CERTIFICATES FOR THE QUALIFICATION OF THE MODEL CHECKER KIND 2

Alain Mebsout

April 26, 2015

University of Iowa

DO-178C, DO-330, DO-333

Traditional process (ex. Alt-Ergo)

Lightweight: through certification (ex. Kind 2)

QUALIFICATION OF ALT-ERGO

SMT Solver

Université Paris-Sud, INRIA, CNRS, OCamlPro

- Project leaders: Sylvain Conchon, Évelyne Contejean
- Current developer: Mohamed Iguernlala
- Contributors: Stéphane Lescuyer, Alain Mebsout

(almost) Purely functional, written in OCaml

Small: $\sim$ 10kloc

Implementation close to formal description

Correctness proof of core in Coq

ALT-ERGO: FEATURES

- First-order polymorphic logic
- Shostak like combination based on CC(X) (and AC(X))

Theories:

- equality over uninterpreted symbols (EUF)
- linear arithmetic (on $\mathbb{Q}$ LRA, and $\mathbb{Z}$ LIA)
- non-linear arithmetic (on $\mathbb{Q}$ NRA, and $\mathbb{Z}$ NIA)
- Records (and pairs)
- Bitvectors (concat and extract)
- Associative/Commutative symbols (AC)
- Functional arrays
- Enumerated datatypes
- Quantifiers ($\forall$, $\exists$)
- ...

Context

- Airbus, for use on A350
- Verification of C code (pre-flight inspection) with Caveat (CEA)

What was qualified

- Version 0.94
- SAT (propositional logic)
- Polymorphic types
- Equality modulo AC
- Quantifiers (with triggers mechanism modulo equality)
- Arithmetic (real and integer)

ALT-ERGO: QUALIFICATION PACKAGE (INRIA)

Formal description of qualified modules (65 pp. French text + inference rules)

Each section gives precise functional tool requirements (TR)

ALT-ERGO: QUA... (INRIA)

Formal ... ench text +

inferenc...

Each se... ents (TR)

Normal-Run

$$\frac{\mathsf{U} \mid \mathsf{R} \mid \mathsf{UF} \mid \mathsf{L}_1 \longrightarrow^* \mathsf{U}' \mid \mathsf{R}' \mid \mathsf{UF}' \mid \emptyset \qquad \mathsf{U}_f \mid \mathsf{R}_f \mid \mathsf{UF}_f \mid \mathsf{L}_1 \longrightarrow^* \mathsf{U}'_f \mid \mathsf{R}'_f \mid \mathsf{UF}_f \mid \emptyset}{(\mathsf{U} \mid \mathsf{R} \mid \mathsf{UF}) \mid\mid (\mathsf{U}_f \mid \mathsf{R}_f \mid \mathsf{UF}_f) \mid\mid (\mathsf{L}_1;\mathsf{L}_2) \longrightarrow (\mathsf{U}' \mid \mathsf{R}' \mid \mathsf{UF}') \mid\mid (\mathsf{U}'_f \mid \mathsf{R}'_f \mid \mathsf{UF}'_f) \mid\mid \mathsf{L}_2}$$

Case-Split

$$\frac{\mathsf{C} = \mathtt{case\_split}(\mathsf{R}_f) \qquad \mathsf{U}_f \mid \mathsf{R}_f \mid \mathsf{UF}_f \mid \mathsf{C} \longrightarrow^* \mathsf{U}'_f \mid \mathsf{R}'_f \mid \mathsf{UF}'_f \mid \emptyset}{(\mathsf{U} \mid \mathsf{R} \mid \mathsf{UF}) \mid\mid (\mathsf{U}_f \mid \mathsf{R}_f \mid \mathsf{UF}_f) \mid\mid \mathsf{L} \longrightarrow (\mathsf{U} \mid \mathsf{R} \mid \mathsf{UF}) \mid\mid (\mathsf{U}'_f \mid \mathsf{R}'_f \mid \mathsf{UF}'_f) \mid\mid \mathsf{L}}$$

Conflict

$$\frac{\mathsf{U} \mid \mathsf{R} \mid \mathsf{UF} \mid \mathsf{L} \longrightarrow^* \bot[d]}{(\mathsf{U} \mid \mathsf{R} \mid \mathsf{UF}) \mid\mid (\mathsf{U}_f \mid \mathsf{R}_f \mid \mathsf{UF}_f) \mid\mid \mathsf{L} \longrightarrow \bot[d]}$$

Case-Split-Erase-Choices

$$\frac{\mathsf{U} \mid \mathsf{R} \mid \mathsf{UF} \mid \mathsf{L}_1 \longrightarrow^* \mathsf{U}' \mid \mathsf{R}' \mid \mathsf{UF}' \mid \emptyset \qquad \mathsf{U}_f \mid \mathsf{R}_f \mid \mathsf{UF}_f \mid \mathsf{L}_1 \longrightarrow^* \bot[d]}{(\mathsf{U} \mid \mathsf{R} \mid \mathsf{UF}) \mid\mid (\mathsf{U}_f \mid \mathsf{R}_f \mid \mathsf{UF}_f) \mid\mid (\mathsf{L}_1;\mathsf{L}_2) \longrightarrow (\mathsf{U}' \mid \mathsf{R}' \mid \mathsf{UF}') \mid\mid (\mathsf{U}' \mid \mathsf{R}' \mid \mathsf{UF}') \mid\mid \mathsf{L}_2}$$

Case-Split-Progress

$$\frac{\begin{array}{c}\mathsf{C} = \mathtt{case\_split}(\mathsf{R}_f) \\ \mathsf{U}_f \mid \mathsf{R}_f \mid \mathsf{UF}_f \mid \mathsf{C} \longrightarrow^* \bot[d] \qquad \mathsf{U}_f \mid \mathsf{R}_f \mid \mathsf{UF}_f \mid \neg\mathsf{C}[d] \longrightarrow^* \mathsf{U}'_f \mid \mathsf{R}'_f \mid \mathsf{UF}_f \mid \emptyset\end{array}}{(\mathsf{U} \mid \mathsf{R} \mid \mathsf{UF}) \mid\mid (\mathsf{U}_f \mid \mathsf{R}_f \mid \mathsf{UF}_f) \mid\mid \mathsf{L} \longrightarrow (\mathsf{U} \mid \mathsf{R} \mid \mathsf{UF}) \mid\mid (\mathsf{U}'_f \mid \mathsf{R}'_f \mid \mathsf{UF}'_f) \mid\mid \mathsf{L}}$$

Case-Split-Conflict

$$\frac{\begin{array}{c}\mathsf{C} = \mathtt{case\_split}(\mathsf{R}_f) \\ \mathsf{U}_f \mid \mathsf{R}_f \mid \mathsf{UF}_f \mid \mathsf{C} \longrightarrow^* \bot[d] \qquad \mathsf{U}_f \mid \mathsf{R}_f \mid \mathsf{UF}_f \mid \neg\mathsf{C}[d] \longrightarrow^* \bot[d']\end{array}}{(\mathsf{U} \mid \mathsf{R} \mid \mathsf{UF}) \mid\mid (\mathsf{U}_f \mid \mathsf{R}_f \mid \mathsf{UF}_f) \mid\mid \mathsf{L} \longrightarrow \bot[d']}$$

Formal description of qualified modules (65 pp. French text + inference rules)

Each section gives precise functional tool requirements (TR)

Version of Alt-Ergo with traces for TRs

Disable all unqualified features

$\sim$ 500 tests for the TRs (correctness and coverage)

Forr … text +

infe

Each … s (TR)

Vers

Disa

∼ 5(

Commande exécutée

```
alt-ergo -restricted -rules cc testfile-case_split001.mlw
```

Référence de l'exigence fonctionnelle

— TR-CCX-Builtin-Conflict
— TR-CCX-Builtin
— TR-CCX-CS-Case-Split
— TR-CCX-Distinct
— TR-UFX-Add
— TR-CCX-AddTerm
— TR-UFX-Find

Objectif(s) du test

La fonctionnalité à vérifier est que l'algorithme CC(X) implémente bien ses spécifications et que Alt-Ergo est capable de prouver des buts demandant un raisonnement par analyse par cas.

Description du test

```
goal g1 : forall x,y,z:int.
  -z <= 0 ->
   3 * y - 8*x - 6 <= 0 ->
  -y + 12*x +3 <= 0 ->
  y*y*y <= 1
```

Non trivial process

Few (2-3) man months

Hardest part = coming up with inference rules and relevant TRs that cover all of the tool

Has to be redone for future versions

Allowed to completely review (and sometimes improve) code

KIND 2

KIND 2

University of Iowa:

- Project leader: Cesare Tinelli
- Developers: Adrien Champion, Alain Mebsout, Christoph Sticksel

SMT-based model checker

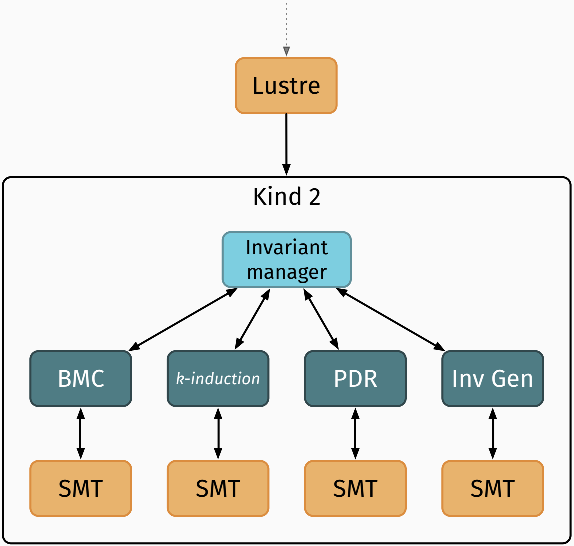
Multi-engine

Parallel

Reactive systems (Lustre)

Modular / compositional reasoning

KIND 2 SYSTEM OVERVIEW



KIND 2 INPUT: LUSTRE (EXAMPLE)

```
node greycounter (reset: bool) returns (out: bool);
var a, b: bool;
let
  a = false -> (not reset and not pre b);
  b = false -> (not reset and pre a);
  out = a and b;
tel


node intcounter (reset: bool; const max: int) returns (out: bool);
var t: int;
let
  t = 0 -> if reset or pre t = max then 0 else pre t + 1;
  out = t = 2;
tel


node top (reset: bool) returns (OK: bool);
var b, d: bool;
let
  b = greycounter(reset);
  d = intcounter(reset, 3);
  OK = b = d;
  --%PROPERTY OK;
tel
```

kind 2 certification

FMTQ

NASA funded project

Qualification of typical formal verification tools

- Theorem provers
- Model checkers
- Abstract interpreters

Application to the model checker Kind 2

- Qualification artifacts
- Investigate alternative approach: proof certificates

QUALIFICATION BY MEANS OF PROOF CERTIFICATES

Proof certificates increase the trust in the results produced by the model checker

- Focus of trust shifted from the model checker to the certificate checker

Can be used as a supplement for more traditional qualification approaches

Partially addresses objectives 6.1.3.1.i of DO-330 and others (?)

Proof ce … uced by the mod

- Focus … ficate checke

Can be u … ification approac

Partially … ners (?)

Table T-3 Verification of Outputs of Tool Requirements Processes

	Objective		Activity	Applicability by TQL					Output		Control Category by TQL				
	Description	Ref.	Ref.	1	2	3	4	5	Description	Ref.	1	2	3	4	5
1	Tool Requirements comply with Tool Operational Requirements.	6.1.3.1.a	6.1.3.1	●	●	○	○		Tool Verification Results	10.2.6	②	②	②	②	
2	Tool Requirements are accurate and consistent.	6.1.3.1.b	6.1.3.1	●	●	○	○		Tool Verification Results	10.2.6	②	②	②	②	
3	Requirements for compatibility with the tool operational environment are defined.	6.1.3.1.c	6.1.3.1	●	●	○	○		Tool Verification Results	10.2.6	②	②	②	②	
4	Tool Requirements define the behavior of the tool in response to error conditions.	6.1.3.1.d	6.1.3.1	●	●	○	○		Tool Verification Results	10.2.6	②	②	②	②	
5	Tool Requirements define user instructions and error messages.	6.1.3.1.e	6.1.3.1	●	●	○	○		Tool Verification Results	10.2.6	②	②	②	②	
6	Tool Requirements are verifiable.	6.1.3.1.f	6.1.3.1	○	○	○	○		Tool Verification Results	10.2.6	②	②	②	②	
7	Tool Requirements conform to Tool Requirements Standards.	6.1.3.1.g	6.1.3.1	○	○	○			Tool Verification Results	10.2.6	②	②	②		
8	Tool Requirements are traceable to Tool Operational Requirements.	6.1.3.1.h	6.1.3.1	○	○	○	○		Tool Verification Results	10.2.6	②	②	②	②	
9	Algorithms are accurate.	6.1.3.1.i	6.1.3.1	●	●	○	○		Tool Verification Results	10.2.6	②	②	②	②	

TRUSTING A MODEL CHECKER

1. **Trusted Logic** — The logics used by the model checker are trusted.
2. **Valid Model** — The model and properties accurately depict the system under scrutiny in the semantic given by the logic of the model checker.
3. **Correct Translation** — The input system is correctly translated to its internal representation.
4. **Correct Algorithms** — The model checking algorithms are sound for the models and properties expressible in the supported logic.
5. **Correct Implementation** — The model checking algorithms are correctly implemented.
6. **Trusted Components and Libraries** — If the model checker makes use of external libraries, their implementation is trusted, and if the model checker uses external tools, they are trusted to be correct.
7. **Correct Compilation** — The model checker and its components are correctly compiled to executable machine code.
8. **Correct Execution** — The machine correctly runs the executable.
9. **Trusted IO** — The parsing and output of the model checker are trusted and interpreted correctly.

TRUSTING A MODEL CHECKER

1.	**Trusted Logic**	The logics used by the model checker are trusted.
2.	**Valid Model**	The model and properties accurately depict the system under scrutiny in the semantic given by the logic of the model checker.
3.	**Correct Translation**	The input system is correctly translated to its internal representation.
4.	**Correct Algorithms**	The model checking algorithms are sound for the models and properties expressible in the supported logic.
5.	**Correct Implementation**	The model checking algorithms are correctly implemented.
6.	**Trusted Components and Libraries**	If the model checker makes use of external libraries, their implementation is trusted, and if the model checker uses external tools, they are trusted to be correct.
7.	**Correct Compilation**	The model checker and its components are correctly compiled to executable machine code.
8.	**Correct Execution**	The machine correctly runs the executable.
9.	**Trusted IO**	The parsing and output of the model checker are trusted and interpreted correctly.

PROVED CORRECT VS CERTIFICATE PRODUCING

Model checkers answers: yes / no

- **no** $\longrightarrow$ counterexample
- **yes** $\longrightarrow$?

PROVED CORRECT VS CERTIFICATE PRODUCING

Model checkers answers: yes / no

- **no** $\longrightarrow$ counterexample
- **yes** $\longrightarrow$?

Prove once and for all that

$$\forall \mathcal{S}, P.\ \ MC(\mathcal{S}, P) = \textbf{yes} \implies \mathcal{S} \models P$$

PROVED CORRECT VS CERTIFICATE PRODUCING

Model checkers answers: yes / no

- **no** $\longrightarrow$ counterexample
- **yes** $\longrightarrow$?

Prove once and for all that

$$\forall \mathcal{S}, P.\ \ MC(\mathcal{S}, P) = \textbf{yes} \implies \mathcal{S} \models P$$

A formal proof of correctness of a model checker is a hard task

- use of large external lib/tools like SMT solvers
- complex and heavily optimized
- parallel architecture

Model checkers answers: yes / no

- **no** $\longrightarrow$ counterexample
- **yes** $\longrightarrow$?

When the answer is **yes**, have the model checker return, for each run, a certificate $C(\mathcal{S}, P)$ such that

$$C(\mathcal{S}, P) \text{ valid} \implies \mathcal{S} \models P$$

Model checkers answers: yes / no

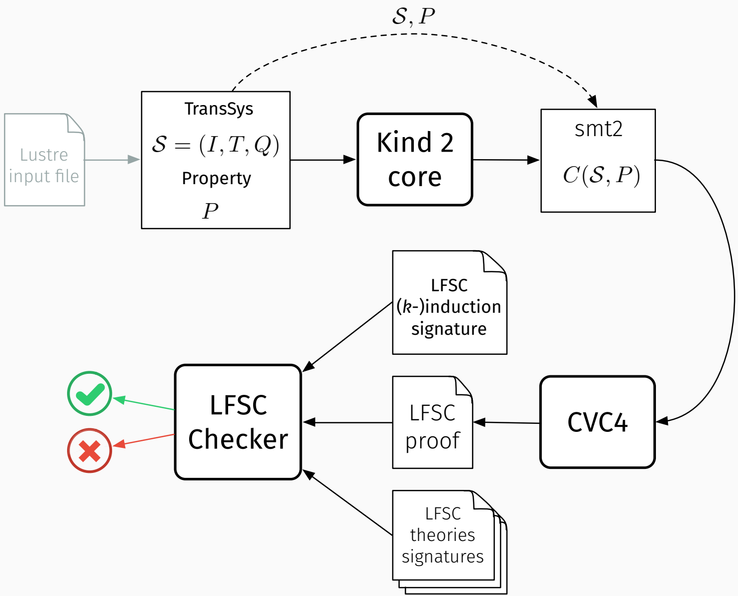
- **no** $\longrightarrow$ counterexample
- **yes** $\longrightarrow$?

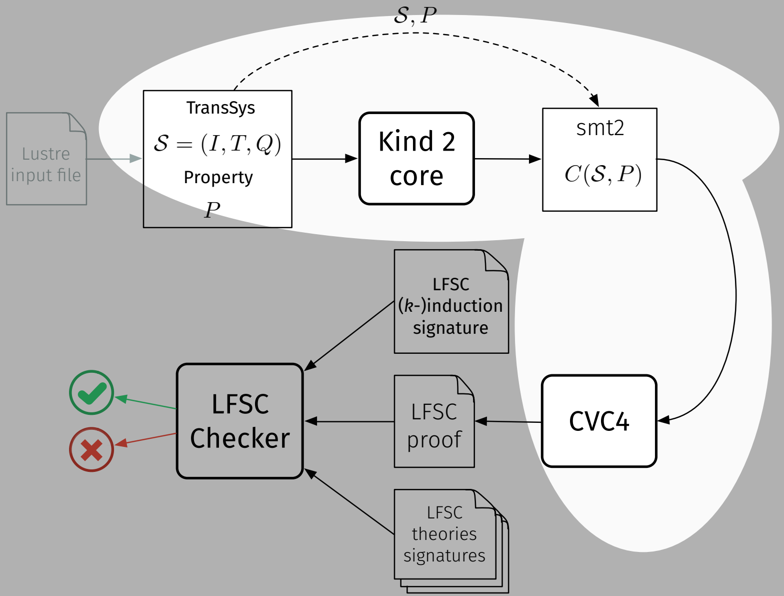
When the answer is **yes**, have the model checker return, for each run, a certificate $C(\mathcal{S}, P)$ such that

$$C(\mathcal{S}, P) \text{ valid} \implies \mathcal{S} \models P$$

Works if the certificate is small and easily verifiable.

CERTIFICATES IN KIND 2




$\mathcal{S}, P$
Lustre input file
TransSys
$\mathcal{S} = (I, T, Q)$
Property
P
Kind 2 core
smt2
$C(\mathcal{S}, P)$
LFSC (k-)induction signature
LFSC Checker
LFSC proof
CVC4
LFSC theories signatures

LFSC

Proof checker generator

Based on Edinburgh LF + side conditions

Efficient (side conditions compiled)

Small (3.5kloc C++)

CERTIFICATES IN KIND 2

$$C(\mathcal{S}, P) = (k, \phi)$$

such that ϕ is k-inductive in $\mathcal{S}$ and implies the property P.

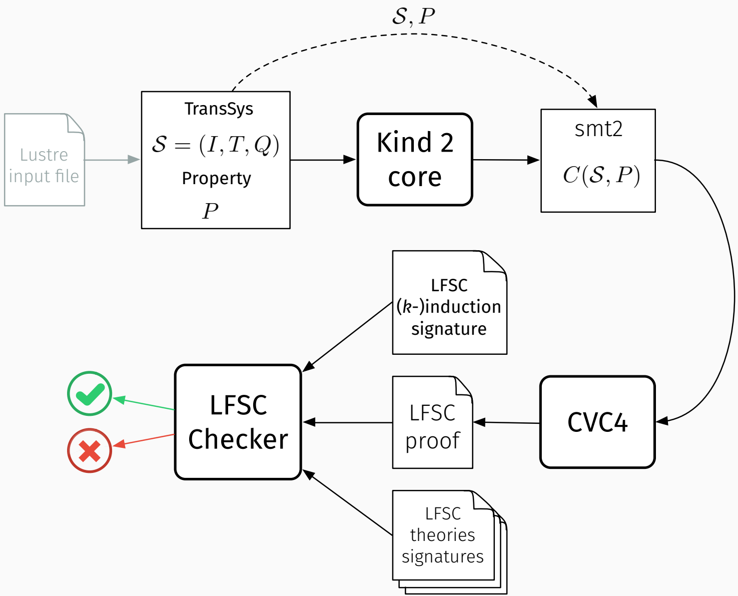
- only engine = k-induction $\longrightarrow (k, P)$
- only engine = PDR $\longrightarrow (1, P \wedge \varphi)$
- works for invariants discovered by k-induction

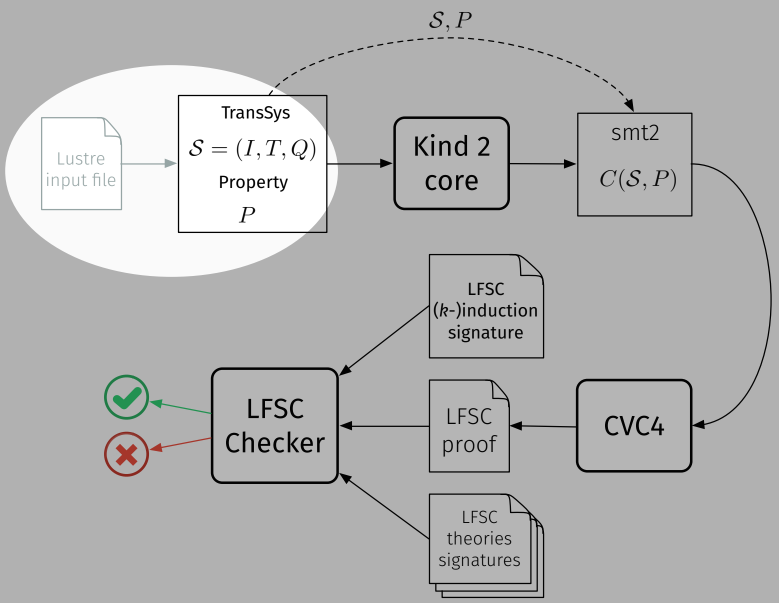
$$(max(k_1, \ldots, k_n, k_\phi),\ \mathcal{I}_1 \wedge \ldots \wedge \mathcal{I}_n \wedge \phi)$$

- may not work if lifting through condacts
- simplified through *a posteriori* induction check with unsat cores extraction and fixpoint

Benchmark	Kind 2 time	k	size	gen time	check (CVC 4)
simple_counter.lus	0.16	1	2	0.02	0.01
add_two.lus	0.15	1	1	0.02	0.01
dfa.lus	0.15	1	4	0.03	0.02
inv_gen.lus	0.21	1	2	0.03	0.01
triangle_peg_impossible.lus	7.47	9	1	20.17	5.70
bridge_and_torch.lus	3.50	3	23	0.31	0.21
pilot_flying.lus	13.96	1	50	0.20	0.22
pid.lus	8.73	24	1	4.69	3.64
microwave.kind.lus	2.32	2	23	0.21	0.32
active-standby.kind.lus	7.88	1	20	0.44	0.84
WBS.lus	806	26	91	866	2036 (Z3)
Leader_Selection.lus	1247	41	31	1026	4049 (Z3)
Pilot_Flying.lus	3342	52	83	5581	10628 (Z3)

times in s


$\mathcal{S}, P$
Lustre input file
TransSys
$\mathcal{S} = (I, T, Q)$
Property
P
Kind 2 core
smt2
$C(\mathcal{S}, P)$
LFSC (*k*-)induction signature
LFSC Checker
LFSC proof
CVC4
LFSC theories signatures


$\mathcal{S}, P$
Lustre input file
TransSys
$\mathcal{S} = (I, T, Q)$
Property
P
Kind 2 core
smt2
$C(\mathcal{S}, P)$
LFSC (k-)induction signature
LFSC Checker
LFSC proof
CVC4
LFSC theories signatures

FRONTEND CERTIFICATES IN KIND 2 (CONT.)

Translation from one formalism to another are sources of error.

In Kind 2,

- several intermediate representations
- many simplifications (slicing, path compression, encodings, ...)

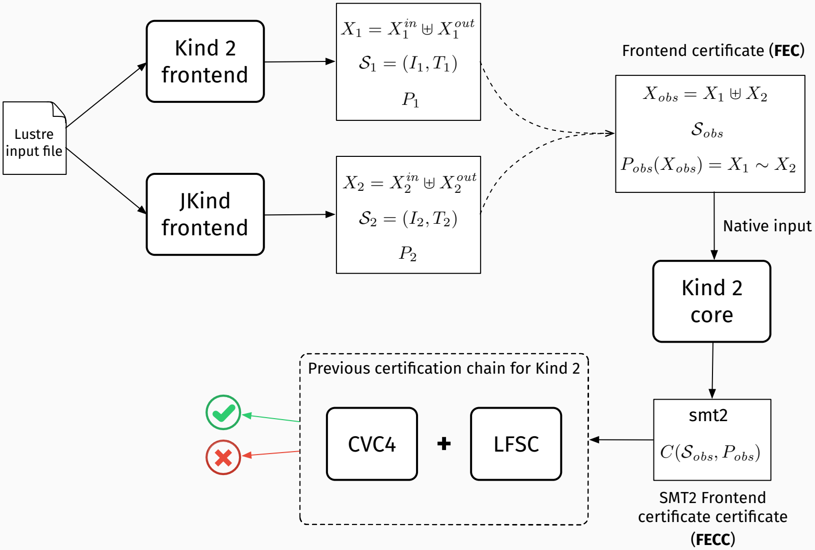
Translation from one formalism to another are sources of error.

In Kind 2,

- several intermediate representations
- many simplifications (slicing, path compression, encodings, ...)

How to trust the translation from Lustre to FOL ?

FRONTEND CERTIFICATES IN KIND 2: APPROACH



FEC: Frontend Certificate

- Observational equivalence between internal transition systems of
  Kind 2: $\mathcal{S}_1 = (\bar{x}_1\bar{y}_1, I_1, T_1)$
  JKind: $\mathcal{S}_2 = (\bar{x}_2\bar{y}_2, I_2, T_2)$
- $\mathcal{S}_{obs} = (\bar{x}_{obs}, I_{obs}, T_{obs})$ with
  - $\bar{x}_{obs} = \bar{x}_1\bar{y}_1\bar{x}_2\bar{y}_2$
  - $I_{obs}(\bar{x}_{obs}) = \bar{x}_1 \sim \bar{x}_2 \ \wedge\ I_1(\bar{x}_1\bar{y}_1) \ \wedge\ I_2(\bar{x}_2\bar{y}_2)$
  - $T_{obs}(\bar{x}_{obs}, \bar{x}'_{obs}) = \bar{x}'_1 \sim \bar{x}'_2 \ \wedge\ T_1(\bar{x}_1\bar{y}_1, \bar{x}'_1\bar{y}'_1) \ \wedge\ T_2(\bar{x}_2\bar{y}_2, \bar{x}'_2\bar{y}'_2)$
  - $P_{obs}(\bar{x}_{obs}) = \bar{x}_1\bar{y}_1 \sim \bar{x}_2\bar{y}_2$

FECC: Frontend Certificate Certificate

- SMT2 certificate obtained from the execution of Kind 2 on the FEC $\mathcal{S}_{obs}$

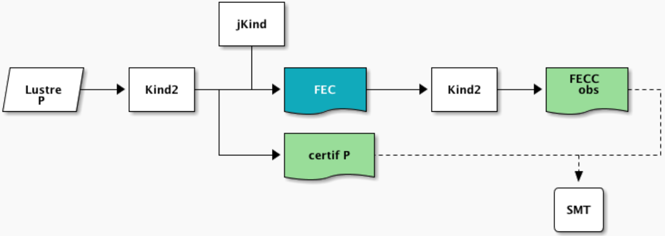
PRELIMINARY RESULTS

Benchmark	Kind 2 time	FEC				FECC			
		gen	size	MC	k	gen	k	size	check (Z3)
simple_counter.lus	0.16	0.73	2	0.16	1	0.02	1	4	0.01
add_two.lus	0.15	0.74	4	0.18	1	0.02	1	4	0.01
dfa.lus	0.16	0.73	4	0.17	1	0.02	1	4	0.01
inv_gen.lus	0.22	0.72	3	0.16	1	0.02	1	4	0.01
triangle_peg_im...	7.90	1.08	158	1.53	1	0.42	1	158	0.44
bridge_and_torc...	1.04	0.80	62	0.24	1	0.12	1	62	0.05
pilot_flying.lus	19.83	1.05	107	0.38	1	0.24	1	107	0.12
pid.lus	8.81	0.80	15	0.18	1	0.04	1	15	0.02
microwave.kind.lus	2.57	1.07	46	35.7	6	1.07	3	79	0.81
active-...	6.54	1.19	75	3.50	1	1.31	1	56	0.67

times in s

For the moment we have three certificates:

- the certificate of invariance of the property (SMT2)
- the frontend certificate (FEC) of observational equivalence between Kind 2 and JKind (Kind 2 system)
- the corresponding frontend certificate certificate (FECC) (SMT2)



TRUSTING A CERTIFYING MODEL CHECKER

1. **Trusted Logic** — The logics used by the model checker are trusted.
2. **Valid Model** — The model and properties accurately depict the system under scrutiny in the semantic given by the logic of the model checker.
3. **Correct Certificate** — The certificate produced by the model checker is sufficient to convince that the properties are true of the system under scrutiny.
4. **Correct Certificate Checker** — The program that checks the certificates is sound and correctly implemented (and compiled/executed).
5. **Trusted IO** — The parsing and output of the *certificate* checker are trusted.

FUTURE WORK

Increase trust (FEC)

Produce LFSC proofs

Glue

Qualify LFSC

- Prove soundness rules (side conditions = functional program), *or*
- Trust rules

Thank you.