University of Southampton

Sandia National Laboratories

University of Southampton

Semantics Formalisation – Some Experience with the Theory Plug-in

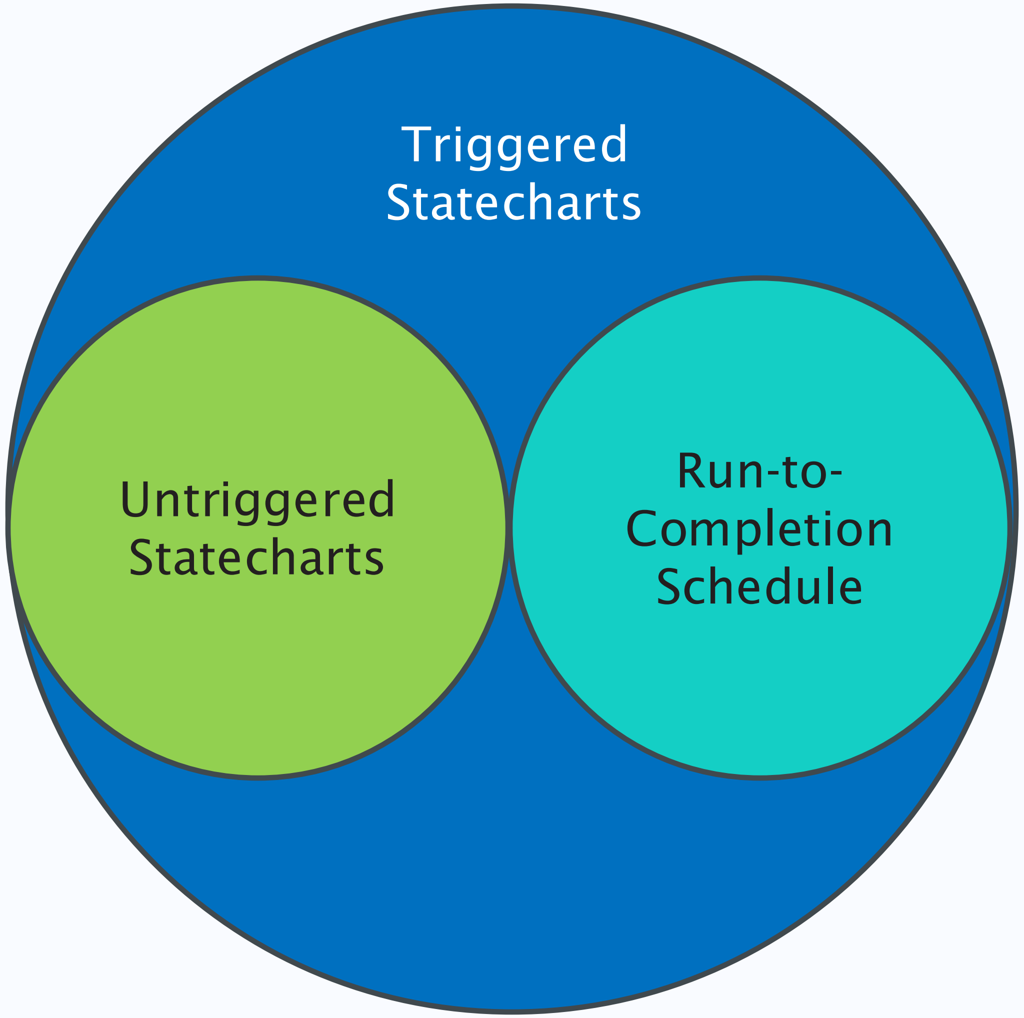
Thai Son Hoang,
Laurent Voisin,
Colin Snook,
Karla Vanessa Morris Wright,
Michael Butler

Rodin Workshop 2024
24 June 2024

Motivation

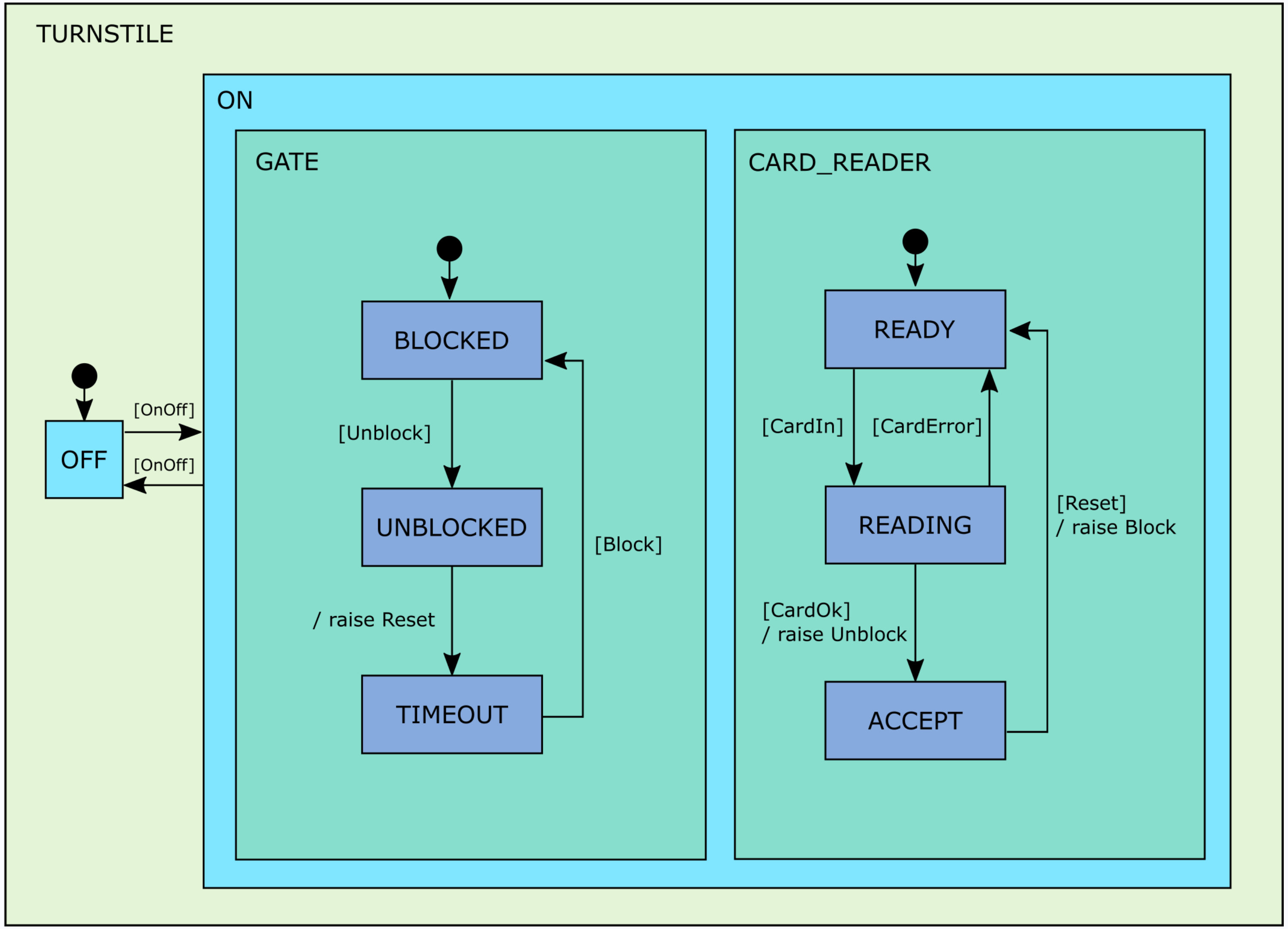
- SCXML = Statecharts + Run-to-completion scheduling
- Introduce the notion of refinement into SCXML (ECSA 2020, ABZ2020, ISSE2022)
- Formalization of the SCXML
  - Derive syntactical constraints (as axioms), i.e., well-definedness conditions for SCXML
  - Formal Language Semantics for Triggered Enable Statecharts with a Run-to-Completion Scheduling (ICTAC2023).
- This talk:
  - Recap of our ICTAC2023 work using Context/Machine
  - Some experience (modelling and proving) using the Theory Plug-in

Structure of our Semantics Definition

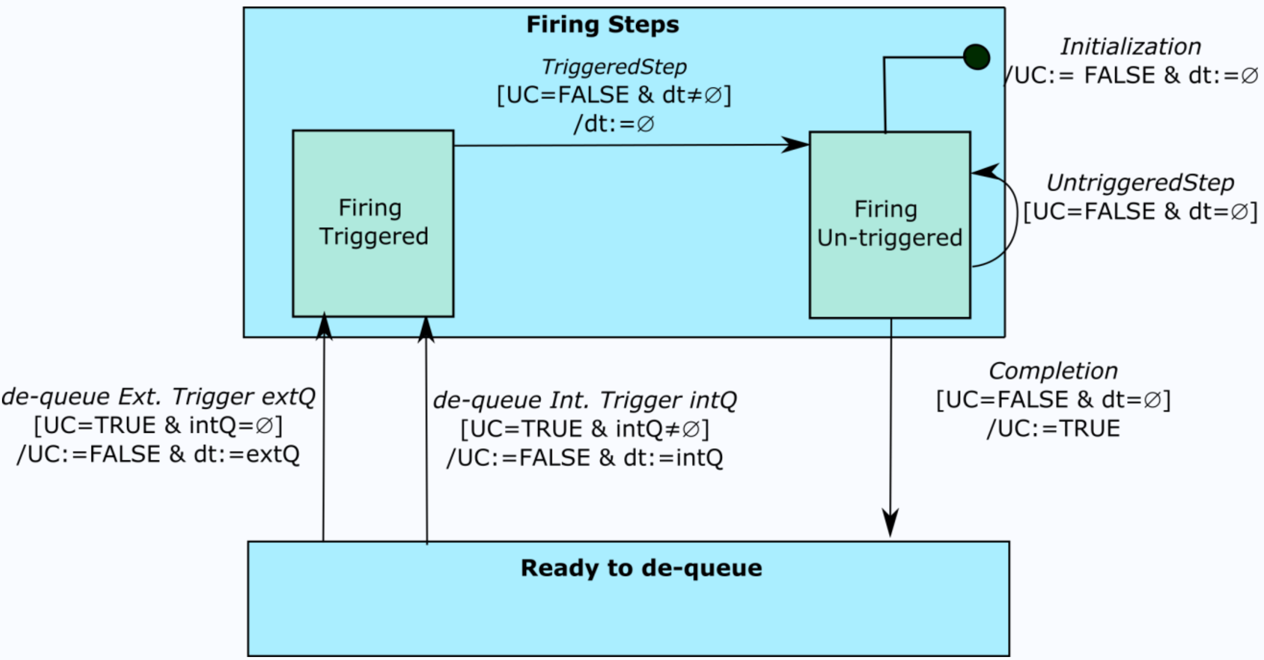


- Composition of 2 Event-B models
  - Contexts represent the “syntax”
  - Machines represent the “semantics”
- Separation of concerns
  - Reduce complexity
  - Reuse parts of the formalisation
    - E.g. we could combine an alternative schedule semantics with the untriggered statechart semantics
- Use Event-B composition mechanism (inclusion) to combine the constructs …
- … composition is correct-by-construction

Example. Turnstile



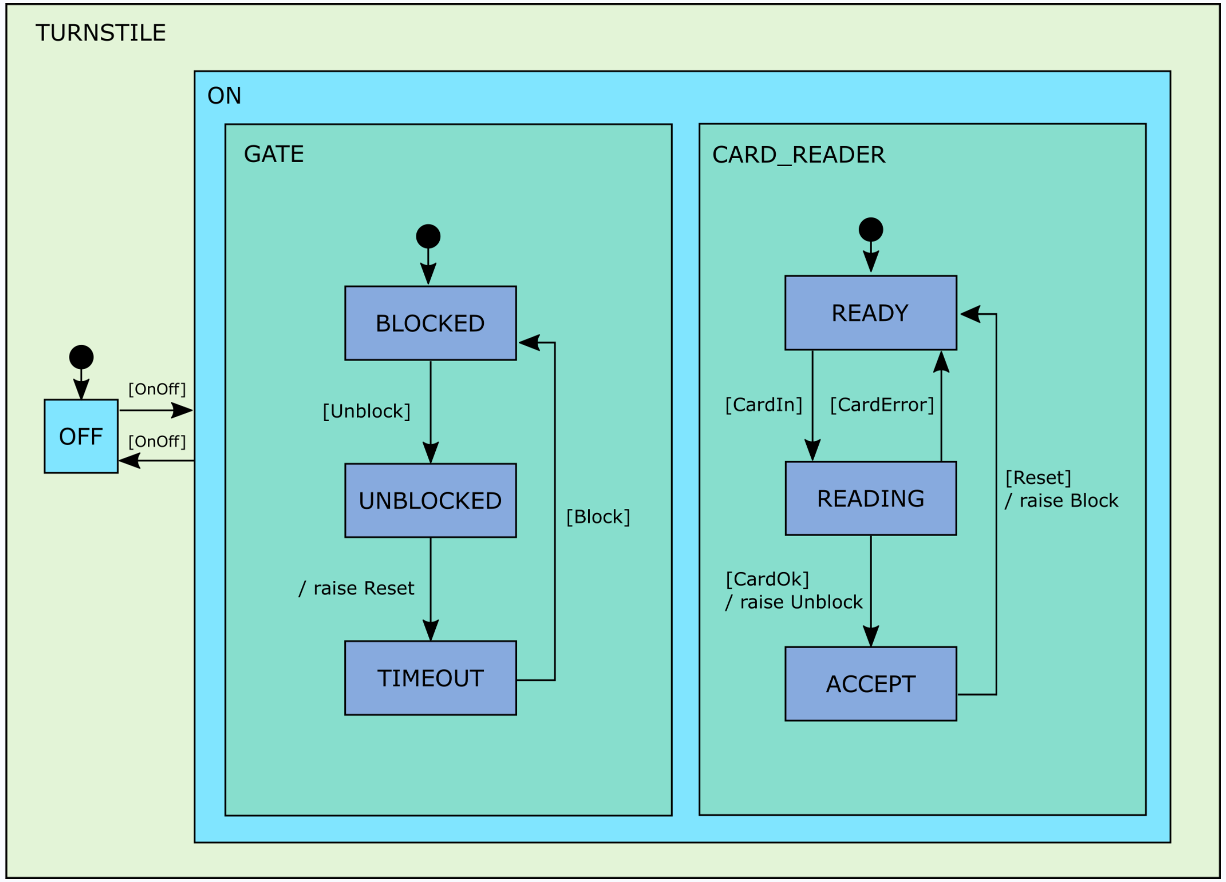
Background - SCXML Run-to-Completion



- State Chart eXtensible Markup Language (SCXML)
  - A general-purpose event-based statemachine language
- Trigger types:
  - internal triggers are raised by transitions
  - external triggers are raised non-deterministically
- Uses a run-to-completion semantics
  1. A trigger is de-queued
  2. All enabled triggered transitions are fired
  3. All enabled (un-triggered) transitions are fired
     - repeat until no un-triggered transitions are enabled
  4. De-queue next trigger
     - Internal triggers have priority

Untriggered Statechart Formalization

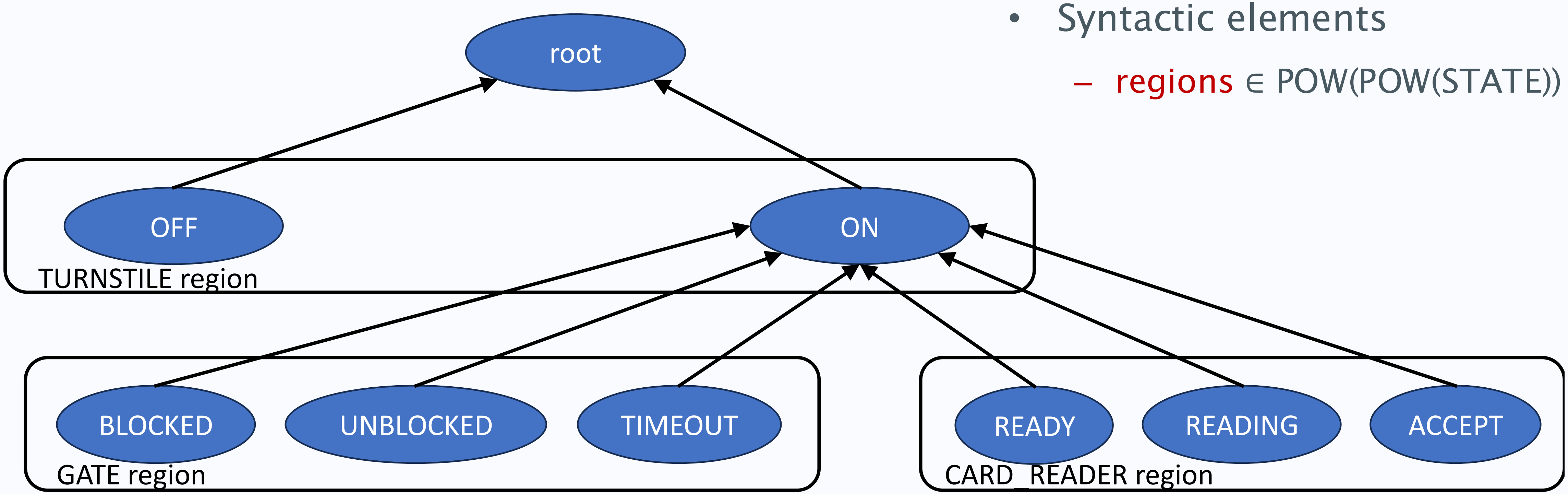
Syntactic Elements



- Syntactic elements
  - states ∈ POW(STATE)
  - root ∈ STATE
  - container ∈ STATE ↔ STATEs
- states = {root, OFF, ON, BLOCKED, UNBLOCKED, TIMEOUT, READY, READING, ACCEPT}
- container = {OFF ↦ root, ON ↦ root, BLOCKED ↦ ON, UNBLOCKED ↦ ON, TIMEOUT ↦ ON, READY ↦ ON, READING ↦ ON, ACCEPT ↦ ON}

Untriggered Statechart Formalization

Syntactic Elements (cont.)



- Syntactic elements
  - regions ∈ POW(POW(STATE))

```
regions = { {ON, OFF},                              // TURNSTILE
            {BLOCKED, UNBLOCKED, TIMEOUT},          // GATE
            {READY, READING, ACCEPT} }              // CARD_READER
```

Untriggered Statechart Formalization

Syntactic Elements (cont.)

- Context c0: Model the tree-structure of the states

  states ↦ root ↦ container ∈ Tree

- Context c1: Model the parallel regions

  regions *partition* children of a parent state

- Context c2: Model the transformations between states

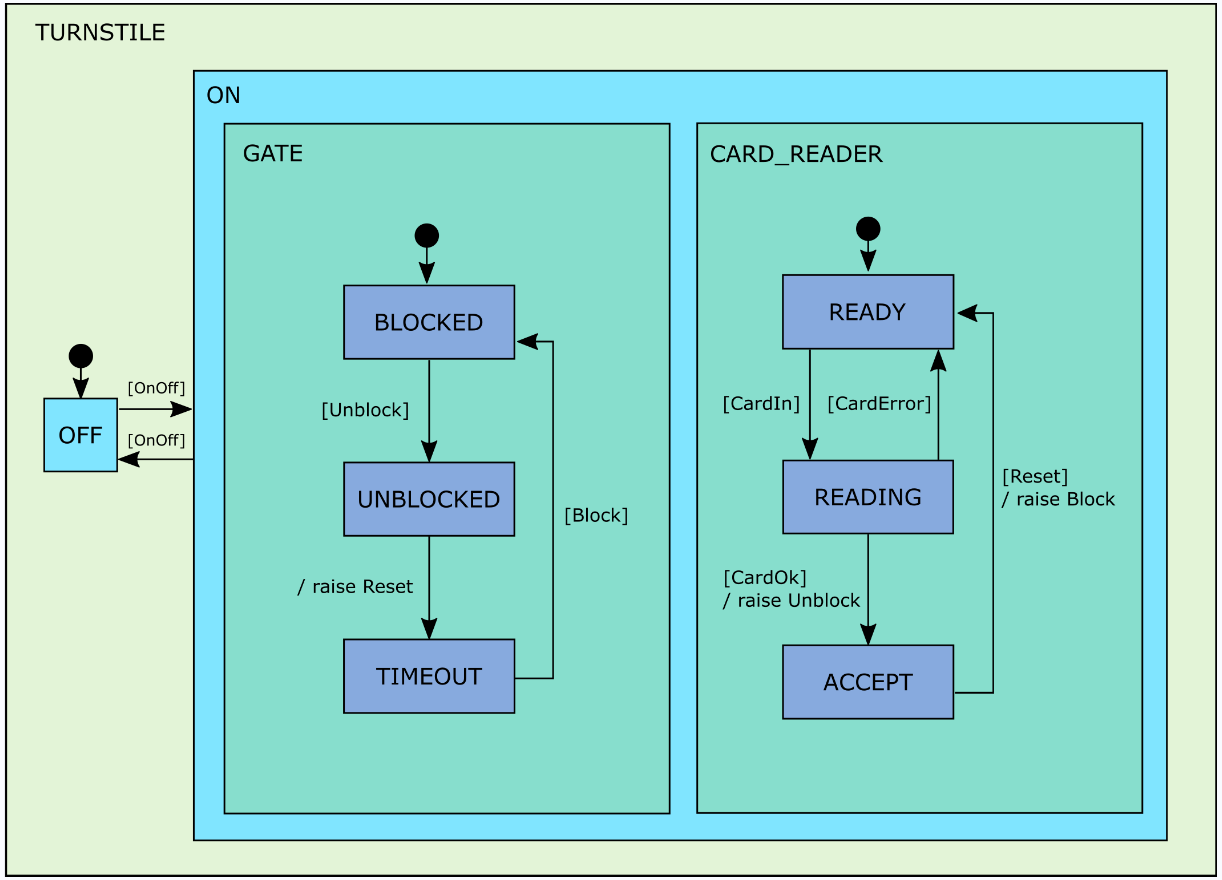
  a set of all simultaneously enabled system transitions, from one enabling configuration to the next.

  - enabling ∈ transformations → $\mathbb{P}1$(states)
  - exiting ∈ transformations → $\mathbb{P}$(states)
  - entering ∈ transformations → $\mathbb{P}$(states)

(where $\mathbb{P}$ = powerset,
$\mathbb{P}1$ = powerset excl. emptyset)

Untriggered Statechart Formalization

Syntactic Elements (cont.)



- transformation from ON to OFF
  - enabling = {ON}
  - exiting = {ON, BLOCKED, UNBLOCKED, TIMEOUT, READY, READING, ACCEPT}
  - entering = {OFF}
- transformation from OFF to ON
  - enabling = {OFF}
  - exiting = {OFF}
  - entering = {ON, BLOCKED, READY}

Sandia National Laboratories

Untriggered Statechart Formalization

Semantical Elements

University of Southampton

- We say that the *configuration* of a statechart is the set of currently active states
- *Active* is the only variable in the semantics model
- The event *transformation* fires any transformation whose enabling states are ALL active
- It then updates the configuration…
  - by removing the transformation's exiting states from active and
  - adding the transformation's entering states to active

Variable active ∈ POW(STATE)

```
event transformation
any trf where
  @typeof−trf: trf ∈ transformations
  @active−enabling: enabling(trf) ⊆ active
then
  @update−active: active := (active \ exiting(trf)) ∪ entering(trf)
end
```

Machine: Dynamic aspects of the untriggered statechart language model

Untriggered Statechart Formalization

Invariant Properties

Invariant 1 (Container active) *If a non-root state is active then its container is also active.*

$$@container\_active: \forall s \cdot s \in active \setminus root \Rightarrow container(s) \in active$$

Invariant 2 (Content active) *If a container state is active then one of its sub-state must be active.*

$$@content\_active: \forall s \cdot s \in ran(container) \land s \in active \Rightarrow container\sim[\{s\}] \cap active \neq \varnothing$$

Invariant 3 (Unique active state within a region) *There can be at most one active state in a region.*

$$@active{-}region{-}unique: \forall r, s \cdot r \in regions \land s \in r \cap active \Rightarrow r \cap active \subseteq \{s\}$$

Invariant 4 (Parallel regions are inactive/active at the same time) *All parallel regions are inactive (hence active) at the same time.*

$$@active{-}region{-}parallel: \forall r1, r2 \cdot r1 \in regions \land r2 \in regions \land container[r1] = container[r2] \land r1 \cap active = \varnothing \Rightarrow r2 \cap active = \varnothing$$

Untriggered Statechart Formalization

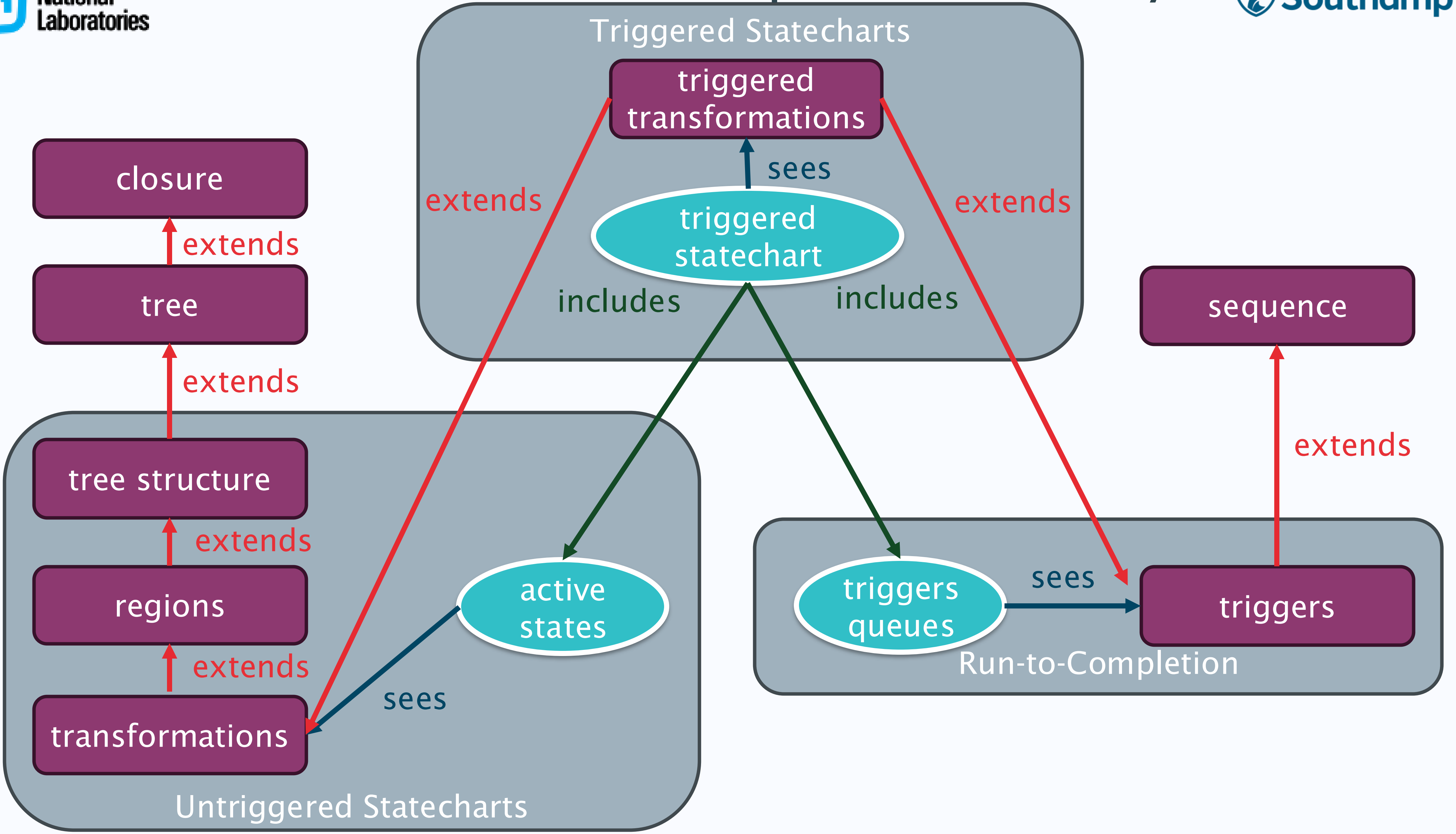
Axioms Discovery

- Question: What are the syntactical constraints for a statechart to ensure the transformations maintaining the invariants properties?
  - Constraints about enabling, exiting, entering
- Some constraints are defined based on experience
- Some other constraints derived from the proof, e.g.,

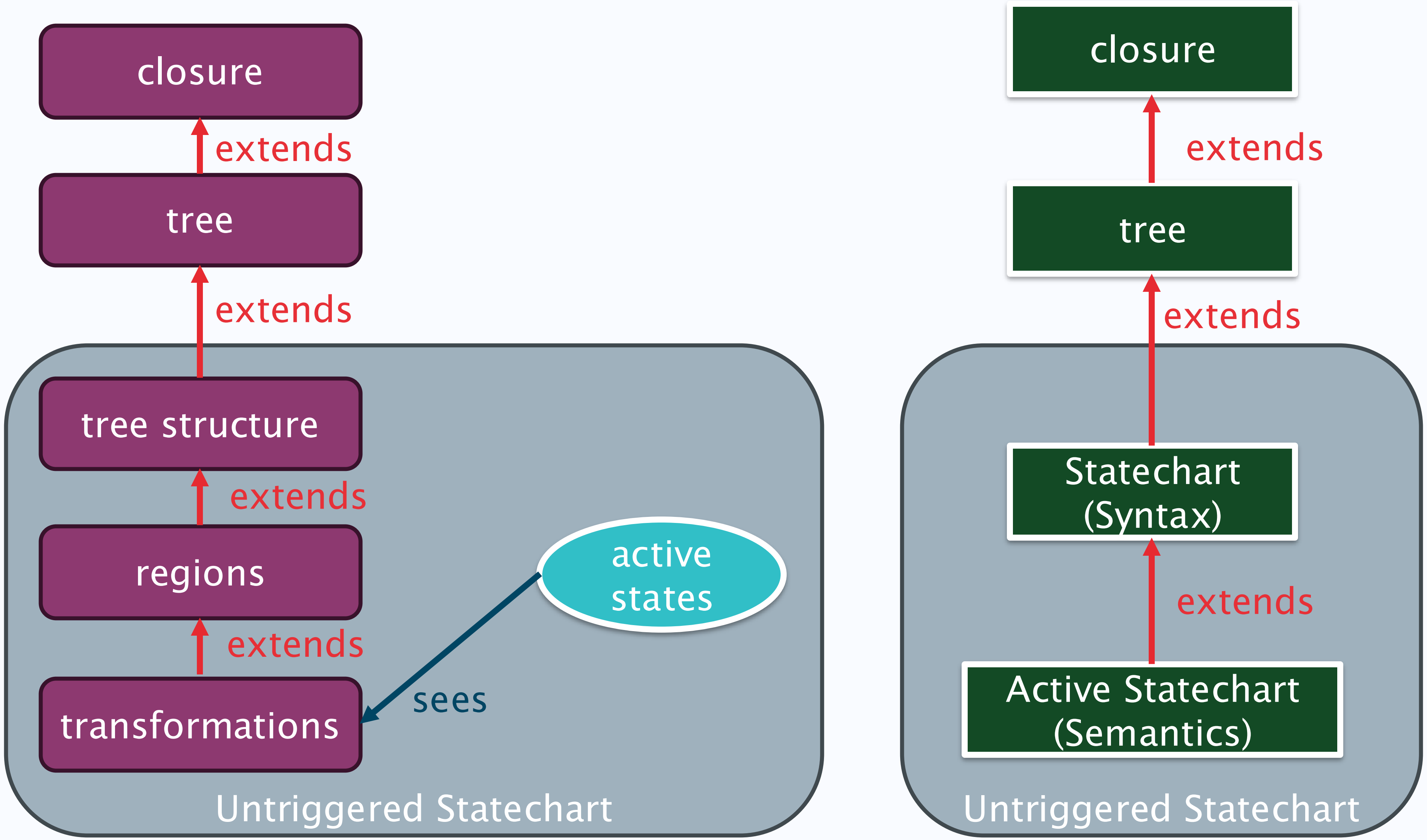
Axiom 7 (Exiting a unique enabling state in a region) *Given a region r and an exiting state s in r, if r has states other than s, s must be the unique enabling state in r.*

```
@exiting–unique_enabling_state_in_a_region:
  ∀trf, s, r · trf ∈ transformations ∧ r ∈ regions ∧ exiting(trf) ∩ r = {s} ∧ r ≠ {s}
     ⇒ enabling(trf) ∩ r = {s}
```

Overview of the Development Hierarchy



Using Theories



Closure

constants cl

axiom @def-cl: cl = (λ r · r ∈ STATE ↔ STATE | inter({p | r ⊆ p ∧ p;p ⊆ p}))

theorem @typeof-cl: cl ∈ (STATE ↔ STATE) → (STATE ↔ STATE)

theorem @thm1: ∀ r· r ⊆ cl(r)

theorem @thm2: ∀ r·cl(r);cl(r) ⊆ cl(r)

theorem @thm3: ∀ r·(∀ p· r⊆p ∧ p;p⊆p ⇒ cl(r)⊆p)

context closure

type parameters S

operators cl == (λ r · r ∈ S ↔ S | inter({p | r ⊆ p ∧ p;p ⊆ p}))

theorems

@typeof–cl: cl ∈ (S↔S)→(S↔S)

@thm1: ∀ r· r ⊆ cl(r)

@thm2: ∀ r·cl(r);cl(r) ⊆ cl(r)

@thm3: ∀ r·(∀ p· r⊆p ∧ p;p⊆p⇒cl(r)⊆p)

theory closure

Polymorphic operator

Tree

constants Tree

axiom @def-Tree: Tree = {Sts ↦ rt ↦ prn | Sts ⊆ STATE ∧ rt ∈ Sts ∧
prn ∈ Sts \ {rt} → Sts ∧ (∀ n · n ∈ Sts \ {rt} ⇒ rt ∈ cl(prn)[{n}])}

theorem @all_node: ∀ Sts, rt, prn · Sts ↦ rt ↦ prn ∈ Tree ⇒ Sts = cl(prn∼)[{rt}] ∪ {rt}

theorem @induction-Tree: ∀ Sts, rt, prn · Sts ↦ rt ↦ prn ∈ Tree ⇒
(∀ T · rt ∈ T ∧ prn∼[T] ⊆ T ⇒ Sts ⊆ T)

context tree

type parameters Tree

datatype TREE(NODE) == Cons_Tree(States:P(NODE), Root:NODE, Parent:P(NODE × NODE))

operator Tree_WD(tr:TREE(NODE)) ==
Root(tr) ∈ States(tr) ∧
Parent(tr) ∈ States(tr) \ {Root(tr)} → States(tr) ∧
(∀ n · n ∈ States(tr) \ {Root(tr)} ⇒ Root(tr) ∈ cl(Parent(tr))[{n}])

theorem @all_node: ∀ tr · tr ∈ TREE(NODE) ∧ TreeWD(tr) ⇒
States(tr) = cl(Parent(tr)∼)[{Root(tr)}] ∪ {Root(tr)}

theory tree

Polymorphic datatype

Rewrite/Inference rules (later)

Statechart (1/2)

constants Sts rt prn regions transformations enabling exiting entering

axiom @tree_structure: $Sts \mapsto rt \mapsto prn \in Tree$

axiom @regions_type: $regions \subseteq \mathbb{P}(Sts)$

axiom @region_disjoint: $\forall r1, r2 \cdot r1 \in regions \land r2 \in regions \land r1 \neq r2 \Rightarrow r1 \cap r2 = \emptyset$

// Other axioms about enabling, exiting, and enabling

…

contexts tree_structure, regions, statechart

type parameters NODE TRAANSFORMATION

datatype STATECHART(NODE, TRANSFORMATION) ==
Cons_Statechart(

Tree : TREE(STATE),

Regions : P(P(STATE)),

Transformation : P(TRANSFORMATION),

Enabling : P(TRANSFORMATION × P(STATE)),

Exiting : P(TRANSFORMATION × P(STATE)),

Entering : P(TRANSFORMATION × P(STATE)))

theory statechart

Polymorphic datatype

Statechart (Syntactic) (2/2)

contexts tree_structure, regions, statechart

constants Sts rt prn regions transformations enabling exiting entering

axiom @tree_structure: Sts ↦ rt ↦ prn ∈ Tree

axiom @regions_type: regions ⊆ ℙ(Sts)

axiom @region_disjoint: ∀r1, r2 · r1 ∈ regions ∧ r2 ∈ regions ∧ r1 ≠ r2 ⇒ r1 ∩ r2 = ∅

// Other axioms about regions, transformations, enabling, exiting, and enabling

…

theory statechart

operators Regions_WD(st: STATECHART(STATE, TRANSFORMATION)) ==
Regions(st) ⊆ P(States(Tree(st))) ∧
(∀r1, r2 · r1 ∈ Regions(st) ∧ r2 ∈ Regions(st) ∧ r1 ≠ r2⇒r1 ∩ r2 = ∅) ∧ …

operators Transformations_WD(st: STATECHART(STATE, TRANSFORMATION)) == …

operators Statechart_WD(st: STATECHART(STATE, TRANSFORMATION)) ==
Tree_WD(st) ∧ Regions_WD(st) ∧ Transformations_WD(st)

Active Statechart (Semantics) (1/2)

Sandia National Laboratories — University of Southampton

machine
active states

variables active

invariant @typeof-active: $active \subseteq Sts$
invariant @container_active: $\forall s \cdot s \in active \setminus \{root\} \Rightarrow container(s) \in active$

invariant @content_active: $\forall s \cdot s \in ran(container) \wedge s \in active \Rightarrow container^{\sim}[\{s\}] \cap active \neq \emptyset$
invariant @active-region-unique: $\forall r, s \cdot r \in regions \wedge s \in r \cap active \Rightarrow r \cap active \subseteq \{s\}$

invariant @active-region-parallel: $\forall region1, region2 \cdot region1 \in regions \wedge region2 \in regions \wedge$
$container[region1] = container[region2] \wedge region1 \cap active = \emptyset \Rightarrow region2 \cap active = \emptyset$

theory
active_statechart

datatype ACTIVE_STATECHART(STATE) == Cons_ActiveStatechart(Active : P(STATE))

operators ActiveStatechart_WD(

sc: STATECHART(STATE, TRANSFORMATION), asc: ACTIVE_STATECHART(STATE)) ==
$Active(asc) \neq \emptyset$
$\wedge\ Active(asc) \subseteq States(Tree(sc))$
$\wedge\ (\forall s \cdot s \in Active(asc) \setminus \{Root(Tree(sc))\} \Rightarrow Parent(Tree(sc))(s) \in Active(asc))$
$\wedge\ (\forall s \cdot s \in ran(Parent(Tree(sc))) \wedge s \in Active(asc) \Rightarrow Parent(Tree(sc))^{\sim}[\{s\}] \cap Active(asc) \neq \emptyset)$
$\wedge\ (\forall r, s \cdot r \in Regions(sc) \wedge s \in r \cap Active(asc) \Rightarrow r \cap Active(asc) \subseteq \{s\})$
$\wedge\ (\forall r1, r2 \cdot r1 \in Regions(sc) \wedge r2 \in Regions(sc) \wedge$
$Parent(Tree(sc))[r1] = Parent(Tree(sc))[r2] \wedge r1 \cap Active(asc)=\emptyset \Rightarrow r2 \cap Active(asc)=\emptyset)$
for Statechart_WD(sc)

Active Statechart (Semantics) (2/2)

Sandia National Laboratories

University of Southampton

machine active states

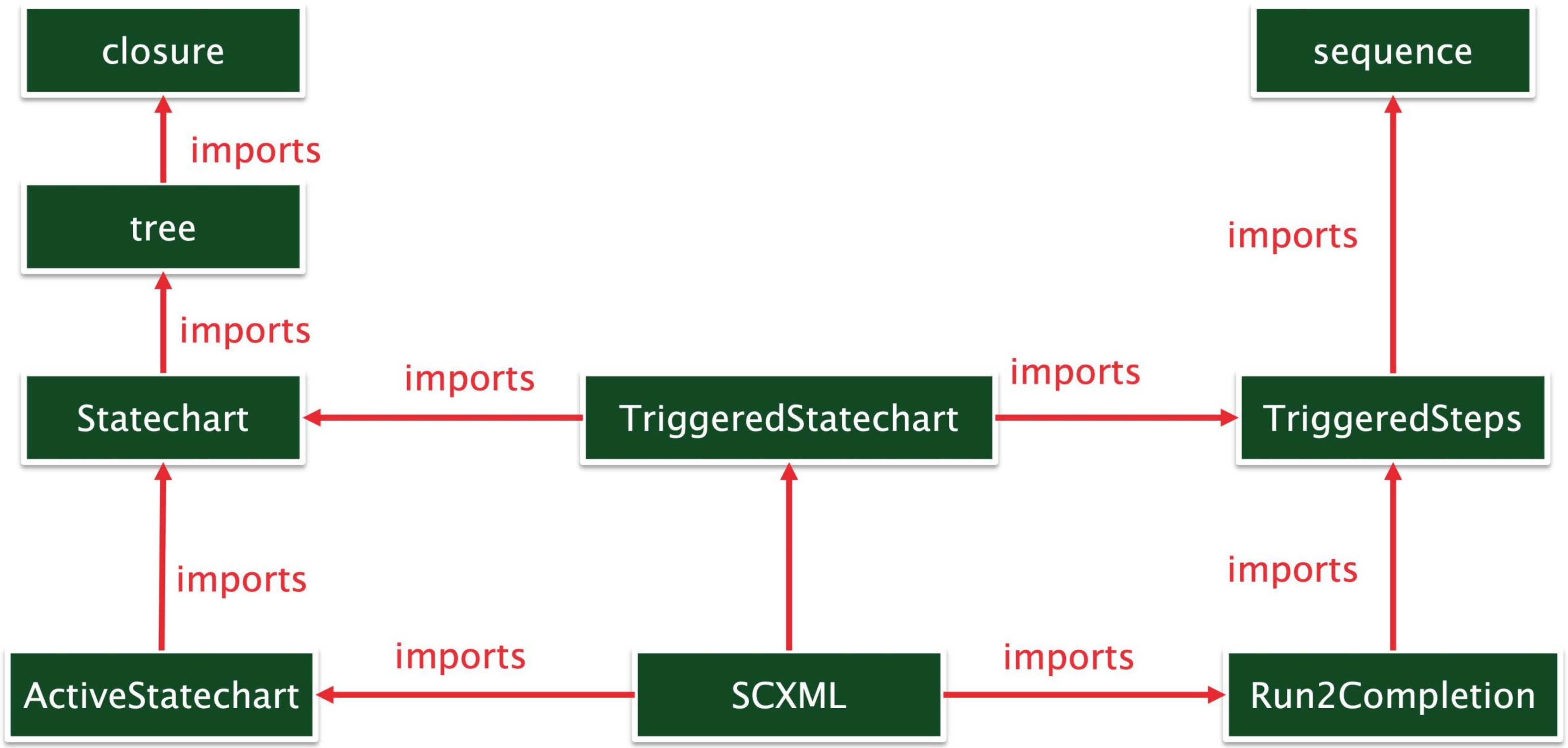
```
event transformation
any trf where
        @typeof-trf: trf ∈ transformations
        @active-enabling: enabling(trf) ⊆ active
then @update-active: active ≔ (active \ exiting(trf)) ∪ entering(trf)
end
```

theory active_statechart

```
operators transform(sc: STATECHART(STATE, TRANSFORMATION),
        asc: ACTIVE_STATECHART(STATE), tr: TRANSFORMATION) ==
        Cons_ActiveStatechart((Active(asc) \ Exiting(sc)(tr)) ∪ Entering(sc)(tr))
for Statechart_WD(sc) ∧ ActiveStatechart_WD(sc, asc) ∧ tr ∈ Transformations(sc) ∧
        Enabling(sc)(tr) ⊆ Active(asc) // WD condition

theorem @statechart_consistency: ∀sc, asc, tr ·
                sc ∈ STATECHART(STATE, TRANSFORMATION) ∧ asc ∈ ACTIVE_STATECHART(STATE)
                ∧ Statechart_WD(sc) ∧ ActiveStatechart_WD(asc, sc) ∧ tr ∈ Transformation(sc)
                ∧ Enabling(sc)(tr) ⊆ Active(asc)
        ⇒
                ActiveStatechart_WD(transform(sc, asc, tr))
```

Hierarchy of Theories



Extending the Theory Provers

1. Definition Expansion (RbPxd)
2. Rewrite rule (RbP0)
3. Inference rules (RbP1)

The above tactics can be configured for individual projects, but NOT for particular components (e.g., an individual theory).

- Use as part of the automatic tactic
- Use as part of post-tactic (i.e., applied automatically after every interactive step).

Definition Expansion (RbPxd)

- Automatic expansion of operator definitions
- Expansion of all operators in scope
  - Expanding Tree_WD, then expanding the definition of closure (cl)
- Useful when proving properties of the operators.
- However, this removes the “encapsulation” of the operator definitions.
  - Ideally, there should be sufficient proof rules about the operators
  - There is no need to expand the definition of operators when proving these proof rules.
- Suggestion, introducing the scope for the definition expansion tactic.
  - Expansion within a project scope or the same construct.

Rewrite Rules (RbP0)

Sandia National Laboratories

University of Southampton

- Rewrite a formula to another formula.
- Can be automatic or interactive or both (set by the users).
- Useful for normalisation
- Care should be taken when deciding if the rules should be applied automatically.
  - Rewrite formula F to true (⊤):
    - Useful in goal (but this can be done by an inference rule).
    - Less useful, if rewrites a hypothesis to true (i.e, removing the hypothesis)
  - Rewrite a simple formula on the LHS with a complex formula on the RHS

theorem @all_node: ∀ tr · tr ∈ TREE(NODE) ∧ TreeWD(tr) ⇒
States(tr) = cl(Parent(tr)∼)[{Root(tr)}] ∪ {Root(tr)}

TreeWD(tr) ⇒
States(tr) == cl(Parent(tr)∼)[{Root(tr)}] ∪ {Root(tr)}

Inference Rules (RbP1)

- Can be automatic or interactive or both (set by the users).
- Can be applied backwardly or forwardly (depending on the proof context)
- Some rules are only useful when applied in one direction

theorem @all_node: $\forall\ tr \cdot tr \in TREE(NODE) \wedge TreeWD(tr) \Rightarrow$
$States(tr) = cl(Parent(tr)\sim)[\{Root(tr)\}] \cup \{Root(tr)\}$

TreeWD(tr) (in hypothesis)

$States(tr) = cl(Parent(tr)\sim)[\{Root(tr)\}] \cup \{Root(tr)\}$

- (Currently, there is a bug in the reasoner that allows infinite forward application of an inference rule.)

Others

- How about theorems?
- Can we instantiate them automatically (within a certain scope)?
- How we can “generate” proof rules automatically from a theorem.

Contexts/Machines vs Theories

Approach 1. Standard Event-B	Approach 2. Theory Plug-in
– Model a single SCXML statechart	+ Model a datatype of SCXML statecharts
= Syntactical elements are captured using contexts	= Syntactical elements are captured using theories
+ Syntactical elements are gradually added to the model using context extension	– Gradually introduce syntactical elements results in nested datatype
= Syntactic constraints are represented as context axioms	= Syntactic constraints are represented as WD operators
– Combination of different parts of the language using the composition plugin (i.e., outside of standard Event-B)	+ Composition is done by defining composite datatypes.
= Semantical consistency is encoded as machine invariants	= Semantical consistency is enconded as theory theorems
+ Consistency proof obligations are decomposed automatically (per individual invariants)	– Must manually construct theorems for decomposing the consistency proof
– No customisation for the provers to discharge proof obligations	+ Define proof rules for the provers to discharge proof obligations
– Model-related properties (e.g., refinement) requires additional tool	+ Model-related properties (e.g., refinement) can be stated as theory theorems

Table 1. Comparison between standard Event-B and Theory plug-in

YOUR QUESTIONS