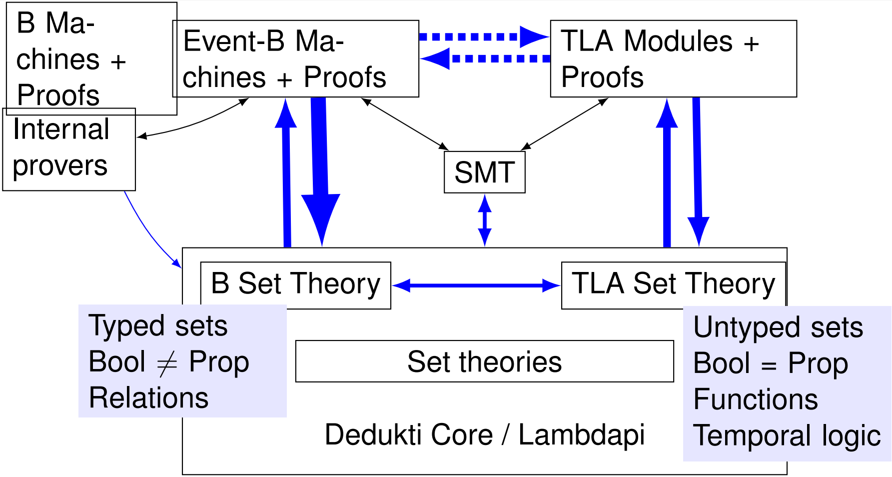
On Proof Support in B/Event-B and TLA

J.P. Bodeveix, M. Filali, A. Grieu
IRIT Université de Toulouse France

TLA+ Community Meeting
September 10, 2024
Co-located with FM 2024 in Milano
Italy

ICSPA project Formal methods based on set theories



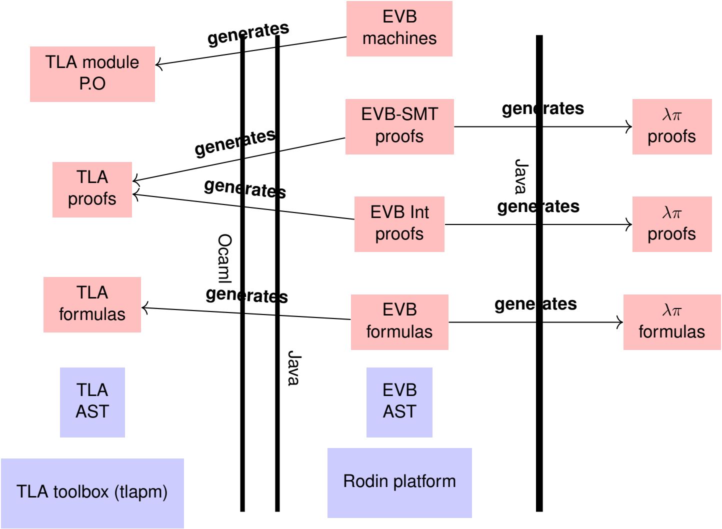

EVB machines
generates
TLA module P.O
EVB-SMT proofs
generates
$\lambda\pi$ proofs
generates
TLA proofs
generates
EVB Int proofs
generates
$\lambda\pi$ proofs
Java
Ocaml
TLA formulas
generates
EVB formulas
generates
$\lambda\pi$ formulas
Java
TLA AST
EVB AST
TLA toolbox (tlapm)
Rodin platform

Table of Contents

Plan

Model expression (1)

```
----------- MODULE SimpleAllocator ---------
(***************************************************************************)
(* Specification of an allocator managing a set of resources:             *)
(* - Clients can request sets of resources whenever all their previous    *)
(*   requests have been satisfied.                                          *)
(* - Requests can be partly fulfilled , and resources can be returned     *)
(*   even before the full request has been satisfied. However, clients    *)
(*   only have an obligation to return resources after they have          *)
(*   obtained all resources they requested.                                *)
(* S. Merz                                                                  *)
(***************************************************************************)
EXTENDS FiniteSets, TLC
CONSTANTS
  Clients,      \* set of all clients
  Resources     \* set of all resources
ASSUME IsFiniteSet(Resources)
VARIABLES
```

Model expression (2)

```
(* Resources are available iff they have ¬been allocated. *)
available ≜ Resources \ (UNION {alloc[c] : c ∈ Clients})
(* Initially , no resources have been requested or allocated. *)
Init ≜
   ∧ unsat = [c ∈ Clients ↦ ∅]
   ∧ alloc = [c ∈ Clients ↦ ∅]
(* A client c may request a set of resources provided that all of its *)
(* previous requests have been satisfied and that it doesn't hold any *)
(* resources.                                                         *)
Request(c,S) ≜
   ∧ unsat[c] = ∅ ∧ alloc[c] = ∅
   ∧ S # ∅ ∧ unsat' = [unsat EXCEPT ![c] = S]
   ∧ UNCHANGED alloc
(* Allocation of a set of available resources to a client that        *)
(* requested them (the entire request does ¬have to be filled).       *)
Allocate(c, S) ≜
   ∧ S # ∅ ∧ S ⊆ available ∩ unsat[c]
```

Model expression (4)

```
(* The next–state relation. *)
Next ≜
  ∃ c ∈ Clients, S ∈ SUBSET Resources :
      Request(c,S) ∨ Allocate(c,S) ∨ Return(c,S)
––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––
(* The complete high–level specification. *)
SimpleAllocator ≜
  ∧ Init  ∧ [□][Next]_vars
  ∧ ∀ c ∈ Clients: WF_vars(Return(c, alloc[c]))
  ∧ ∀ c ∈ Clients: SF_vars(∃ S ∈ SUBSET Resources: Allocate(c,S))
```

Properties expression

```
(* Safety property *)
ResourceMutex ≜
  ∀ c1,c2 ∈ Clients : c1 # c2 ⇒ alloc[c1] ∩ alloc[c2] = ∅
(* Liveness property *)
ClientsWillReturn ≜
  ∀ c ∈ Clients : unsat[c]=∅ ⇝ alloc[c]=∅
(* Fairness properties *)
ClientsWillObtain ≜
  ∀ c ∈ Clients, r ∈ Resources : r ∈ unsat[c] ⇝ r ∈ alloc[c]
InfOftenSatisfied ≜
  ∀ c ∈ Clients : [□]<>(unsat[c] = ∅)
```

Obligations expression

THEOREM SimpleAllocator $\Rightarrow$ [$\Box$]ResourceMutex
THEOREM SimpleAllocator $\Rightarrow$ ClientsWillReturn
THEOREM SimpleAllocator2 $\Rightarrow$ ClientsWillReturn
THEOREM SimpleAllocator $\Rightarrow$ ClientsWillObtain
THEOREM SimpleAllocator $\Rightarrow$ InfOftenSatisfied
(** The following do $\neg$hold: **)

Properties proof

- TLC model checker for model-checking (finite instances).
- TLAPS proofsystem (parameterized instances).

Discuss about the assistance for a TLA proof based development.

Specification in B/Event-B

context cSimpleAllocator
sets Clients Resources
axioms
  @f_Resources finite(Resources)
end

Specification in B/Event-B

machine mSimpleAllocator
sees cSimpleAllocator
variables unsat alloc
invariants
@unsat_ty unsat $\in$ Clients $\rightarrow \mathbb{P}$(Resources)
@alloc_ty alloc $\in$ Clients $\rightarrow \mathbb{P}$(Resources)
@ResourceMutex
$\forall$ c1, c2$\cdot$ (c1 $\in$ Clients $\wedge$ c2 $\in$ Clients $\wedge$ c1 $\neq$ c2)
$\Rightarrow$ ((alloc(c1) $\cap$ alloc(c2)) = $\emptyset$)
events
event INITIALISATION **then**
@unsat_init unsat := Clients $\times$ $\{\emptyset\}$
@alloc_init alloc := Clients $\times$ $\{\emptyset\}$
end

Specification in B/Event-B

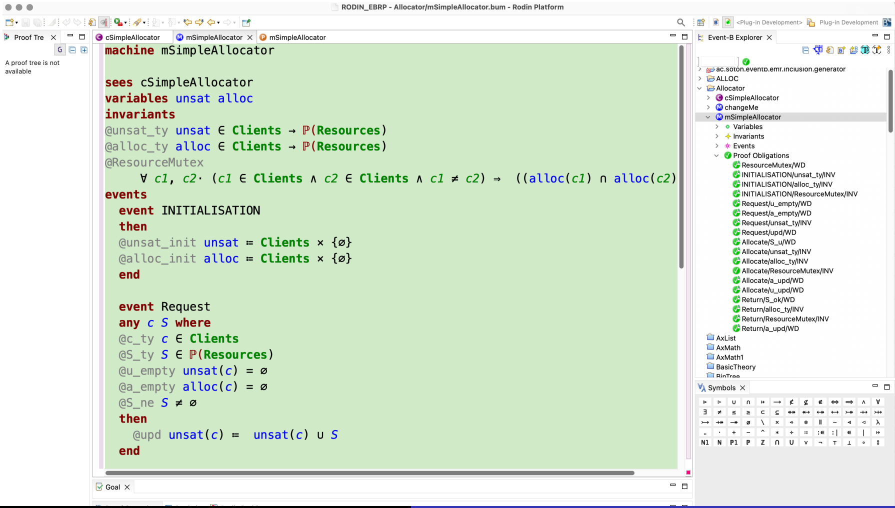
event Request
any c S **where**
@c_ty c $\in$ Clients
@S_ty S $\in \mathbb{P}$(Resources)
@u_empty unsat(c) = $\emptyset$
@a_empty alloc(c) = $\emptyset$
@S_ne S $\neq \emptyset$
then
  @upd unsat(c) $:=$ unsat(c) $\cup$ S
end

Development in B/Event-B

- Predefined properties.
- Automatic generation of proof obligations.
- Automatic proof and Interactive proof development.
- Support for model checking (ProB).

Development in B/Event-B (II)

- Well definedness (wrt. B type theory).
- Invariance.
- Well foundedness.
- Refinement.


RODIN_EBRP - Allocator/mSimpleAllocator.bum - Rodin Platform
Proof Tre
A proof tree is not available
cSimpleAllocator
mSimpleAllocator
mSimpleAllocator
machine mSimpleAllocator
sees cSimpleAllocator
variables unsat alloc
invariants
@unsat_ty unsat ∈ Clients → ℙ(Resources)
@alloc_ty alloc ∈ Clients → ℙ(Resources)
@ResourceMutex
∀ c1, c2· (c1 ∈ Clients ∧ c2 ∈ Clients ∧ c1 ≠ c2) ⇒ ((alloc(c1) ∩ alloc(c2)
events
event INITIALISATION
then
@unsat_init unsat ≔ Clients × {∅}
@alloc_init alloc ≔ Clients × {∅}
end
event Request
any c S where
@c_ty c ∈ Clients
@S_ty S ∈ ℙ(Resources)
@u_empty unsat(c) = ∅
@a_empty alloc(c) = ∅
@S_ne S ≠ ∅
then
@upd unsat(c) ≔ unsat(c) ∪ S
end
Goal
Event-B Explorer
<Plug-in Development>
Plug-in Development
ac.soton.eventb.emf.inclusion.generator
ALLOC
Allocator
cSimpleAllocator
changeMe
mSimpleAllocator
Variables
Invariants
Events
Proof Obligations
ResourceMutex/WD
INITIALISATION/unsat_ty/INV
INITIALISATION/alloc_ty/INV
INITIALISATION/ResourceMutex/INV
Request/u_empty/WD
Request/a_empty/WD
Request/unsat_ty/INV
Request/upd/WD
Allocate/S_u/WD
Allocate/unsat_ty/INV
Allocate/alloc_ty/INV
Allocate/ResourceMutex/INV
Allocate/a_upd/WD
Allocate/u_upd/WD
Return/S_ok/WD
Return/alloc_ty/INV
Return/ResourceMutex/INV
Return/a_upd/WD
AxList
AxMath
AxMath1
BasicTheory
Symbols

Plan

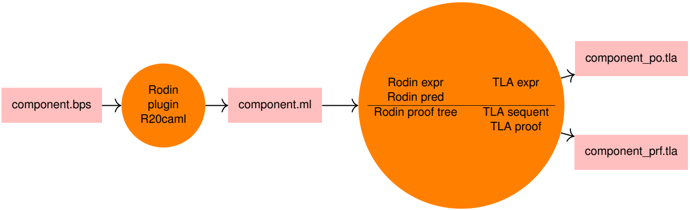
TLAPS as a proof environment for B/Event-B

- B/Event-B and TLA+ are both based on set theory.
- B/Event-B and TLA+ expressions are almost the same.
- Both proof languages adopt a ML approach (sequent based).

EB2TLA

- Event-B proof obligations are translated to TLA+ theorems to be proved.
- The Rodin generated Event-B proofs (proof tree) are translated to TLAPS proofs to discharge the TLA+ generated theorems (sequent + proof).

EB2TLA


component.bps
Rodin
plugin
R20caml
component.ml
Rodin expr
Rodin pred
Rodin proof tree
TLA expr
TLA sequent
TLA proof
component_po.tla
component_prf.tla

Proof obligations in TLA (SimpleAllocator_po)

––––––––––––– **MODULE** SimpleAllocator_po–––––––––––––––––
EXTENDS Naturals, Integers, FiniteSets, TLC, TLAPS
, Relations, Partitions
THEOREM SimpleAllocator_ResourceMutex_WD_po $\triangleq$
ASSUME NEW Clients, **NEW** Resources, **NEW** alloc $\in$ **SUBSET**((Clients $\times$ **SUBSET**(Resources))), (alloc $\in$ TotalFunctions(Clients, **SUBSET**(Resources)))
PROVE ($\forall$ c1 $\in$ Clients,c2 $\in$ Clients: ((c1 $\in$ Clients) $\wedge$ (c2 $\in$ Clients) $\wedge$ (c1 # c2) $\Rightarrow$ (c1 $\in$ Dom(alloc)) $\wedge$ (alloc $\in$ PartialFunctions(Clients, **SUBSET**(Resources))) $\wedge$ (c2 $\in$ Dom(alloc))))

```
THEOREM T_THM_rodin ≜
ASSUME NEW c1 ∈ Int, NEW c2 ∈ Int, (c2 ∈ Nat), (c1 ∈ Nat)
PROVE ((∃ x ∈ Int: (∃ y ∈ Int : (y ∈ Nat) ∧ (c1 > y) ∧ (c2 < x))) ⇔ (c1 >
     0))

<0> USE ProdSingleton, FunImageSingleton, OverwritePoint DEF Rel,
     TotalFunctions, PartialFunctions, Dom, Ran, PartialInjections, Rev,
     Surjections, PartialSurjections, TotalSurjections, Bijections , Overwrite,
     AntirestrictDom, FunImage, RImage
<0>0. ((∃ x ∈ Int: (∃ y ∈ Int : (y ∈ Nat) ∧ (c1 > y) ∧ (c2 < x))) ⇒ (c1 >
     0))
  <1>0. ((∃ x ∈ Int, y ∈ Int : (x ∈ Nat) ∧ (c1 > x) ∧ (c2 < y)) ⇒ (c1 > 0))
    <2>0. ASSUME (∃ x ∈ Int,y ∈ Int: (x ∈ Nat) ∧ (c1 > x) ∧ (c2 < y))
    PROVE (c1 > 0)
      <3>0. (c1 > 0)
 BY <2>0
      <3>1. QED  BY <3>0
    <2>1. QED  BY <2>0
```

(first) experiments-Feedback

(On going work)

- B/Event-B typed set theory helps.
- Many leafs of the proof tree are actually discharged thanks to SMT solvers.
- We have to devise strategies between full expansion of definitions and dedicated theorems. Instantiations of TLA theorems with some goal terms could help ?
- B/Event-B interactive approach remains appreciated.

Plan

1. Development processes in B/Event-B and TLA
   - TLA
   - B/Event-B
2. TLAPS as a proof environment for B/Event-B
3. A TLA development process à la B/Event-B (study)
4. Conclusion

TLA development process à la B/Event-B (1) (study)

- Starting point: TLA model with a “configuration” (Init, Next, Invariants, . . .)
- $\leadsto$ Generation of proof obligations.

```
---------------------- MODULE Allocator_po_1 ----------------------------
(* Proof squeletons generated for SimpleAllocator module *)
EXTENDS Allocator

THEOREM InitTypeInvariant ≜
  Init ⇒ TypeInvariant
  OMITTED
THEOREM RequestTypeInvariant ≜
  ASSUME NEW c ∈ Client,
         NEW S ∈ SUBSET Resource
  PROVE TypeInvariant ∧ Request(c,S) ⇒ TypeInvariant'
  OMITTED
```

TLA development process à la B/Event-B (2)

- ...
- $\rightsquigarrow$ Generation of meta theorems.

THEOREM NextTypeInvariant $\triangleq$ ($\star$TypeInvariant $\wedge$ Next $\Rightarrow$
TypeInvariant'$\star$)
ASSUME TypeInvariant, Next
PROVE TypeInvariant'
<1>1. **ASSUME NEW** c $\in$ Client, **NEW** S $\in$ **SUBSET** Resource,
TypeInvariant,
Request(c, S) $\vee$ Allocate(c, S) $\vee$ Return(c, S)
PROVE TypeInvariant'

Plan

- ICSPA project $\leadsto$ the study of proofs in B/Event-B and TLA.
- B/Event-B and TLA mathematical languages are quasi compatible at the syntax level.
- Study of a synthesis between:
  - the proof language of TLA.
  - the assisted development of proofs in B/Event-B
- TLAPS as a proof environment for B/Event-B seems reasonable.
- An environment for a TLA development process à la B/Event-B is to be investigated.