Event-B to lambdapi

Jean-Paul Bodeveix, Mamoun Filali, Anne Grieu

INP - IRIT
Université de Toulouse
Équipe ACADIE

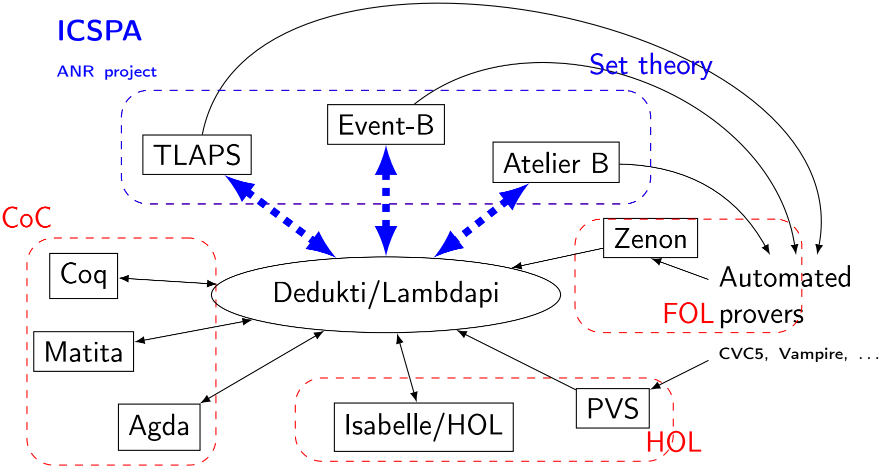
Working Group Meeting
Septembre 2024 Fontainebleau

Outline

ICSPA project

ACADIE

Formal methods - Interoperability



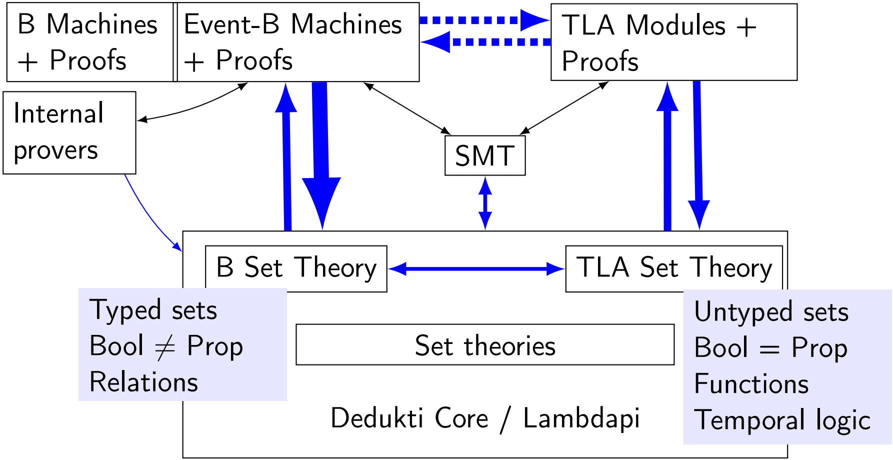
ICSPA Partners

- SAMOVAR
- INRIA Nancy
- INRIA Paris-Saclay
- IRIT
- LIRMM
- CLEARSY

europroofnet.github.io/_pages/WG1/Jun2022/frederic.pdf

ACADIE

Formal methods based on set theories



Mathematical constructs of Event-B

B Mathematical Theory

The mathematical theory of Event-B, First Order Classical Predicate Calculus extended with Set Theory, is defined in several steps :

- Proposition language
- Predicate language
- Typed-set theory
- Arithmetic.

We will show the methodology with the construction of the propositional language and give some details on the typed-set theory.

Proposition Language

Basic constructs

1. $\wedge, \Rightarrow, \neg$

   $\rightarrow$Axiomatic theory

2. Constant $\perp$ + more practical expression of rules.

   Strategy $\rightarrow$Semi-decision algorithm

	Antecedents	Consequent
R1	$H \vdash P$ $H \vdash Q$	$H \vdash P \wedge Q$
R2	$H \vdash P \wedge Q$	$H \vdash P$
R3	$H \vdash P \wedge Q$	$H \vdash Q$
R4	$H, P \vdash Q$	$H \vdash P \Rightarrow Q$
R5	$H \vdash P \Rightarrow Q$	$H, P \vdash Q$
R6	$H, \neg Q \vdash P$ $H, \neg Q \vdash \neg P$	$H \vdash Q$
R7	$H, Q \vdash P$ $H, Q \vdash \neg P$	$H \vdash \neg Q$

ACADIE

Proposition Language

Basic constructs

1. $\wedge, \Rightarrow, \neg$
   →Axiomatic theory
2. Constant $\perp$ + more practical expression of rules.
   Strategy →Semi-decision algorithm

	Antecedents	Consequent
INI	$H \vdash \neg R \Rightarrow \perp$	$H \vdash R$
AXM		$H, P, \neg P \vdash R$
AND1	$H \vdash \neg Q \Rightarrow R$ $H \vdash \neg P \Rightarrow R$	$H \vdash \neg(P \wedge Q) \Rightarrow R$
AND2	$H \vdash P \Rightarrow (Q \Rightarrow R)$	$H \vdash (P \wedge Q) \Rightarrow R$
IMP1	$H \vdash P \Rightarrow (\neg Q \Rightarrow R)$	$H \vdash \neg(P \Rightarrow Q) \Rightarrow R$
IMP2	$H \vdash Q \Rightarrow R$ $H \vdash \neg P \Rightarrow R$	$H \vdash (P \Rightarrow Q) \Rightarrow R$
NEG	$H \vdash P \Rightarrow R$	$H \vdash \neg\neg P \Rightarrow R$
DED	$H, P \vdash R$	$H \vdash P \Rightarrow R$

ACADIE

Proposition Language

Basic constructs

1. $\wedge, \Rightarrow, \neg$
   →Axiomatic theory
2. Constant $\bot$ + more practical expression of rules.
   Strategy →Semi-decision algorithm

	Antecedents	Consequent
INI	$H \vdash \neg R \Rightarrow \bot$	$H \vdash R$
AXM		$H, P, \neg P \vdash R$
AND1	$H \vdash \neg Q \Rightarrow R$ $H \vdash \neg P \Rightarrow R$	$H \vdash \neg(P \wedge Q) \Rightarrow R$
AND2	$H \vdash P \Rightarrow (Q \Rightarrow R)$	$H \vdash (P \wedge Q) \Rightarrow R$
IMP1	$H \vdash P \Rightarrow (\neg Q \Rightarrow R)$	$H \vdash \neg(P \Rightarrow Q) \Rightarrow R$
IMP2	$H \vdash Q \Rightarrow R$ $H \vdash \neg P \Rightarrow R$	$H \vdash (P \Rightarrow Q) \Rightarrow R$
NEG	$H \vdash P \Rightarrow R$	$H \vdash \neg\neg P \Rightarrow R$
DED	$H, P \vdash R$	$H \vdash P \Rightarrow R$

Order of rules :	AXM, IMP1, IMP2, AND1, AND2, NEG
Proof procedure :	INI ; (RULES* ;DED)*

ACADIE

Propositional calculus

Derived constructs

$\vee$, $\Leftrightarrow$ and $\top$, defined as rewriting of basic constructs.

Predicate	Definition
$\top$	$\neg\bot$
$P \vee Q$	$\neg P \Rightarrow Q$
$P \Leftrightarrow Q$	$(P \Rightarrow Q) \wedge (Q \Rightarrow P)$

Derived rules

Proved with previous rules.

	Antecedents	Consequent
OR1	$H \vdash \neg P \Rightarrow (\neg Q \Rightarrow R)$	$H \vdash \neg(P \vee Q) \Rightarrow R$
OR2	$H \vdash Q \Rightarrow R$ $H \vdash P \Rightarrow R$	$H \vdash (P \vee Q) \Rightarrow R$
EQV1	$H \vdash P \Rightarrow (\neg Q \Rightarrow R)$ $H \vdash \neg P \Rightarrow (Q \Rightarrow R)$	$H \vdash (\neg P \Leftrightarrow Q) \Rightarrow R$
EQV2	$H \vdash P \Rightarrow (Q \Rightarrow R)$ $H \vdash \neg P \Rightarrow (\neg Q \Rightarrow R)$	$H \vdash (P \Leftrightarrow Q) \Rightarrow R$

ACADIE

Derived rules

With these rules, we can prove some classical results : commutativity, associativity, distributivity, law of excluded middle, idempotence, absorption, de Morgan laws, contraposition, double negation, transitivity, monotony, equivalence, like :

For P and Q predicates :

$P \lor \neg P$	Law of excluded middle
$P \Leftrightarrow \neg\neg P$	Double negation
$\neg(P \land Q) \Leftrightarrow \neg P \lor \neg Q$ $\neg(P \lor Q) \Leftrightarrow \neg P \land \neg Q$	de Morgan laws
$P \lor P \Leftrightarrow P$ $P \land P \Leftrightarrow P$	Idempotence
$(P \lor Q) \land P \Leftrightarrow P$ $(P \land Q) \lor P \Leftrightarrow P$	Absorption

ACADIE

Equivalence rewriting

For P, Q, R predicates, such as $P \Leftrightarrow Q$:

$$\boxed{\begin{array}{l} (P \wedge R) \Rrightarrow (Q \wedge R) \\ (P \vee R) \Rrightarrow (Q \vee R) \\ (R \Rightarrow P) \Rrightarrow (R \Rightarrow Q) \\ (Q \Rightarrow R) \Rrightarrow (P \Rightarrow R) \\ \neg P \Rrightarrow \neg Q \end{array}}$$

« *The last series of properties shows that when two predicates have been proved to be equivalent then replacing one by the other in any predicate preserves equivalence (this can be proved by induction on the syntactic structure of the predicate notation). In other words, once proved, an equivalence assertion can be used operationally as if it were a rewriting rule.* »[1]

1. J.-R. Abrial. The B-Book, assigning programs to meaning. Cambridge University Press, 1996.

ACADIE

First order predicate calculus

Predicates language

Following the same methodology, we define :

- Variables, expressions, substitutions,
- Basic predicate universal quantifier $\forall$,
- Derived predicate universal quantifier $\exists$
- Definition of equality.

First order predicate calculus

$$
\begin{array}{lll}
\mathit{predicate} & := & \bot \\
 & & \top \\
 & & \mathit{predicate} \wedge \mathit{predicate} \\
 & & \mathit{predicate} \vee \mathit{predicate} \\
 & & \mathit{predicate} \Rightarrow \mathit{predicate} \\
 & & \mathit{predicate} \Leftrightarrow \mathit{predicate} \\
 & & \forall \mathit{varList} . \mathit{predicate} \\
 & & \exists \mathit{varList} . \mathit{predicate} \\
 & & [\mathit{varList} := \mathit{expList}]\mathit{predicate} \\
 & & \mathit{expression} = \mathit{expression} \\
 & & \\
\mathit{expression} & := & \mathit{variable} \\
 & & [\mathit{varList} := \mathit{expList}]\mathit{expression} \\
 & & \mathit{expression} \rightarrow \mathit{expression} \\
 & & \\
\mathit{variable} & := & \mathit{identifier}
\end{array}
$$

Event-B Set theory

We extend the theory with the syntactic category *set* and the membership predicate : $E \in s$, E expression and s set.

Some rules :

$$
\begin{array}{lll}
E \in Pow(S) & \Rrightarrow & \forall x . x \in E \Rightarrow x \subset S \\
S \subset T & \Rrightarrow & S \in \mathbb{P}(T) \\
E \in S \cap T & \Rrightarrow & E \in S \wedge E \in T
\end{array}
$$

This completes the syntax :

$$
\begin{array}{lll}
predicate & := & \ldots \\
 & & expression \in expression \\
expression & := & \ldots \\
 & & set \\
\ldots & & \\
set & := & set \times set \\
 & & \mathbb{P}(set) \\
 & & \{varList.predicate | expression\} \\
 & & variable
\end{array}
$$

ACADIE IRIT

Event-B Type theory

Any predicate will be type-checked before being proved. A type denotes the set of values an expression can take.

Event-B types :

$T ::=$	$\mathrm{BOOL} \mid \mathbb{Z}$	built-in boolean and integer types
	$\mid S$	carrier set S provided by user
	$\mid \mathbb{P}\ T$	power set of a type
	$\mid T \times T$	cartesian product of types

ACADIE

Embedding Event-B in lambdapi

Lambdapi

« Lambdapi is an interactive proof system featuring ***dependent types*** *like in Martin-Lőf's type theory, but allowing to define objects and types using* ***oriented equations****, aka* ***rewriting rules****, and reason modulo those equations. »* [2]

$\lambda\Pi$ terms

$t\ , t' ::=$	V	variable
	$\mid \mathtt{TYPE}$	sort for types
	$\mid \Pi\ (V : t),\ t'$	dependent product type
	$\mid \lambda\ (V : t),\ t'$	abstraction
	$\mid t\ t'$	application
	$\mid t \to t'$	abbreviation for $\Pi\ (V : t),\ t'$ when $V \notin t'$

Rules

$r \quad ::= \quad t \hookrightarrow t'$ reasoning modulo rewriting rules

2. `https://lambdapi.readthedocs.io/en/latest/about.html`

First order logic [3]

> *« Lambdapi is a logical framework, that is, it does not come with a pre-defined logic. Instead, one has to start defining its own logic. »*

Propositional logic

```
constant symbol Prop : TYPE;
// Associates a type of a proof to a proposition
injective symbol π : Prop → TYPE;
```

Types of datatypes

```
constant symbol Set : TYPE;
// Associates a type to a datatype
injective symbol τ : Set → TYPE;
```

3. Standard library : `https://github.com/Deducteam/lambdapi-stdlib`

First order logic

« Lambdapi is a logical framework, that is, it does not come with a pre-defined logic. Instead, one has to start defining its own logic. »

Conjunction

```
constant symbol ∧ : Prop → Prop → Prop;
notation ∧ infix left 7;
constant symbol ∧ᵢ p q: π p → π q → π (p ∧ q);
symbol ∧ₑ₁ p q : π (p ∧ q) → π p;
symbol ∧ₑ₂ p q : π (p ∧ q) → π q;
```

Implication (Coq style)

```
constant symbol ⇒ : Prop → Prop → Prop;
notation ⇒ infix right 5;
rule π ($p ⇒ $q) ↪ π $p → π $q;
```

Related sequents for conjunction

$$\frac{\Gamma \vdash p \quad \Gamma \vdash q}{\Gamma \vdash p \wedge q}\ (\wedge_i)$$

$$\frac{\Gamma \vdash p \wedge q}{\Gamma \vdash p}\ (\wedge_{e1})$$

$$\frac{\Gamma \vdash p \wedge q}{\Gamma \vdash q}\ (\wedge_{e2})$$

ACADIE

Event-B set theory

Event-B types :

$S ::=$	$\sigma\mathbb{P}\ S$	power set
	$\mid S\ \sigma\times\ S$	cartesian product
	$\mid \sigma\mathrm{BOOL} \mid \sigma\mathbb{Z}$	built-in boolean and integer types
	$\mid\ \sigma S$	for each user declared set S

In lambdapi :

```
injective symbol σℙ: Set → Set; // power set
injective symbol σ×: Set → Set → Set; // cartesian product
notation σ× infix left 24;
constant symbol σBOOL: Set; // pre-defined boolean set
constant symbol σℤ: Set;//pre-defined integer set
constant symbol σS: Set; // user declared set S
```

Set operators

Classical set operators of Event-B derive from membership operator :

```
symbol ∈ [T:Set] : τ T → τ (σℙ T) → Prop;
rule _ ∈ ∅ ↪ ⊥;
rule $x ∈ $s1 ∩ $s2 ↪ $x ∈ $s1 ∧ $x ∈ $s2;
rule $e ∈ ℙ $S ↪ $e ⊆ $S;
rule $s1 ⊆ $s2 ↪ `∀ x, x ∈ $s1 ⇒ x ∈ $s2;
```

Generic maximal set BIG

```
constant symbol BIG [T:Set]: τ (σℙ T);// set of all elements of type τ T
rule $x ∈ BIG ↪ ⊤;// BIG is maximal: contains all elements of type τ T
rule ℙ BIG ↪ BIG;// power set of BIG is a maximal set
rule BIG × BIG ↪ BIG;//cartesian product of two maximal sets is maximal
```

Critical pairs

- $P \land \top \Rightarrow P$
- $P \lor \top \Rightarrow \top$
- In Rodin, the rule type rewrites do some automatic rewriting : $x \in S$ if S is maximal, then $x \in S \hookrightarrow \top$. The choice of BIG and its rules is a solution to express some of these rules, but this is also a source of conflicts.

Example :

- $\boxed{x \in \mathbb{P}(BIG)}$
- $\boxed{x \in \mathbb{P}(BIG)}$

ACADIE

Critical pairs

- $P \land \top \Rightarrow P$
- $P \lor \top \Rightarrow \top$
- In Rodin, the rule type rewrites do some automatic rewriting : $x \in S$ if S is maximal, then $x \in S \hookrightarrow \top$. The choice of BIG and its rules is a solution to express some of these rules, but this is also a source of conflicts.

Example :

- $\boxed{x \in \mathbb{P}(BIG)} \hookrightarrow x \subseteq BIG$
- $\boxed{x \in \mathbb{P}(BIG)}$

ACADIE

Critical pairs

- $P \wedge \top \Rightarrow P$
- $P \vee \top \Rightarrow \top$
- In Rodin, the rule type rewrites do some automatic rewriting : $x \in S$ if S is maximal, then $x \in S \hookrightarrow \top$. The choice of BIG and its rules is a solution to express some of these rules, but this is also a source of conflicts.

Example :

- $\boxed{x \in \mathbb{P}(BIG)} \hookrightarrow x \subseteq BIG \hookrightarrow \forall u.u \in x \Rightarrow u \in BIG$
- $\boxed{x \in \mathbb{P}(BIG)}$

ACADIE

Critical pairs

- $P \wedge \top \Rightarrow P$
- $P \vee \top \Rightarrow \top$
- In Rodin, the rule type rewrites do some automatic rewriting : $x \in S$ if S is maximal, then $x \in S \hookrightarrow \top$. The choice of BIG and its rules is a solution to express some of these rules, but this is also a source of conflicts.

Example :

- $\boxed{x \in \mathbb{P}(BIG)} \hookrightarrow x \subseteq BIG \hookrightarrow \forall u.u \in x \Rightarrow u \in BIG$

  As $u \in BIG \hookrightarrow \top$, we have $\forall u.u \in x \Rightarrow \top$
- $\boxed{x \in \mathbb{P}(BIG)}$

ACADIE

Critical pairs

- $P \wedge \top \Rightarrow P$
- $P \vee \top \Rightarrow \top$
- In Rodin, the rule type rewrites do some automatic rewriting : $x \in S$ if S is maximal, then $x \in S \hookrightarrow \top$. The choice of BIG and its rules is a solution to express some of these rules, but this is also a source of conflicts.

Example :

- $\boxed{x \in \mathbb{P}(BIG)} \hookrightarrow x \subseteq BIG \hookrightarrow \forall u.u \in x \Rightarrow u \in BIG \hookrightarrow \forall u.u \in x \Rightarrow \top$

  As $u \in BIG \hookrightarrow \top$, we have $\forall u.u \in x \Rightarrow \top$
- $\boxed{x \in \mathbb{P}(BIG)}$

Critical pairs

- $P \wedge \top \Rightarrow P$
- $P \vee \top \Rightarrow \top$
- In Rodin, the rule type rewrites do some automatic rewriting : $x \in S$ if S is maximal, then $x \in S \hookrightarrow \top$. The choice of BIG and its rules is a solution to express some of these rules, but this is also a source of conflicts.

Example :

- $\boxed{x \in \mathbb{P}(BIG)} \hookrightarrow x \subseteq BIG \hookrightarrow \forall u.u \in x \Rightarrow u \in BIG \hookrightarrow \forall u.u \in x \Rightarrow \top$

  As $u \in BIG \hookrightarrow \top$, we have $\forall u.u \in x \Rightarrow \top$
- $\boxed{x \in \mathbb{P}(BIG)} \hookrightarrow x \in BIG$

Critical pairs

- $P \wedge \top \Rightarrow P$
- $P \vee \top \Rightarrow \top$
- In Rodin, the rule type rewrites do some automatic rewriting : $x \in S$ if S is maximal, then $x \in S \hookrightarrow \top$. The choice of BIG and its rules is a solution to express some of these rules, but this is also a source of conflicts.

Example :

- $\boxed{x \in \mathbb{P}(BIG)} \hookrightarrow x \subseteq BIG \hookrightarrow \forall u.u \in x \Rightarrow u \in BIG \hookrightarrow \forall u.u \in x \Rightarrow \top$

  As $u \in BIG \hookrightarrow \top$, we have $\forall u.u \in x \Rightarrow \top$
- $\boxed{x \in \mathbb{P}(BIG)} \hookrightarrow x \in BIG \hookrightarrow \top$

ACADIE

Relational operators

```
symbol rel (T1 T2: Set) ≔ τ (σℙ (T1 σ× T2));
injective symbol ↦ [T1:Set] [T2:Set] (x:τ T1) (y:τ T2) : τ (T1 σ× T2);
symbol ↔ [T1:Set] [T2:Set] (A:τ (σℙ T1)) (B: τ (σℙ T2)):
    τ (σℙ (σℙ (T1 σ× T2))) ≔ ℙ (A × B); notation ↔ infix 11;

constant symbol dom [T1:Set] [T2:Set] : rel T1 T2 → τ (σℙ T1);
notation dom prefix 30;
rule $x ∈ dom($r) ↪ `∃ y, $x ↦ y ∈ $r;
```

Proofs in Rodin

Rodin : Rigourous Open Development Environment for Complex Systems

The Rodin Platform is an Eclipse-based IDE for Event-B that provides effective support for refinement and mathematical proof. The platform is open source, contributes to the Eclipse framework and is further extendable with plugins.[4]

The mathematical proofs are shown as **proof trees**.
Statements are declared in Contexts and the proof trees are built automatically and/or guided by the user.

We will present these notions using the example of Cantor's Theorem.

4. https ://www.event-b.org/platform.html

Context - Cantor's theorem

In Rodin

cantor ×

```
context cantor

sets S

axioms
  theorem @th ¬(∃f·f∈S↠ℙ(S))
end
```

In Lambdapi

```
constant symbol σS: Set;
symbol S: τ (σℙ σS) ≔ BIG;
```

S is the embedding in lambdapi of the set S defined in the Rodin-context.

Cantor's theorem

In Rodin

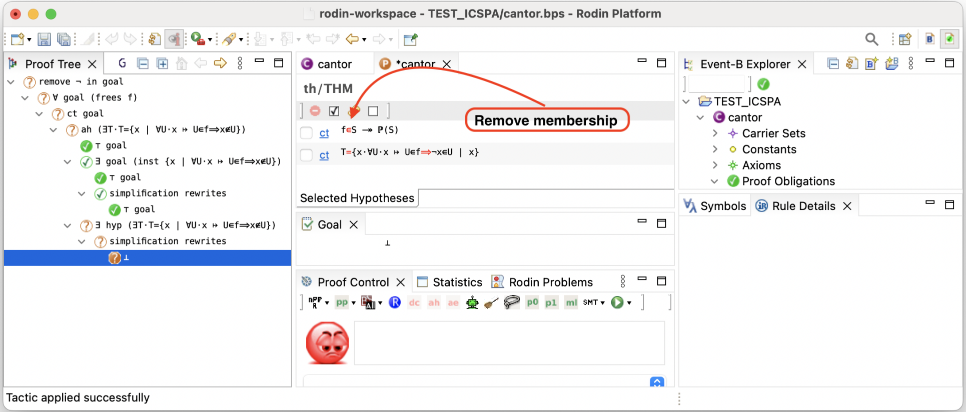
```
cantor ×
context cantor

sets S

axioms
  theorem @th ¬(∃f·f∈S↠ℙ(S))
end
```

In Lambdapi

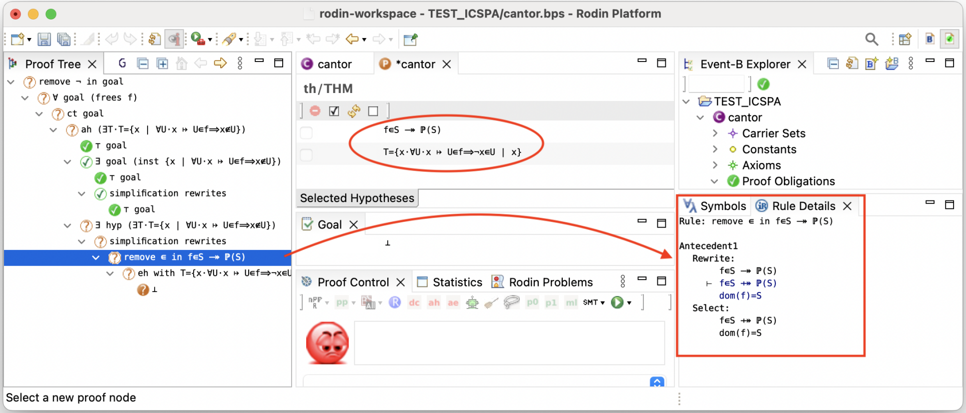
f is a total surjection from S to $\mathbb{P}((S))$, that is $\boxed{f \in \mathbb{P}(S \times \mathbb{P}((S)))}$ and a total function and a surjection.

```
constant symbol σS: Set;
symbol S: τ (σℙ σS)≔ BIG;
symbol th:π(¬((`∃ (f: τ(σℙ (σS σ× (σℙ σS)))), f ∈ (S↠(ℙ S))))):=
... end;
```

Building Cantor's proof with Rodin



Building Cantor's proof with Rodin



Proof tree

Proof Tree

```
remove ¬ in goal
  ∀ goal (frees f)
    ct goal
      remove ∈ in f∈S ↠ ℙ(S)
        remove ∈ in f∈S ⤀ ℙ(S)
          remove ∈ in f∈S ⇸ ℙ(S)
            type rewrites
              rewrites set equality in hyp (ran(f)=ℙ(S))
                type rewrites
                  remove ⊆ in ℙ(S)⊆ran(f)
                    type rewrites
                      simplification rewrites
                        ∀ hyp (inst {x | ∀U·x ↦ U∈f⇒¬x∈U})
                          ⊤ goal
                          remove ∈ in {x | ∀U·x ↦ U∈f⇒¬x∈U}∈ran(f)
                            ∃ hyp (∃x·x ↦ {x | ∀U·x ↦ U∈f⇒¬x∈U}∈f)
                              dc (x∈{x | ∀U·x ↦ U∈f⇒¬x∈U})
                                ⊤ goal
```

Translation of the Rules from Event-B to lambdapi

Rodin proof rule	Lambdapi tactic
$\dfrac{}{\Gamma, h : p \vdash p}$ (hyp)	refine h
$\dfrac{h : p, \Gamma \vdash q}{\Gamma \vdash p \Rightarrow q}$ ($\Rightarrow$ goal)	assume h
$\dfrac{\Gamma, h : x_i \in T_i \vdash p}{\Gamma \vdash \forall x_1, \ldots, x_n \cdot p}$ ($\forall$goal)	assume $x_1 \ldots x_n$
$\dfrac{\Gamma \vdash p_1 \ldots \quad \Gamma \vdash p_n}{\Gamma \vdash p_1 \wedge \cdots \wedge P_n}$ ($\wedge$goal)	apply $\wedge_i$ p_1 (apply $\wedge_i$ p_2 $(\ldots)$)

Rules from Event-B to lambdapi

Rules defined as theorems

```
symbol Or2ImpGoal [P Q: Prop] :
      π (((¬ P) ⇒ Q) ⇒ (P ∨ Q)) ≔
begin
  assume P Q h;
  apply (λ h1 h2, ∨ₑ P (¬ P)
                   (P ∨ Q) h1 h2 (classic P))
    {assume hp; apply (∨ᵢ₁ _ _ hp)}
    {assume hnp; apply (∨ᵢ₂ _ _ (h hnp))}
end;
```

Rodin proof rule	Lambdapi tactic
$\dfrac{\Gamma \vdash \neg p \Rightarrow q}{\Gamma \vdash p \vee q}$	apply Or2ImpGoal

ACADIE

Conclusion

Open topics

Problems

- lot's of automatic rewriting rules in Event-B/Rodin
- Prop and Bool are different in Rodin, $=$ and $\Leftrightarrow$ can't be identify
- some operators, like $\wedge$ or $\vee$ are n-ary, difficult to express in lambdapi

Open topics

Problems

- lot's of automatic rewriting rules in Event-B/Rodin
- Prop and Bool are different in Rodin, $=$ and $\Leftrightarrow$ can't be identify
- some operators, like $\wedge$ or $\vee$ are n-ary, difficult to express in lambdapi

Investigations

- Use Lambdapi rewriting rules , but too much rewriting rules leads to critical pairs (eg. BIG, former neg)
- Theorems, preprocessing and tactics (repeat, setoid rewrite,...) with synthesis of lambdapi proof term in Java.
- Integration of Coq[5] setoid rewrite in Lambdapi ?

5. https://coq.inria.fr/doc/V8.10.2/refman/addendum/generalized-rewriting.html

Generalized rewriting

Rewriting rules

- equal by equal rewriting : $((a = b) ==> f(a)) \Rrightarrow f(b)$
- equivalent by equivalent rewriting : $((P \Leftrightarrow Q) ==> f(P)) \Rrightarrow f(Q)$
- $P \wedge \top \Leftrightarrow P$
- $P \wedge \ldots Q \wedge P \wedge R \cdots \Leftrightarrow P \wedge \ldots Q \wedge R \ldots \quad \ldots$

In Lambdapi

Tactic `rewrite` [6] allows rewriting only for equality, not for equivalence.

6. https://lambdapi.readthedocs.io/en/latest/tactics.html
https://inria.hal.science/inria-00258384

Dedukti/Lambdapi is a logical framework based on $\lambda\Pi$-calculus modulo rewriting system, meant to allow interoperability between formal method systems.

We presented some steps of our translation of the first order logic and set theory of Event-B and its deduction rules in Lambdapi to translate a statement and a guided proof from Rodin in Lambdapi.

A first usecase has been a guided proof of Cantor's theorem in Event-B.

Ongoing work

- Continue translation of deduction rules
- Deals with generalized rewriting
- Deals with internal and external automated provers
- Translate machines and events

Thanks for your attention