Your Connection to ICT Research

Building Event-B Interlocking Theories

Lessons Learned Using the Theory Plug-in

23/05/2016

Yoann Guyot Renaud De Landtsheer Christophe Ponsard

PLUS HAUT
ET PLUS PROCHE
LE FONDS EUROPÉEN DE DÉVELOPPEMENT RÉGIONAL
ET LA WALLONIE INVESTISSENT DANS VOTRE AVENIR

FEDER
UNION EUROPEENNE

Wallonie

Centre d'Excellence en **Technologies** de l'**Information** et de la **Communication**
www.cetic.be

Who Am I ?

Yet another formal methods research engineer…

...with a bit of interest in the event-b method.

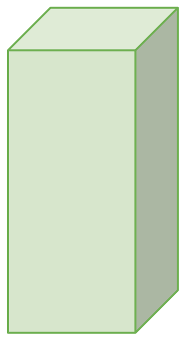
Before @ Systerel

Coded for the SMT Solvers Plug-in of Rodin

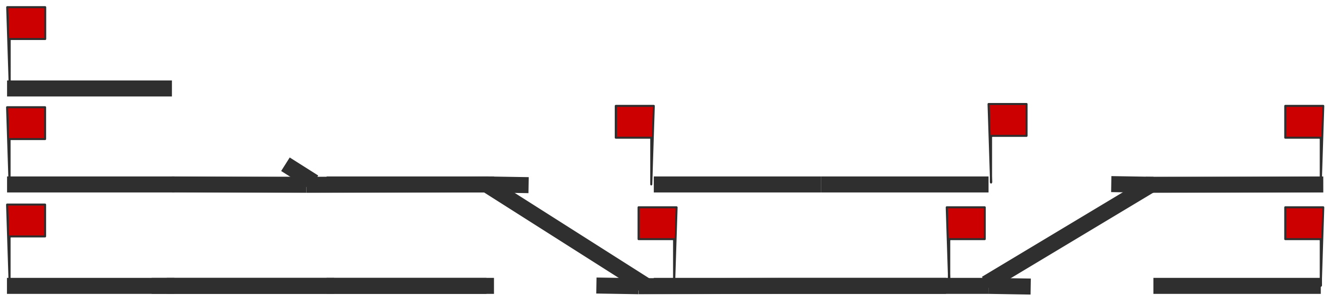
Nowadays @ **CETIC** : *Technology Transfert*

Experimenting with the Theory Plug-in of Rodin

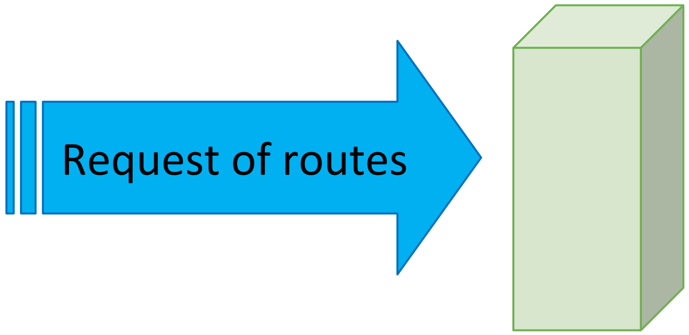
What is an Interlocking ?



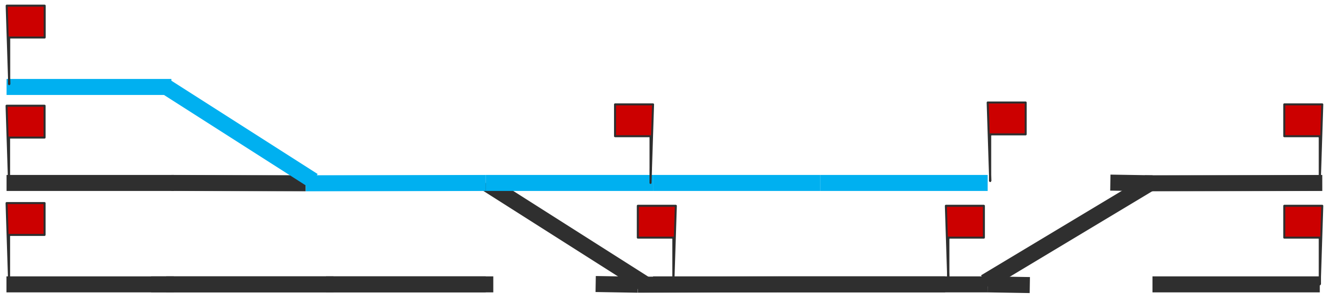
A Railway Signalling System



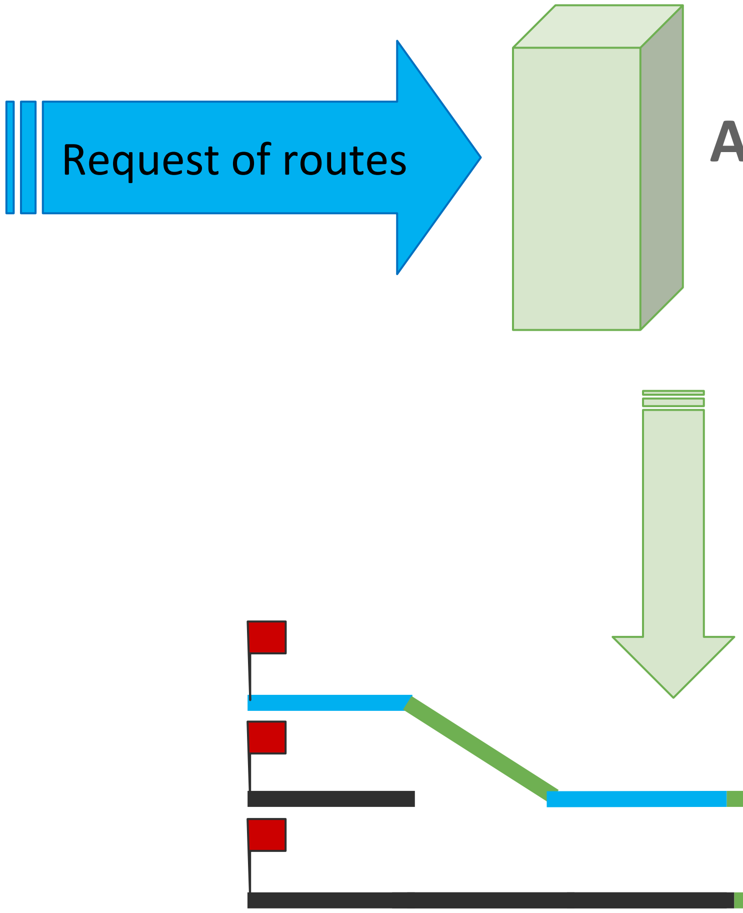
What is an Interlocking ?



A Railway Signalling System



What is an Interlocking ?

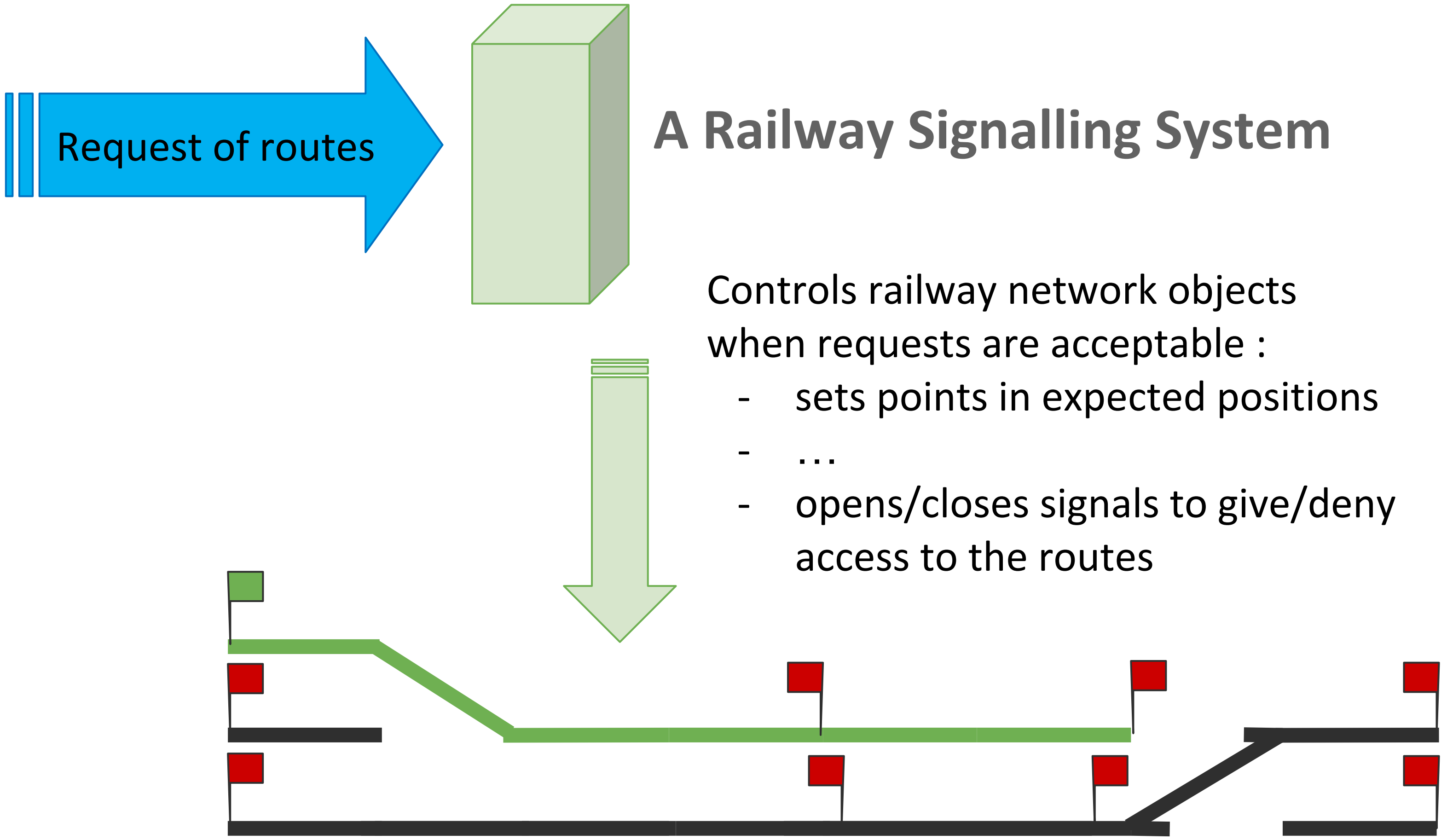


A Railway Signalling System

Controls railway network objects when requests are acceptable :

- sets points in expected positions

What is an Interlocking ?

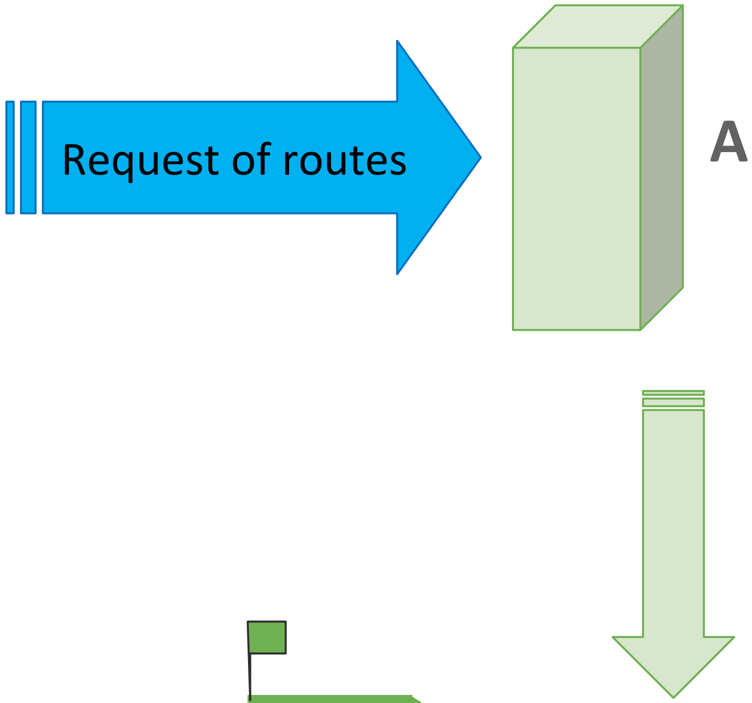


A Railway Signalling System

Controls railway network objects when requests are acceptable :

- sets points in expected positions
- …
- opens/closes signals to give/deny access to the routes

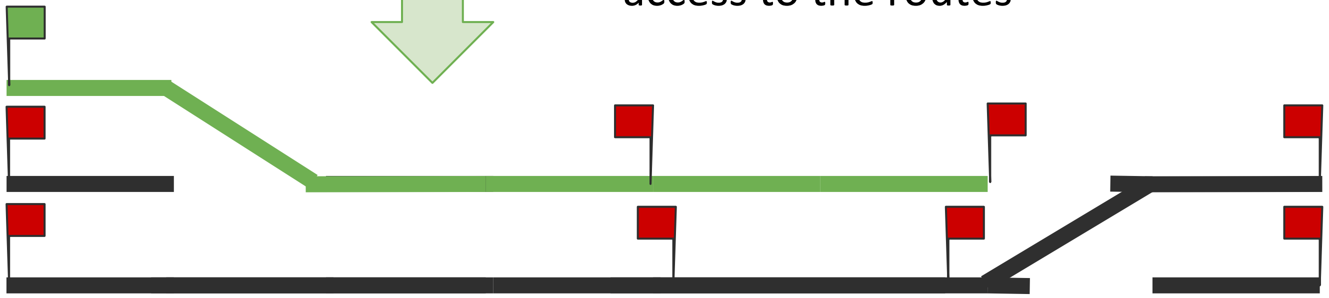
What is an Interlocking ?



A Railway Signalling System

Controls railway network objects when requests are acceptable :

- sets points in expected positions
- …
- opens/closes signals to give/deny access to the routes



Trains move on the network without colliding with each others nor derailing…

Who Makes Interlockings ?

Signalling engineers…

- possess the historical knowledge
- incrementally design new interlockings
- directly produce specific code
  based on generic programming rules
- *validate by reviewing and testing*

Who Makes Interlockings ?

Signalling engineers…

- possess the historical knowledge
- incrementally design new interlockings
- directly produce specific code
  based on generic programming rules
- *validate by reviewing and testing*

What about mathematical
proof of safety ?

- are generally *not* fluent in formal methods
- are especially *not* event-b lovers

Outline

How to Bring These Worlds Together

Use the Theory Plug-in of Rodin[1]

"Cover up those mathematical expressions which I cannot endure…"[2]

Defining Interlocking Theories

Reducing the Proof Effort

[1]Modeling a Safe Interlocking Using the Event-B Theory Plug-in, M-T. Khuu, L. Voisin and F. Mejia

[2](almost) Molière's Tartuffe

Covering Mathematical Expressions

The Famous Train Example First Context

axm1: blocks_routes ∈ BLOCKS ↔ ROUTES

axm2: dom(blocks_routes) = BLOCKS

axm3: ran(blocks_routes) = ROUTES

axm4: next ∈ ROUTES → (BLOCKS □ BLOCKS)

axm5: fst ∈ ROUTES → BLOCKS

axm6: last ∈ ROUTES → BLOCKS

axm7: fst~ ⊆ blocks_routes

axm8: last~ ⊆ blocks_routes

axm9: ∀r · r ∈ ROUTES ⇒ fst(r) ≠ last(r)

axm10: ∀r · r ∈ ROUTES ⇒ next(r) ∈ blocks_routes~[{r}] ∖ {last(r)} □ blocks_routes~[{r}] ∖ {fst(r)}

axm11: ∀r · r ∈ ROUTES ⇒ (∀S · S ⊆ next(r)[S] ⇒ S = ∅)

axm12: ∀r1, r2 · r1 ∈ ROUTES ∧ r2 ∈ ROUTES ∧ r1 ≠ r2
⇒ fst(r1) ∉ blocks_routes~[{r2}] ∖ {fst(r2), last(r2)}

axm13: ∀r1, r2 · r1 ∈ ROUTES ∧ r2 ∈ ROUTES ∧ r1 ≠ r2
⇒ last(r1) ∉ blocks_routes~[{r2}] ∖ {fst(r2), last(r2)}

The Famous Train Example First Context

axm1: $blocks\_routes \in BLOCKS \leftrightarrow ROUTES$

axm2: $dom(blocks\_routes) = BLOCKS$

axm3: $ran(blocks\_routes) = ROUTES$

Objects definitions

axm4: $next \in ROUTES \rightarrow (BLOCKS\ \square\ BLOCKS)$

axm5: $fst \in ROUTES \rightarrow BLOCKS$

axm6: $last \in ROUTES \rightarrow BLOCKS$

axm7: $fst^{\sim} \subseteq blocks\_routes$

axm8: $last^{\sim} \subseteq blocks\_routes$

Operators for manipulating the objects

Expected properties of the objects

axm9: $\forall r \cdot r \in ROUTES \Rightarrow fst(r) \neq last(r)$

axm10: $\forall r \cdot r \in ROUTES \Rightarrow next(r) \in blocks\_routes^{\sim}[\{r\}] \setminus \{last(r)\}\ \square\ blocks\_routes^{\sim}[\{r\}] \setminus \{fst(r)\}$

axm11: $\forall r \cdot r \in ROUTES \Rightarrow (\forall S \cdot S \subseteq next(r)[S] \Rightarrow S = \varnothing)$

axm12: $\forall r1, r2 \cdot r1 \in ROUTES \land r2 \in ROUTES \land r1 \neq r2$
$\Rightarrow fst(r1) \notin blocks\_routes^{\sim}[\{r2\}] \setminus \{fst(r2), last(r2)\}$

axm13: $\forall r1, r2 \cdot r1 \in ROUTES \land r2 \in ROUTES \land r1 \neq r2$
$\Rightarrow last(r1) \notin blocks\_routes^{\sim}[\{r2\}] \setminus \{fst(r2), last(r2)\}$

What Is Our Goal ?

Be as close as possible to the signaling engineers usage:

- Implicit objects definitions (blocks, routes, signals…)
- Implicit operators definitions (reserve, lock, open...)
- Rich DSL
- Implicit objects basic properties

Just model *this* interlocking.

The Famous Train Example First Context

axm1: $\text{blocks\_routes} \in \text{BLOCKS} \leftrightarrow \text{ROUTES}$

axm2: $\text{dom}(\text{blocks\_routes}) = \text{BLOCKS}$

axm3: $\text{ran}(\text{blocks\_routes}) = \text{ROUTES}$

axm4: $\text{next} \in \text{ROUTES} \rightarrow (\text{BLOCKS} \;\square\; \text{BLOCKS})$

axm5: $\text{fst} \in \text{ROUTES} \rightarrow \text{BLOCKS}$

axm6: $\text{last} \in \text{ROUTES} \rightarrow \text{BLOCKS}$

axm7: $\text{fst}^{\sim} \subseteq \text{blocks\_routes}$

axm8: $\text{last}^{\sim} \subseteq \text{blocks\_routes}$

axm9: $\forall r \cdot r \in \text{ROUTES} \Rightarrow \text{fst}(r) \neq \text{last}(r)$

axm10: $\forall r \cdot r \in \text{ROUTES} \Rightarrow \text{next}(r) \in \text{blocks\_routes}^{\sim}[\{r\}] \setminus \{\text{last}(r)\} \;\square\; \text{blocks\_routes}^{\sim}[\{r\}] \setminus \{\text{fst}(r)\}$

axm11: $\forall r \cdot r \in \text{ROUTES} \Rightarrow (\forall S \cdot S \subseteq \text{next}(r)[S] \Rightarrow S = \varnothing)$

axm12: $\forall r1, r2 \cdot r1 \in \text{ROUTES} \land r2 \in \text{ROUTES} \land r1 \neq r2$

$\Rightarrow \text{fst}(r1) \notin \text{blocks\_routes}^{\sim}[\{r2\}] \setminus \{\text{fst}(r2), \text{last}(r2)\}$

axm13: $\forall r1, r2 \cdot r1 \in \text{ROUTES} \land r2 \in \text{ROUTES} \land r1 \neq r2$

$\Rightarrow \text{last}(r1) \notin \text{blocks\_routes}^{\sim}[\{r2\}] \setminus \{\text{fst}(r2), \text{last}(r2)\}$

Using Our Theories

axm1:	ROUTES $\subseteq$ routes(BLOCKS)
axm2:	wellFormedRoutes(ROUTES)

Using Our Theories

Concrete set of routes (event-b constant)

Definition of *routes* constructor is hidden in the theories

Concrete set of blocks (event-b constant)

axm1: ROUTES ⊆ routes(BLOCKS)

axm2: wellFormedRoutes(ROUTES)

Additional properties of routes

- routes don’t start in the middle of others
- routes don’t end in the middle of others

The Famous Train Example First Machine

Without Theories

```
route_reservation:
ANY r WHERE
   r ∉ res_routes

   blocks_routes~[{r}] ∩ res_blocks = ∅
THEN
   res_routes ≔ res_routes ∪ {r}
   resbl_resrt ≔ resbl_resrt ∪ (blocks_routes ▷ {r})
   res_blocks ≔ res_blocks ∪ blocks_routes~[{r}]
END
```

Using Our Theories

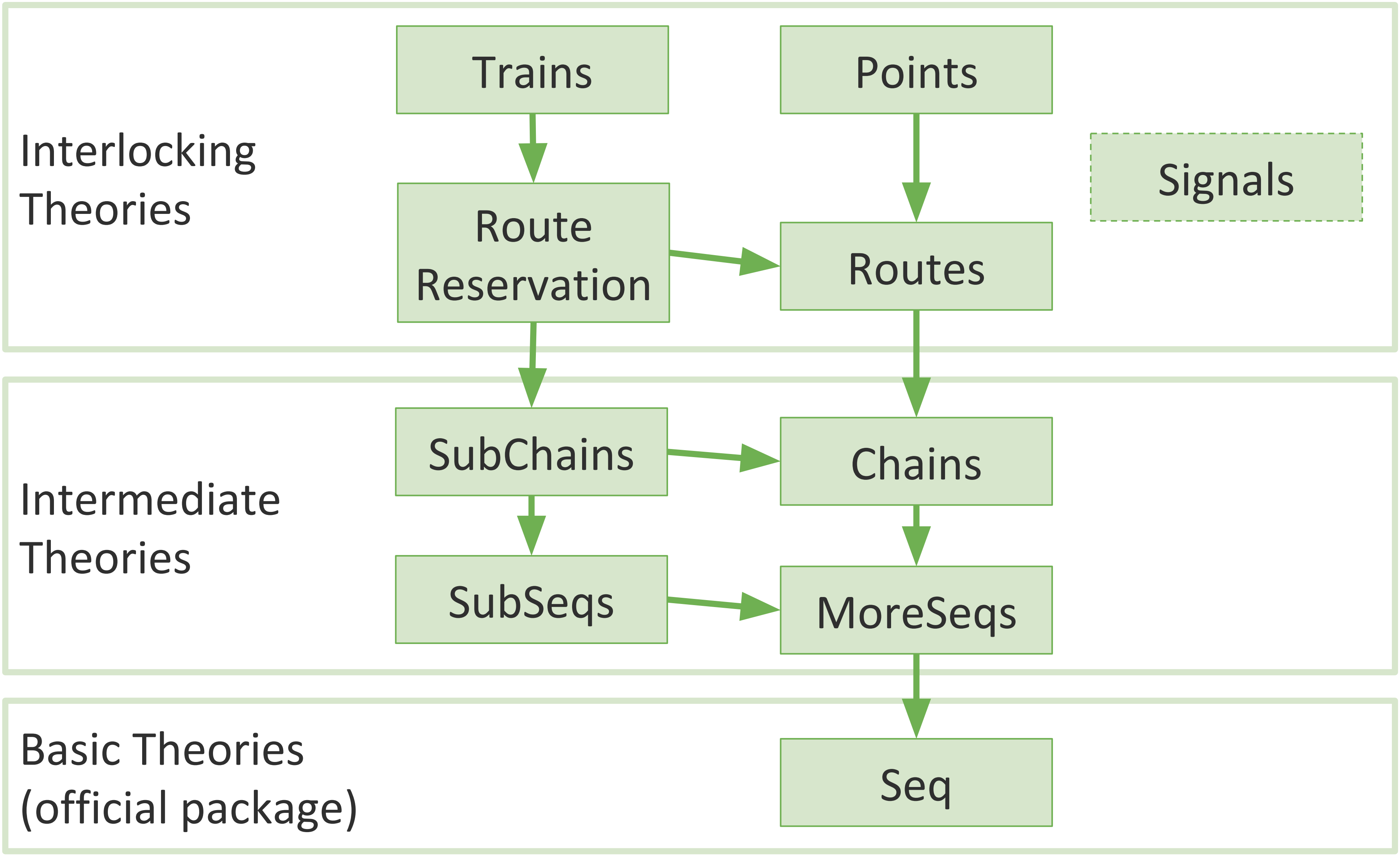
```
route_reservation:
ANY r WHERE
   r ∈ ROUTES
   ¬ isReserved(r, res_routes)
   noReservedBlocks(r, res_routes)
THEN
   res_routes ≔ res_routes ∪ completeRes(r)


END
```

Your Connection to ICT Research

Defining Interlocking Theories

Interlocking Theories Dependencies



Chains

Railway network is made of chains of blocks.

$\{ c \mid c \in seq(S) \land c^{\sim} \in T \mathrel{+\!\!\!\rightarrow} \mathbb{Z} \}$

equivalent to iseq
garantees that there is no cycle

Operators:

- chains (constructor)
- chainSize
- emptyChain
- chainFst / Last / Tail
- chainPrev / Next
- chainPrepend / Append
- …

Theorems:

- chainIsFinite
- chainIsMonotonic
- nextIsInRange
- chainTailIsChain
- chainPrependIsChain
- …

Routes

Routes are chains of at least two blocks.

$\{ c \mid c \in \text{chains(blocks)} \wedge \text{card}(c) > 1 \}$

Operators:

- routes (constructor)
- blcks
- routeLength
- routeFst / Last / Tail
- routePrev / Next
- starts / endsInTheMiddle
- wellFormedRoutes
- ...

Theorems:

- routeIsFinite
- routeIsMonotonic
- routeIsFunc
- routeDomain
- routeIsChain
- ...

Routes (How To Use It ?)

Routes are chains of at least two blocks.

$\{ c \mid c \in \text{chains}(\text{blocks}) \land \text{card}(c) > 1 \}$

• r is a route	$r \in \text{routes}(\text{BLOCKS})$
• b is a block of r	$b \in \text{blcks}(r)$
• first block of r	routeFst(r)
• block after b on r	routeNext({b}, r)
• r1 starts in the middle of r2	startsInTheMiddle(r1, r2)
• R is a set of routes	$R \subseteq \text{routes}(\text{BLOCKS})$
• no route in R ends in the middle of another	routesDontEndInTheMiddle(R)

Subchains

Routes reservations and trains are subchains of routes.

$\{ sub \mid sub \in chains(T) \land subChain(sub, c) \}$

Operators:

- subChain (constructor)
- subChains (constructor)
- frontSubChains (constructor)
- backSubChains (constructor)

Theorems:

- subChainIsChain
- frontSubChainIsSubChain
- backSubChainIsSubChain
- chainTailIsABackSubChain
- chainTail
- ...

Subchains (How To Use It ?)

Routes reservations and trains (routes occupation) are subchains of routes.

$\{ \text{sub} \mid \text{sub} \in \text{chains}(T) \land \text{subChain}(\text{sub}, c) \}$

• r1 is a subchain of r2	subChain(r1, r2)
• set of subchains of r	subChains(r)
• set of “front subchains” of r	frontSubChains(r)
• set of “back subchains” of r	backSubChains(r)

Routes Reservations

A theory for maintaining the set of routes reservations.

$\{ r \mapsto b \mid r \in routes(blocks) \land b \in backSubChains(r) \}$

Operators:

- allRouteRes (constructor)
- validRouteRes (constructor)
- onlyOneResByRoute
- compatibleRoutesOnly
- wellFormedRouteRes
- blocksResForThisRoute
- isReserved
- addToRouteRes
- freeResHeadBlock
- …

Theorems:

- routeReservationIsAFunction
- routeResAreBackSubChains
- addRouteResStillOnlyOne
- addRouteResStillCompatible
- onlyOneResByRouteTrans
- filterResUnion
- …

Routes Reservations (How To Use It ?)

A theory for maintaining the set of routes reservations.

$\{ r \mapsto b \mid r \in routes(blocks) \land b \in backSubChains(r) \}$

• the set of route reservations on the track	res ∈ validRouteRes(BLOCKS, ROUTES)
• only one reservation can be made for each route	onlyOneResByRoute(res)
• a block cannot be reserved twice	compatibleRoutesOnly(res)

Routes Reservations (How To Use It ?)

A theory for maintaining the set of routes reservations.

$\{ r \mapsto b \mid r \in routes(blocks) \land b \in backSubChains(r) \}$

$r = \{b1; b2; \ldots; bn\}$

Events	**Route Reservations**
Initialisation	$\{\}$
Reserve r	$\{ r \mapsto \{ b1; b2; \ldots; bn \}\}$
Enter route; Front moves...	$\{ r \mapsto \{ b1; b2; \ldots; bn \}\}$
Back move	$\{ r \mapsto \{ b2; \ldots; bn \}\}$
Back moves…	$\{ r \mapsto \{\}\}$
Free r	$\{\}$

Your Connection to ICT Research

Reducing the Proof Effort ?

Conclusion

Current Status

Theorems defined = proof factorized.

Manual proof is easier, sometimes even trivial.

But only 40% POs automatically discharged in our theories and models.

Defining theories does not naturally simplify the proof.

The Right Theorems… (future work)

How to define the right theorems ?

Suggestion:

1. define more theorems… a lot of them !
2. generalization of the theorems
3. simplification of the theorems

Proof Strategies? (future work)

Theorems not automatically applied: use strategies ?

How to define them ?

Discussion (1/2)

- Infix predicates ?
  - startsInTheMiddle(r1, r2) => r1 startsInTheMiddle r2

- Type inference ?
  - r ∈ ROUTES
    ¬ isReserved(r, res_routes)

- Local theories ? Using symbols that are local for a given project.
  - ¬ isReserved(r, res_routes)

- Assignment operators ?
  - res_routes ≔ res_routes ∪ completeRes(r)

Discussion (2/2)

- Theory instanciation?
  - ex: Instantiate the Routes theory with the constant BLOCKS

- What about a Rodin plug-in for generating *Domain Specific Platforms* (DSP) ?
  1. Define the DSL using the Theory plug-in
  2. Validate the DSL with signaling engineers
  3. Generate the DSP
  4. Let signaling engineers model their system

Your Connection to ICT Research

Aéropole de Charleroi-Gosselies
Avenue Jean Mermoz 28
B-6041 Charleroi - Belgique

twitter.com/@CETIC 🇫🇷
twitter.com/@CETIC_be 🇬🇧

linkedin.com/company/cetic

info@cetic.be

+32 71 490 700

www.cetic.be

Your Connection to ICT Research

Thanks

Yoann Guyot

Senior Research Engineer

yoann.guyot@cetic.be