : $Var \nrightarrow \mathbb{Z}$)
well-defindness condition
  $ASTD\_WellCons(astd, dom(E_e))$
recursive definition
case astd :
  Elementary(inv) => ...
  Automaton($i, f, ..., mapping$) =>
    $aut_1 \cup$ **aut₂** $\cup\, aut_3$
  Sequence($fst, snd, attr, initAttr, inv$)
    => ....
  Closure(..) => ...
  Guard(...) => ...



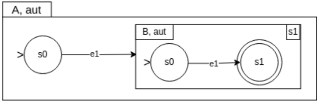
$$aut_2\ \frac{s \xrightarrow{\sigma, E_g, E'_g}_{a.\nu(n)} s' \qquad E_g = E_e \mathbin{\lhd\!\!\!+} E \qquad E'_e = E_e \mathbin{\lhd\!\!\!+} attr \mathbin{\lhd\!\!\!-} E'_g \qquad E' = attr \lhd E'_g}{(aut_\circ, n, E, s) \xrightarrow{\sigma, E_e, E'_e}_{a} (aut_\circ, n, E', s')}$$

$\{aut_\circ(n, E', s') \mapsto E'_e \mid n, E, s, s', E'_e, E_g, E'_g, E' \cdot$
2. $\wedge n = StateName(current)$
3. $\wedge s = AutSubState(current)$
4. $\wedge E = AutCache(current)$
5. $\wedge s' \mapsto E'_g \in NextState(mapping(n), \sigma, s, E_g)$
6. $\wedge E_g = E_e \mathbin{\lhd\!\!\!+} E$
7. $\wedge E'_e = E_e \mathbin{\lhd\!\!\!+} attr \mathbin{\lhd\!\!\!-} E'_g$
8. $\wedge E' = attr \lhd E'_g$
$\}$

ASTDBehavior - Transition rules

Inductive sequent-based operational semantics rules. (For Eg. aut_2)

NextState ***expression*** (
$astd$: $ASTD(St, Ev, Var)$,
$current$: $ASTDState(St, Var)$,
E_e : $Var \rightarrow\!\!\!\!\!\!+\;\, \mathbb{Z}$)
well−definedness condition
$ASTD\_WellCons(astd, dom(E_e))$
recursive definition
case astd :
Elementary(inv) => ...
Automaton$(i, f, ..., mapping)$ =>
$aut_1 \cup \mathbf{aut_2} \cup aut_3$
Sequence$(fst, snd, attr, initAttr, inv)$
=> ....
Closure(..) => ...
Guard(...) => ...



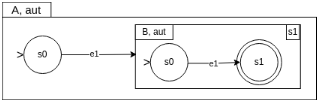
$$aut_2 \; \frac{s \xrightarrow{\sigma, E_g, E'_g}_{a.\nu(n)} s' \qquad E_g = E_e \mathbin{<\!\!+} E \quad E'_e = E_e \mathbin{<\!\!+} attr \lhd E'_g \quad E' = attr \lhd E'_g}{(aut_\circ, n, E, s) \xrightarrow{\sigma, E_e, E'_e}_a (aut_\circ, n, E', s')}$$

$\{aut_\circ(n, E', s') \mapsto E'_e \mid n, E, s, s', E'_e, E_g, E'_g, E' \cdot$
2. $\wedge n = StateName(current)$
3. $\wedge s = AutSubState(current)$
4. $\wedge E = AutCache(current)$
5. $\wedge s' \mapsto E'_g \in NextState(mapping(n), \sigma, s, E_g)$
6. $\wedge E_g = E_e \mathbin{<\!\!+} E$
7. $\wedge E'_e = E_e \mathbin{<\!\!+} attr \lhd E'_g$
8. $\wedge E' = attr \lhd E'_g$
$\}$

ASTDBehavior - Transition rules

Inductive sequent-based operational semantics rules. (For Eg. aut_2)

```
NextState expression (
      astd : ASTD(St, Ev, Var) ,
      current : ASTDState(St, Var) ,
      E_e : Var ⇸ ℤ)
well–definedness condition
      ASTD_WellCons(astd, dom(E_e))
recursive definition
case astd:
   Elementary(inv) => ...
   Automaton(i, f, ..., mapping) =>
      aut_1 ∪ aut_2 ∪ aut_3
   Sequence(fst, snd, attr, initAttr, inv)
           => ....
   Closure(..) => ...
   Guard(...) => ...
```



$$aut_2 \quad \frac{s \xrightarrow{\sigma, E_g, E'_g}_{a.\nu(n)} s' \qquad \begin{array}{l} E_g = E_e \Lsh E \\ E'_e = E_e \Lsh attr \lhd E'_g \\ E' = attr \lhd E'_g \end{array}}{(aut_\circ, n, E, s) \xrightarrow{\sigma, E_e, E'_e}_{a} (aut_\circ, n, E', s')}$$

$\{aut_\circ(n, E', s') \mapsto E'_e \mid n, E, s, s', E'_e, E_g, E'_g, E' \cdot$

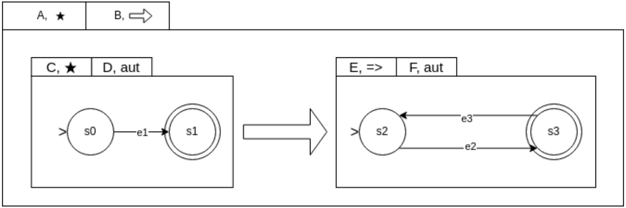
2. $\wedge n = StateName(current)$
3. $\wedge s = AutSubState(current)$
4. $\wedge E = AutCache(current)$
5. $\wedge s' \mapsto E'_g \in NextState(mapping(n), \sigma, s, E_g)$
6. $\wedge E_g = E_e \Lsh E$
7. $\wedge E'_e = E_e \Lsh attr \lhd E'_g$
8. $\wedge E' = attr \lhd E'_g$

$\}$

Outline

Modelling specific ASTD as an instance

Back to the example



State variable, actions, invariant are defined :

ASTD	Attributes	Initialisation	Invariant
A	x_A	$x_A := 0$	$x_A \geq 0$
B	x_B	$x_B := x_A + 1$	$x_B > 0$
...			
S3			$x_F > 0 \wedge x_A > 2 * x_E$

Deep Instantiation

CONTEXT IllustrativeExample
CONSTANTS
e1, e2, e3,
s0, s1, s2, s3,
astd, B, C, D, E, F,
xA, xB, xC, xD, xE, xF
AXIOMS
axm1: $partition(Ev, \{e1\}, \{e2\}, \{e3\})$
axm2: $partition(St, \{s0\}, \{s1\}, \{s2\}, \{s3\})$
axm3: $partition(Var, \{xA\}, \ldots, \{xF\})$
axm4: $root \in ASTD(St, Ev, Var)$
axm12: $F = Automaton(s2, \{s3\}, \ldots)$
axm13: $E = Guard(\{\ldots\}, F, \{xE\}, \ldots\})$
axm14: $D = Automaton(\ldots)$
axm15: $C = Closure(D, \ldots)$
axm16: $B = Sequence(C, E, \ldots)$
axm17: $root = Closure(B, \ldots)$
THEOREMS
thm_of_WD: $ASTD\_WellCons(root, \emptyset)$
END

Deep Instantiation

CONTEXT IllustrativeExample
CONSTANTS
e1 , e2 , e3 ,
s0 , s1 , s2 , s3 ,
astd , B, C, D, E, F,
xA , xB , xC , xD , xE , xF
AXIOMS
axm1 : $partition(Ev, \{e1\}, \{e2\}, \{e3\})$
axm2 : $partition(St, \{s0\}, \{s1\}, \{s2\}, \{s3\})$
axm3 : $partition(Var, \{xA\}, \ldots, \{xF\})$
axm4 : $root \in ASTD(St, Ev, Var)$
axm12 : $F = Automaton(s2, \{s3\}, \ldots)$
axm13 : $E = Guard(\{\ldots\}, F, \{xE\}, \ldots\})$
axm14 : $D = Automaton(\ldots)$
axm15 : $C = Closure(D, \ldots)$
axm16 : $B = Sequence(C, E, \ldots)$
axm17 : $root = Closure(B, \ldots)$
THEOREMS
thm_of_WD : $ASTD\_WellCons(root, \emptyset)$
END

Deep Instantiation

```
CONTEXT IllustrativeExample
CONSTANTS
  e1, e2, e3,
  s0, s1, s2, s3,
  astd, B, C, D, E, F,
  xA, xB, xC, xD, xE, xF
AXIOMS
```

axm1: $partition(Ev, \{e1\}, \{e2\}, \{e3\})$
axm2: $partition(St, \{s0\}, \{s1\}, \{s2\}, \{s3\})$
axm3: $partition(Var, \{xA\}, \ldots, \{xF\})$
axm4: $root \in ASTD(St, Ev, Var)$
axm12: $F = Automaton(s2, \{s3\}, \ldots)$
axm13: $E = Guard(\{\ldots\}, F, \{xE\}, \ldots\})$
axm14: $D = Automaton(\ldots)$
axm15: $C = Closure(D, \ldots)$
axm16: $B = Sequence(C, E, \ldots)$
axm17: $root = Closure(B, \ldots)$
THEOREMS
thm_of_WD: $ASTD\_WellCons(root, \emptyset)$
END

- Applying datatype constructors as compositional operator

Deep Instantiation

```
CONTEXT IllustrativeExample
CONSTANTS
  e1, e2, e3,
  s0, s1, s2, s3,
  astd, B, C, D, E, F,
  xA, xB, xC, xD, xE, xF
AXIOMS
axm1: partition(Ev, {e1}, {e2}, {e3})
axm2: partition(St, {s0}, {s1}, {s2}, {s3})
axm3: partition(Var, {xA}, ..., {xF})
axm4: root ∈ ASTD(St, Ev, Var)
axm12: F = Automaton(s2, {s3}, ...)
axm13: E = Guard({...}, F, {xE}, ...})
axm14: D = Automaton(...)
axm15: C = Closure(D, ...)
axm16: B = Sequence(C, E, ...)
axm17: root = Closure(B, ...)
THEOREMS
thm_of_WD: ASTD_WellCons(root, ∅)
END
```

- Applying datatype constructors as compositional operator

Deep Instantiation

CONTEXT IllustrativeExample
CONSTANTS
e1 , e2 , e3 ,
s0 , s1 , s2 , s3 ,
astd , B, C, D, E, F,
xA , xB , xC , xD , xE , xF
AXIOMS
axm1 : $partition(Ev, \{e1\}, \{e2\}, \{e3\})$
axm2 : $partition(St, \{s0\}, \{s1\}, \{s2\}, \{s3\})$
axm3 : $partition(Var, \{xA\}, \ldots, \{xF\})$
axm4 : $root \in ASTD(St, Ev, Var)$
axm12 : $F = Automaton(s2, \{s3\}, \ldots)$
axm13 : $E = Guard(\{\ldots\}, F, \{xE\}, \ldots\})$
axm14 : $D = Automaton(\ldots)$
axm15 : $C = Closure(D, \ldots)$
axm16 : $B = Sequence(C, E, \ldots)$
axm17 : $root = Closure(B, \ldots)$
THEOREMS
thm_of_WD : $ASTD\_WellCons(root, \emptyset)$
END

- Applying datatype constructors as compositional operator

Deep Instantiation

```
CONTEXT IllustrativeExample
CONSTANTS
  e1, e2, e3,
  s0, s1, s2, s3,
  astd, B, C, D, E, F,
  xA, xB, xC, xD, xE, xF
AXIOMS
axm1: partition(Ev, {e1}, {e2}, {e3})
axm2: partition(St, {s0}, {s1}, {s2}, {s3})
axm3: partition(Var, {xA}, ..., {xF})
axm4: root ∈ ASTD(St, Ev, Var)
axm12: F = Automaton(s2, {s3}, ...)
axm13: E = Guard({...}, F, {xE}, ...})
axm14: D = Automaton(...)
axm15: C = Closure(D, ...)
axm16: B = Sequence(C, E, ...)
axm17: root = Closure(B, ...)
THEOREMS
thm_of_WD: ASTD_WellCons(root, ∅)
END
```

- Applying datatype constructors as compositional operator
- ASTD consistency is ensured by discharging the *THM_ctx* PO generated for the *thm_of_WD* theorem.

$$axm1 \wedge .. \wedge axm17 \implies thm\_of\_WD$$

Deep Instantiation

CONTEXT IllustrativeExample
CONSTANTS
e1, e2, e3,
s0, s1, s2, s3,
astd, B, C, D, E, F,
xA, xB, xC, xD, xE, xF
AXIOMS
axm1: $partition(Ev, \{e1\}, \{e2\}, \{e3\})$
axm2: $partition(St, \{s0\}, \{s1\}, \{s2\}, \{s3\})$
axm3: $partition(Var, \{xA\}, \ldots, \{xF\})$
axm4: $root \in ASTD(St, Ev, Var)$
axm12: $F = Automaton(s2, \{s3\}, \ldots)$
axm13: $E = Guard(\{\ldots\}, F, \{xE\}, \ldots\})$
axm14: $D = Automaton(\ldots)$
axm15: $C = Closure(D, \ldots)$
axm16: $B = Sequence(C, E, \ldots)$
axm17: $root = Closure(B, \ldots)$
THEOREMS
thm_of_WD: $ASTD\_WellCons(root, \emptyset)$
END

- Applying datatype constructors as compositional operator
- ASTD consistency is ensured by discharging the *THM_ctx* PO generated for the *thm_of_WD* theorem.

$$axm1 \wedge .. \wedge axm17 \implies thm\_of\_WD$$

ASTD is instantiated as First class citizen

Shallow instantiation

A generic machine that interpret ASTD.

1. The *current* state is initialized by Init
2. The *progress* event

```
MACHINE ShallowGenAnim
SEES ASTD_Ctx
VARIABLES current
INVARIANTS
  inv1 : current ∈ ASTDState(St, Var)
EVENTS
  INITIALISATION
  THEN
    act1 : current := Init(root, ∅)
  END
  progress
  ANY evnt , nxt
  WHERE
    grd1 : evnt ∈ Ev
    grd2 : nxt ∈ NextState(root, evnt, current, ∅)
    grd3 : NextState(root, evnt, current, ∅) ≠ ∅
  THEN
    act1 : current := prj1(nxt)
  END
END
```

Listing 1 – Shallow Instantiation

Shallow Instantiation : Animation

Soundness principle

A proof obligation is a maths formula to be proven associated to the correctness of a given property.

How do I know the proof obligation actually entails the property ?

Soundness principle

A proof obligation is a maths formula to be proven associated to the correctness of a given property.

How do I know the proof obligation actually entails the property ?

Pattern

Encoding the POs in form of properties on traces.

$ASTD \vdash PO \implies Spec\_\mathbf{[}PO\mathbf{]}\_On\_Traces(ASTD)$

Soundness principle

A proof obligation is a maths formula to be proven associated to the correctness of a given property.

How do I know the proof obligation actually entails the property ?

Pattern

Encoding the POs in form of properties on traces.

$ASTD \vdash PO \implies Spec\_\mathbf{[}PO\mathbf{]}\_On\_Traces(ASTD)$

Invariants case

$ASTD \vdash PO_{INV} \implies Spec\_PO_{INV}\_On\_Traces(ASTD)$

Soundness principle

A proof obligation is a maths formula to be proven associated to the correctness of a given property.

How do I know the proof obligation actually entails the property ?

Pattern

Encoding the POs in form of properties on traces.

$ASTD \vdash PO \implies Spec\_[\![PO]\!]\_On\_Traces(ASTD)$

Invariants case

$ASTD \vdash PO_{INV} \implies Spec\_PO_{INV}\_On\_Traces(ASTD)$

$$\boxed{ASTD \vdash PO_{INV} \implies \underbrace{\forall tr \in Traces(ASTD) : \forall i \geq 0 : INV(tr(i)))}_{Spec\_PO_{INV}\_On\_Traces(ASTD)}}$$

Outline

ASTDPO - Proof Obligations from pASTD

$$ASTD \vdash \boxed{\textbf{PO}} \implies \forall tr \in Traces(ASTD) : \forall i \geq 0 : INV(tr(i)))$$

Two POs are defined :

- PO_{init} : The invariants hold for the initialization
- PO_{trans} : Transitions preserves invariants

Automatic PO generation

```
THEORY ASTDPO
IMPORT THEORY ASTDBehaviour
TYPE PARAMETERS St, Ev, Var
OPERATORS
   POinit predicate (a : ASTD(St, Ev, Var), ...)
     well-definedness condition ....
     recursive definition ...

   POtrans predicate (a : ASTD(St, Ev, Var), ...)
     well-definedness condition ....
     recursive definition ...
```

```
CONTEXT   ASTD_Ctx_PO
EXTENDS   ASTD_Ctx
THEOREMS
   Generation_of_POi : POinit(root, ...)
   Generation_of_POtr : POtrans(root, ...)
END
```

ASTDPO - Proof Obligations from pASTD

$$ASTD \vdash \boxed{PO} \implies \forall tr \in Traces(ASTD) : \forall i \geq 0 : INV(tr(i)))$$

Two POs are defined :

- PO_{init} : The invariants hold for the initialization
- PO_{trans} : Transitions preserves invariants

```
THEORY ASTDPO
IMPORT THEORY ASTDBehaviour
TYPE PARAMETERS St, Ev, Var
OPERATORS
   POinit predicate (a : ASTD(St, Ev, Var), ...)
     well-definedness condition ....
     recursive definition ...

   POtrans predicate (a : ASTD(St, Ev, Var), ...)
     well-definedness condition ....
     recursive definition ...
```

```
CONTEXT   ASTD_Ctx_PO
EXTENDS   ASTD_Ctx
THEOREMS
   Generation_of_POi : POinit(root, ...)
   Generation_of_POtr : POtrans(root, ...)
END
```

Automatic PO generation

ASTDPO - Proof Obligations from pASTD

$$ASTD \vdash \boxed{PO} \implies \forall tr \in Traces(ASTD) : \forall i \geq 0 : INV(tr(i)))$$

Two POs are defined :

- PO_{init} : The invariants hold for the initialization
- PO_{trans} : Transitions preserves invariants

THEORY ASTDPO
IMPORT THEORY ASTDBehaviour
TYPE PARAMETERS St, Ev, Var
OPERATORS
  POinit ***predicate*** $(a : ASTD(St, Ev, Var), \dots)$
    well–definedness condition ....
    recursive definition ...

  POtrans ***predicate*** $(a : ASTD(St, Ev, Var), \dots)$
    well–definedness condition ....
    recursive definition ...

CONTEXT ASTD_Ctx_PO
EXTENDS ASTD_Ctx
THEOREMS
  Generation_of_POi : **POinit**$(root, \dots)$
  Generation_of_POtr : **POtrans**$(root, \dots)$
END

Automatic PO generation

ASTDPO - Proof Obligations from pASTD

$$ASTD \vdash \boxed{PO} \implies \forall tr \in Traces(ASTD) : \forall i \geq 0 : INV(tr(i)))$$

Two POs are defined :

- PO_{init} : The invariants hold for the initialization
- PO_{trans} : Transitions preserves invariants

Automatic PO generation

```
THEORY ASTDPO
IMPORT THEORY ASTDBehaviour
TYPE PARAMETERS St, Ev, Var
OPERATORS
   POinit predicate (a : ASTD(St, Ev, Var), ...)
     well-definedness condition ....
     recursive definition ...

   POtrans predicate (a : ASTD(St, Ev, Var), ...)
     well-definedness condition ....
     recursive definition ...
```

```
CONTEXT   ASTD_Ctx_PO
EXTENDS   ASTD_Ctx
THEOREMS
   Generation_of_POi : POinit(root, ...)
   Generation_of_POtr : POtrans(root, ...)
END
```

Outline

ASTDTraces - Trace based semantics

$$ASTD \vdash \textbf{PO} \implies \forall tr \in \boxed{\textbf{Traces(ASTD)}} : \forall i \geq 0 : INV(tr(i))$$

A sequence of $s_0 \mapsto \cdots \mapsto s_n$ states

1. s_0 is the initial state, i.e. $s_0 = Init(A, \emptyset)$
2. consecutive states link by triggered event $\forall i, \exists e \in Ev, s_i \xrightarrow{e} s_{i+1}$
3. trace is infinite or finite and deadlock, i.e. total/partial function

```
THEORY ASTDTraces
IMPORT THEORY ASTDPO
TYPE PARAMETERS St , Ev , Var
OPERATORS
IsANextState predicate
        (a : ASTD(St, Ev, Var), s, sp : ASTDState(St, Var))
well-definedness condition ASTD_WellCons(a, ∅)
direct definition ...

IsATrace predicate (a : ASTD(St, Ev, Var) ,
        tr : ℕ ⇸ ASTDState(St, Var))
well-definedness condition ASTD_WellCons(a, ∅)
direct definition
1−     tr(0) = Init(a, ∅)
2− ∧ (∀i, j · i ∈ dom(tr) ∧ j ∈ dom(tr) ∧ j = i + 1 ⇒
        IsANextState(a, tr(i), tr(j)))
3− ∧ (tr ∈ ℕ → ASTDState(St, Var) ∨ // infinite trace
        (∃n · n ∈ ℕ ∧ tr ∈ 0..n → ASTDState(St, Var)
           ∧ (¬CanProgress(a, tr(n))))) // finite trace
```

ASTDTraces - Trace based semantics

$$ASTD \vdash \textbf{PO} \implies \forall tr \in \boxed{\textbf{Traces(ASTD)}} : \forall i \geq 0 : INV(tr(i))$$

A sequence of $s_0 \mapsto \cdots \mapsto s_n$ states

```
THEORY ASTDTraces
IMPORT THEORY ASTDPO
TYPE PARAMETERS St , Ev , Var
OPERATORS
IsANextState predicate
        (a : ASTD(St, Ev, Var), s, sp : ASTDState(St, Var))
well−definedness condition ASTD_WellCons(a, ∅)
direct definition ...

IsATrace predicate (a : ASTD(St, Ev, Var) ,
        tr : ℕ ⇸ ASTDState(St, Var))
well−definedness condition ASTD_WellCons(a, ∅)
direct definition
1−      tr(0) = Init(a, ∅)
2− ∧ (∀i, j · i ∈ dom(tr) ∧ j ∈ dom(tr) ∧ j = i + 1 ⇒
        IsANextState(a, tr(i), tr(j)))
3− ∧ (tr ∈ ℕ → ASTDState(St, Var) ∨ // infinite trace
        (∃n · n ∈ ℕ ∧ tr ∈ 0..n → ASTDState(St, Var)
          ∧ (¬CanProgress(a, tr(n))))) // finite trace
```

ASTDTraces - Trace based semantics

$$ASTD \vdash \mathbf{PO} \implies \forall tr \in \boxed{\textbf{Traces(ASTD)}} : \forall i \geq 0 : INV(tr(i))$$

A sequence of $s_0 \mapsto \cdots \mapsto s_n$ states

1. s_0 is the initial state, i.e. $s_0 = Init(A, \emptyset)$

```
THEORY ASTDTraces
IMPORT THEORY ASTDPO
TYPE PARAMETERS St , Ev , Var
OPERATORS
IsANextState predicate
        (a : ASTD(St, Ev, Var), s, sp : ASTDState(St, Var))
well−definedness condition ASTD_WellCons(a, ∅)
direct definition . . .

IsATrace predicate (a : ASTD(St, Ev, Var) ,
        tr : ℕ ⇸ ASTDState(St, Var))
well−definedness condition ASTD_WellCons(a, ∅)
direct definition
1−      tr(0) = Init(a, ∅)
2− ∧ (∀i, j · i ∈ dom(tr) ∧ j ∈ dom(tr) ∧ j = i + 1 ⇒
        IsANextState(a, tr(i), tr(j)))
3− ∧ (tr ∈ ℕ → ASTDState(St, Var) ∨ // infinite trace
        (∃n · n ∈ ℕ ∧ tr ∈ 0..n → ASTDState(St, Var)
           ∧ (¬CanProgress(a, tr(n))))) // finite trace
```

ASTDTraces - Trace based semantics

$$ASTD \vdash \textbf{PO} \implies \forall tr \in \boxed{\textbf{Traces(ASTD)}} : \forall i \geq 0 : INV(tr(i))$$

A sequence of $s_0 \mapsto \cdots \mapsto s_n$ states

1. s_0 is the initial state, i.e. $s_0 = Init(A, \emptyset)$
2. consecutive states link by triggered event $\forall i, \exists e \in Ev, s_i \xrightarrow{e} s_{i+1}$

```
THEORY ASTDTraces
IMPORT THEORY ASTDPO
TYPE PARAMETERS St , Ev , Var
OPERATORS
IsANextState predicate
        (a : ASTD(St, Ev, Var), s, sp : ASTDState(St, Var))
 well-definedness condition ASTD_WellCons(a, ∅)
 direct definition ...

IsATrace predicate (a : ASTD(St, Ev, Var) ,
        tr : ℕ ⇸ ASTDState(St, Var))
 well-definedness condition ASTD_WellCons(a, ∅)
 direct definition
1-     tr(0) = Init(a, ∅)
2-  ∧ (∀i, j · i ∈ dom(tr) ∧ j ∈ dom(tr) ∧ j = i + 1 ⇒
        IsANextState(a, tr(i), tr(j)))
3-  ∧ (tr ∈ ℕ → ASTDState(St, Var) ∨ // infinite trace
        (∃n · n ∈ ℕ ∧ tr ∈ 0..n → ASTDState(St, Var)
          ∧ (¬CanProgress(a, tr(n))))) // finite trace
```

ASTDTraces - Trace based semantics

$$ASTD \vdash \mathbf{PO} \implies \forall tr \in \boxed{\textbf{Traces(ASTD)}} : \forall i \geq 0 : INV(tr(i))$$

A sequence of $s_0 \mapsto \cdots \mapsto s_n$ states

1. s_0 is the initial state, i.e. $s_0 = Init(A, \emptyset)$
2. consecutive states link by triggered event $\forall i, \exists e \in Ev, s_i \xrightarrow{e} s_{i+1}$
3. trace is infinite or finite and deadlock, i.e. total/partial function

```
THEORY ASTDTraces
IMPORT THEORY ASTDPO
TYPE PARAMETERS St , Ev , Var
OPERATORS
IsANextState predicate
        (a : ASTD(St, Ev, Var), s, sp : ASTDState(St, Var))
well−definedness condition ASTD_WellCons(a, ∅)
direct definition ...

IsATrace predicate (a : ASTD(St, Ev, Var) ,
        tr : ℕ ⇸ ASTDState(St, Var))
well−definedness condition ASTD_WellCons(a, ∅)
direct definition
1−    tr(0) = Init(a, ∅)
2− ∧ (∀i, j · i ∈ dom(tr) ∧ j ∈ dom(tr) ∧ j = i + 1 ⇒
        IsANextState(a, tr(i), tr(j)))
3− ∧ (tr ∈ ℕ → ASTDState(St, Var) ∨ // infinite trace
        (∃n · n ∈ ℕ ∧ tr ∈ 0..n → ASTDState(St, Var)
            ∧ (¬CanProgress(a, tr(n))))) // finite trace
```

Specification on astd state

Definition of "holding invariants" for an ASTD state.

$$ASTD \vdash \mathbf{PO} \implies \forall tr \in \mathbf{Traces(ASTD)} : \forall i \geq 0 : \boxed{\mathbf{INV(tr(i))}}$$

```
THEORY ASTDCorrectness
IMPORT THEORY ASTDTraces
TYPE PARAMETERS St, Ev, Var
OPERATORS
  InvSpecOnState predicate (a : ASTD(St, Ev, Var),
      s : ASTDState(St, Var))
    well-definedness condition
    recursive definition
  Elementary(inv) => ...
  Automaton(i, f, ..., mapping) => ...
  Sequence(fst, snd, attr, initAttr, inv) => ....
  Closure(...) => ...
  Guard(...) => ...
```

Listing 2 – Definition of satisfying invariants

Ultimate proof of soundness

$$ASTD \vdash \textbf{PO} \boxed{\Longrightarrow} \forall tr \in \textbf{Traces(ASTD)} : \forall i \geq 0 : \textbf{INV(tr(i))}$$

thm_of_PO_Correctness :
$\forall astd, tr \cdot$

$astd \in ASTD(St, Ev, Var)$ // for all astd
$\wedge ASTD\_WellCons(astd, \emptyset)$ // well cons

$\wedge tr \in \mathbb{N} \nrightarrow ASTDState(St, Var)$ // and trace
$\wedge IsATrace(astd, tr)$ // of the astd

$\wedge POinit(astd, Var \nrightarrow \mathbb{Z}, Var \nrightarrow \mathbb{Z}, (Var \nrightarrow \mathbb{Z}) \lhd id)$
$\wedge POtrans(astd, Var \nrightarrow \mathbb{Z}, \emptyset)$ // when POs hold

$\Rightarrow$
$(\forall i \cdot i \in dom(tr) \Rightarrow InvSpecOnState(astd, tr(i)))$
// every state in the trace satisfies the invariant

Ultimate proof of soundness

$$ASTD \vdash \mathbf{PO} \boxed{\Longrightarrow} \forall tr \in \mathbf{Traces(ASTD)} : \forall i \geq 0 : \mathbf{INV(tr(i))}$$

thm_of_PO_Correctness :
$\forall astd, tr \cdot$

$astd \in ASTD(St, Ev, Var)$ // for all astd
$\wedge ASTD\_WellCons(astd, \emptyset)$ // well cons

$\wedge tr \in \mathbb{N} \rightarrowtail ASTDState(St, Var)$ // and trace
$\wedge IsATrace(astd, tr)$ // of the astd

$\wedge POinit(astd, Var \rightarrowtail \mathbb{Z}, Var \rightarrowtail \mathbb{Z}, (Var \rightarrowtail \mathbb{Z}) \lhd id)$
$\wedge POtrans(astd, Var \rightarrowtail \mathbb{Z}, \emptyset)$ // when POs hold

$\Rightarrow$
$(\forall i \cdot i \in dom(tr) \Rightarrow InvSpecOnState(astd, tr(i)))$
// every state in the trace satisfies the invariant

Ultimate proof of soundness

$$ASTD \vdash \mathbf{PO} \boxed{\Longrightarrow} \forall tr \in \mathbf{Traces(ASTD)} : \forall i \geq 0 : \mathbf{INV(tr(i))}$$

thm_of_PO_Correctness :
$\forall astd, tr \cdot$

$astd \in ASTD(St, Ev, Var)$ // for all astd
$\wedge ASTD\_WellCons(astd, \emptyset)$ // well cons

$\wedge tr \in \mathbb{N} \nrightarrow ASTDState(St, Var)$ // and trace
$\wedge IsATrace(astd, tr)$ // of the astd

$\wedge POinit(astd, Var \nrightarrow \mathbb{Z}, Var \nrightarrow \mathbb{Z}, (Var \nrightarrow \mathbb{Z}) \lhd id)$
$\wedge POtrans(astd, Var \nrightarrow \mathbb{Z}, \emptyset)$ // when POs hold

$\Rightarrow$
$(\forall i \cdot i \in dom(tr) \Rightarrow InvSpecOnState(astd, tr(i)))$
// every state in the trace satisfies the invariant

Ultimate proof of soundness

$$ASTD \vdash \mathbf{PO} \boxed{\Longrightarrow} \forall tr \in \mathbf{Traces(ASTD)} : \forall i \geq 0 : \mathbf{INV(tr(i))}$$

thm_of_PO_Correctness :
$\forall astd, tr \cdot$

$astd \in ASTD(St, Ev, Var)$ // for all astd
$\wedge ASTD\_WellCons(astd, \emptyset)$ // well cons

$\wedge tr \in \mathbb{N} \nrightarrow ASTDState(St, Var)$ // and trace
$\wedge IsATrace(astd, tr)$ // of the astd

$\wedge POinit(astd, Var \nrightarrow \mathbb{Z}, Var \nrightarrow \mathbb{Z}, (Var \nrightarrow \mathbb{Z}) \lhd id)$
$\wedge POtrans(astd, Var \nrightarrow \mathbb{Z}, \emptyset)$ // when POs hold

$\Rightarrow$
$(\forall i \cdot i \in dom(tr) \Rightarrow InvSpecOnState(astd, tr(i)))$
// every state in the trace satisfies the invariant

Ultimate proof of soundness

$$ASTD \vdash \mathbf{PO} \boxed{\color{red}{\Longrightarrow}} \forall tr \in \mathbf{Traces(ASTD)} : \forall i \geq 0 : \mathbf{INV(tr(i))}$$

thm_of_PO_Correctness :
$\forall astd, tr \cdot$

$astd \in ASTD(St, Ev, Var)$ // for all astd
$\wedge ASTD\_WellCons(astd, \emptyset)$ // well cons

$\wedge tr \in \mathbb{N} \nrightarrow ASTDState(St, Var)$ // and trace
$\wedge IsATrace(astd, tr)$ // of the astd

$\wedge POinit(astd, Var \nrightarrow \mathbb{Z}, Var \nrightarrow \mathbb{Z}, (Var \nrightarrow \mathbb{Z}) \lhd id)$
$\wedge POtrans(astd, Var \nrightarrow \mathbb{Z}, \emptyset)$ // when POs hold

$\Rightarrow$
$(\forall i \cdot i \in dom(tr) \Rightarrow InvSpecOnState(astd, tr(i)))$
// every state in the trace satisfies the invariant

Ultimate proof of soundness

$$ASTD \vdash \textbf{PO} \boxed{\Longrightarrow} \forall tr \in \textbf{Traces(ASTD)} : \forall i \geq 0 : \textbf{INV(tr(i))}$$

thm_of_PO_Correctness :
$\forall astd, tr \cdot$

$astd \in ASTD(St, Ev, Var)$ // for all astd
$\wedge ASTD\_WellCons(astd, \emptyset)$ // well cons

$\wedge tr \in \mathbb{N} \nrightarrow ASTDState(St, Var)$ // and trace
$\wedge IsATrace(astd, tr)$ // of the astd

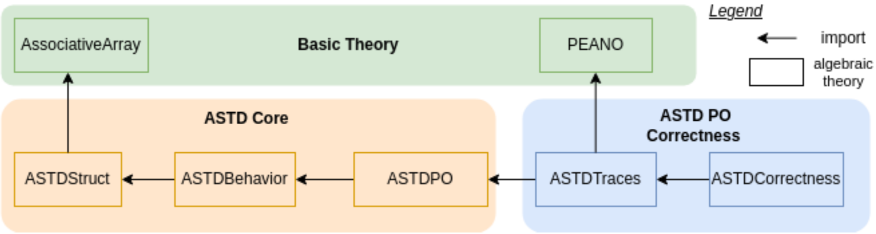
$\wedge POinit(astd, Var \nrightarrow \mathbb{Z}, Var \nrightarrow \mathbb{Z}, (Var \nrightarrow \mathbb{Z}) \lhd id)$
$\wedge POtrans(astd, Var \nrightarrow \mathbb{Z}, \emptyset)$ // when POs hold

$\Rightarrow$
$(\forall i \cdot i \in dom(tr) \Rightarrow InvSpecOnState(astd, tr(i)))$
// every state in the trace satisfies the invariant

Discover ***incomplete*** WD conditions and *minor* **bugs** in the ASTD manual specification (for eg. variable environment).

1. Introduction
   - Motivation
   - Definition
   - Related work
2. An algebraic meta-model for ASTD
   - Background
   - The core meta-theory
   - Two instantiation mechanisms : Deep and Shallow
3. Proof-based reasoning
   - Proof obligations
   - Soundness
4. Conclusion

Overview of EB[ASTD] development



Development effort :

- Definition of 7 **Theories**
- 22 **operators** - 17 **theorems**
- 54 Proof obligations (WD/Thm) for proving theories
- Mostly *interactive* proof but automation is possible

Conclusion

The meta-model provides a solid tool for ASTD by :

- establishing a **proof-based** framework to reason on ASTDs
- offering the possibility of **model-checking** (ProB)
- visualising the **animation** (VisB)
- allowing **automatic** generation of POs
- proving the **soundness** w.r.t. trace semantics

All of this are grounded in a unique tool support Rodin.
The methodology is also suitable for other (state-based) formalism.

Perspectives

- Handling **refinement** in ASTDs
- Temporal, Security properties on ASTDs (ongoing two PhD thesis..)
- Model to Model transformation (EB4EB $\leftrightarrow$ EB[ASTD])

References I

[1] J.-R. ABRIAL. *Modeling in Event-B : System and Software Engineering*. Cambridge University Press, 2010.

[2] Abhishek ANAND et al. “CertiCoq : A verified compiler for Coq”. In : *The third international workshop on Coq for programming languages (CoqPL)*. 2017.

[3] ASTD TEAM. *cASTD and eASTD*. URL : https://github.com/DiegoOliveiraUDES/ASTD-tools.

[4] Diego de AZEVEDO OLIVEIRA et Marc FRAPPIER. “Modelling an Automotive Software System with TASTD”. In : *Rigorous State-Based Methods - 9th International Conference, ABZ 2023, Nancy, France, May 30 - June 2, 2023, Proceedings*. Sous la dir. d’Uwe GLÄSSER et al. T. 14010. Lecture Notes in Computer Science. Springer, 2023, p. 124-141.

References II

[5] Diego de AZEVEDO OLIVEIRA et Marc FRAPPIER. “TASTD : A Real-Time Extension for ASTD”. In : *Rigorous State-Based Methods - 9th International Conference, ABZ 2023, Nancy, France, May 30 - June 2, 2023, Proceedings*. Sous la dir. d’Uwe GLÄSSER et al. T. 14010. Lecture Notes in Computer Science. Springer, 2023, p. 142-159.

[6] Diego de AZEVEDO OLIVEIRA et Marc FRAPPIER. *Technical report 27 - Extending ASTD with real-time*. Rapp. tech. Université de Sherbrooke, 2024. URL : https://marcfrappier.espaceweb.usherbrooke.ca/Papers/report-27.pdf.

References III

[7] Michael J. Butler et Issam Maamria. “Practical Theory Extension in Event-B”. In : *Theories of Programming and Formal Methods - Essays Dedicated to Jifeng He on the Occasion of His 70th Birthday*. Sous la dir. de Zhiming Liu, Jim Woodcock et Huibiao Zhu. T. 8051. Lecture Notes in Computer Science. Springer, 2013, p. 67-81.

[8] Quelen Cartellier, Marc Frappier et Amel Mammar. “Proving Local Invariants in ASTDs”. In : *Formal Methods and Software Engineering - 24th International Conference on Formal Engineering Methods, ICFEM 2023, Brisbane, QLD, Australia, November 21-24, 2023, Proceedings*. Sous la dir. d’Yi Li et Sofiène Tahar. T. 14308. Lecture Notes in Computer Science. Springer, 2023, p. 228-246.

References IV

[9] Benja FALLENSTEIN et Ramana KUMAR. "Proof-Producing Reflection for HOL - With an Application to Model Polymorphism". In : *Interactive Theorem Proving - 6th International Conference, ITP*. Sous la dir. de Christian URBAN et Xingyuan ZHANG. T. 9236. LNCS. Springer, 2015, p. 170-186.

[10] C. A. R. HOARE. "Communicating Sequential Processes". In : *Commun. ACM* 21.8 (1978), p. 666-677.

[11] Chaymae El JABRI et al. "Development of Monitoring Systems for Anomaly Detection Using ASTD Specifications". In : *Theoretical Aspects of Software Engineering - 16th International Symposium, TASE 2022, Cluj-Napoca, Romania, July 8-10, 2022, Proceedings*. Sous la dir. d'Yamine AÏT AMEUR et Florin CRACIUN. T. 13299. Lecture Notes in Computer Science. Springer, 2022, p. 274-289.

References V

[12] Jérémy MILHAU. "Un processus formel d'intégration de politiques de contrôle d'accès dans les systèmes d'information". Theses. Université Paris-Est ; Université de Sherbrooke (Québec, Canada), déc. 2011. URL : https://theses.hal.science/tel-00674865.

[13] César MUÑOZ et John RUSHBY. "Structural embeddings : Mechanization with method". In : *International Symposium on Formal Methods*. Springer. 1999, p. 452-471.

[14] Alex Rodrigue NDOUNA et Marc FRAPPIER. "Modelling a Mechanical Lung Ventilation System Using TASTD". In : *Rigorous State-Based Methods - 10th International Conference, ABZ 2024, Bergamo, Italy, June 25-28, 2024, Proceedings*. Sous la dir. de Silvia BONFANTI et al. T. 14759. Lecture Notes in Computer Science. Springer, 2024, p. 324-340.

References VI

[15] Lionel NGANYEWOU TIDJON. "Formal modeling of intrusion detection systems". Theses. Institut Polytechnique de Paris ; Université de Sherbrooke (Québec, Canada), nov. 2020. URL : https://theses.hal.science/tel-03137661.

[16] Peter RIVIERE et al. *Formalising Liveness Properties in Event-B with the Reflexive EB4EB Framework*. 2023.

[17] Peter RIVIÈRE, Neeraj Kumar SINGH et Yamine AÏT AMEUR. "Reflexive Event-B : Semantics and Correctness the EB4EB Framework". In : *IEEE Trans. Reliab.* 73.2 (2024), p. 835-850.

References VII

[18] Steve A. SCHNEIDER, Helen TREHARNE et Heike WEHRHEIM. “A CSP Approach to Control in Event-B”. In : *Integrated Formal Methods - 8th International Conference, IFM 2010, Nancy, France, October 11-14, 2010. Proceedings*. Sous la dir. de Dominique MÉRY et Stephan MERZ. T. 6396. Lecture Notes in Computer Science. Springer, 2010, p. 260-274.

[19] M. SOZEAU et al. “The MetaCoq Project”. In : *J. Autom. Reason.* 64.5 (2020), p. 947-999.

Appendices

Des figures, listings, documents complets qui peuvent être appuyer pour répondre au question