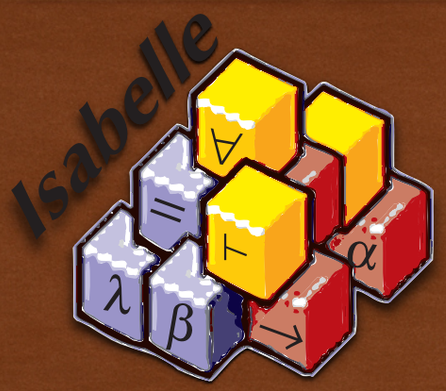


MUNTA: A VERIFIED MODEL CHECKER FOR TIMED AUTOMATA

SIMON WIMMER

FAKULTÄT FÜR INFORMATIK,
TECHNISCHE UNIVERSITÄT MÜNCHEN

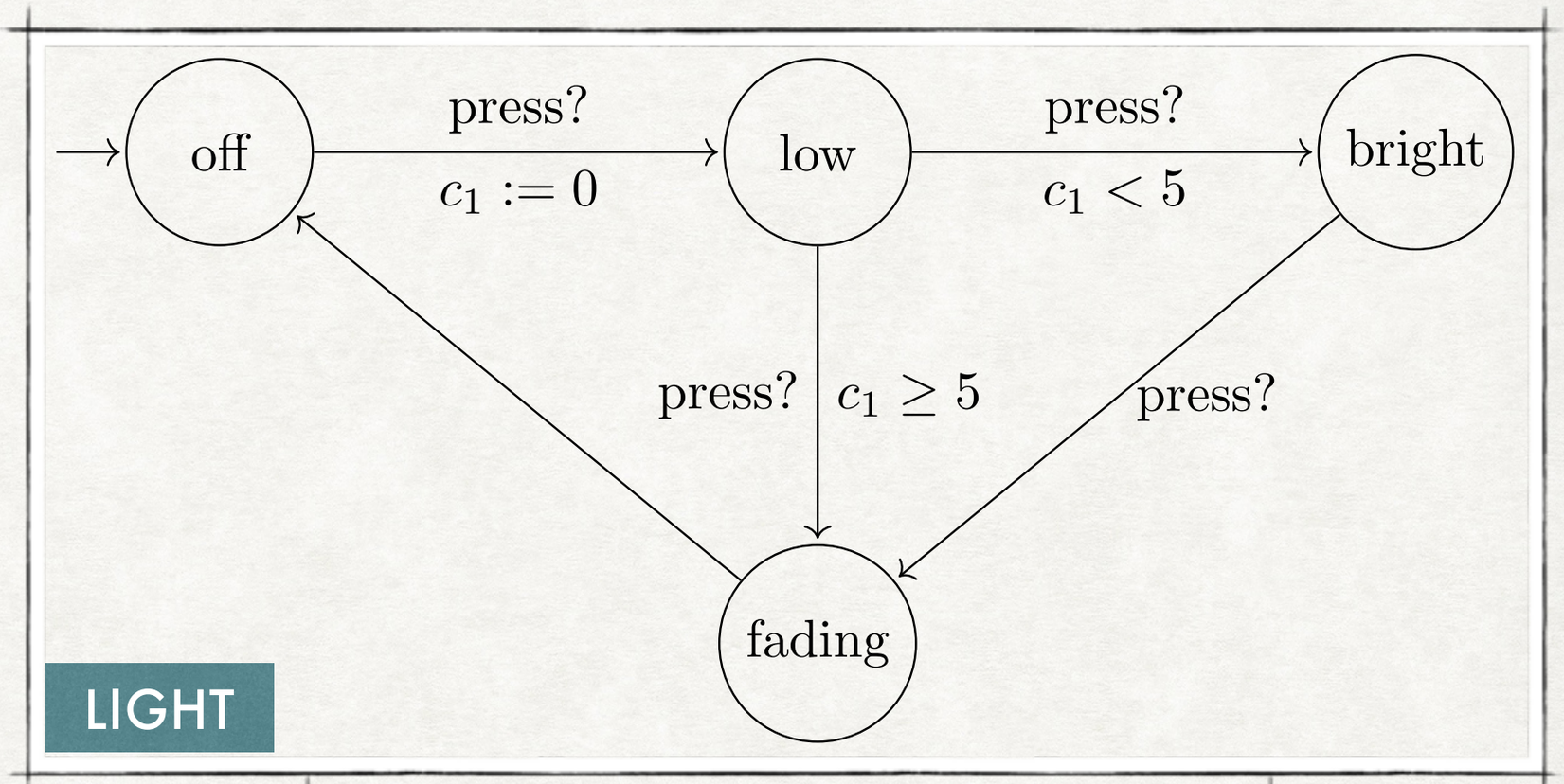
MOTIVATION

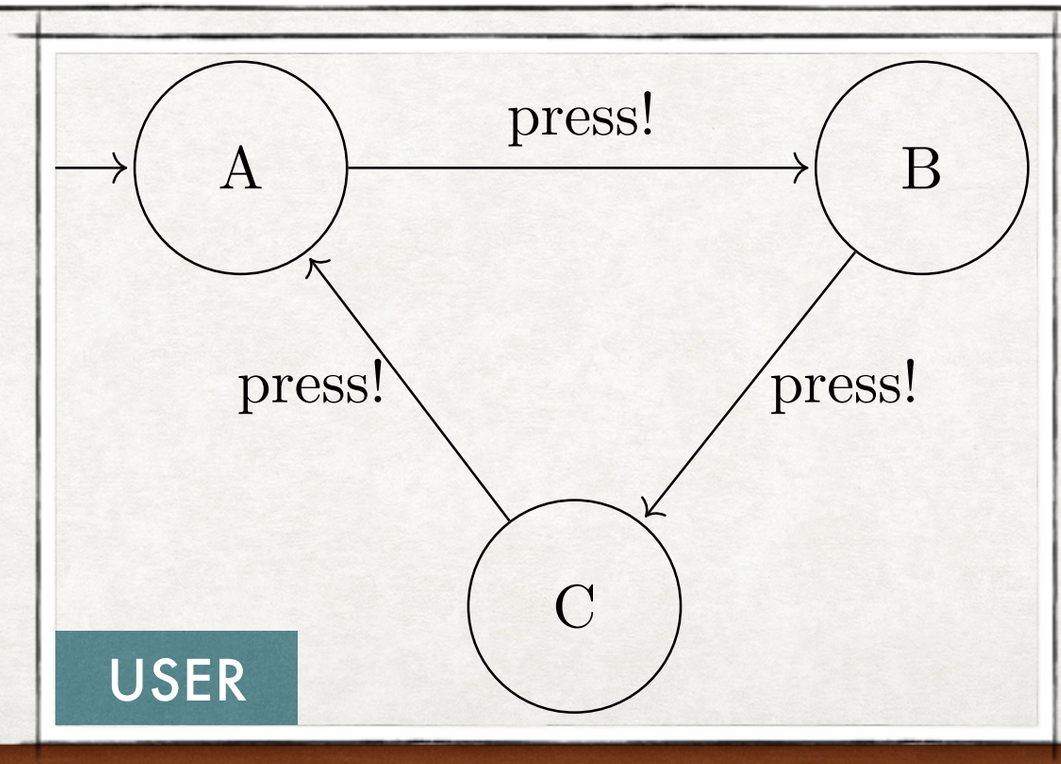
- Model checkers as **trust-multipliers**
- Goals for trustworthy model checkers
  - Theory should be sound and well-understood
  - Implementation of theory should be correct
- Verification with a proof assistant (Isabelle/HOL) guarantees both!
- Here: timed automata model checking

TIMED AUTOMATA

LIGHT SWITCH EXAMPLE

- One press: light turns low
- Two quick presses: light turns bright
- Two slow presses: light turns off



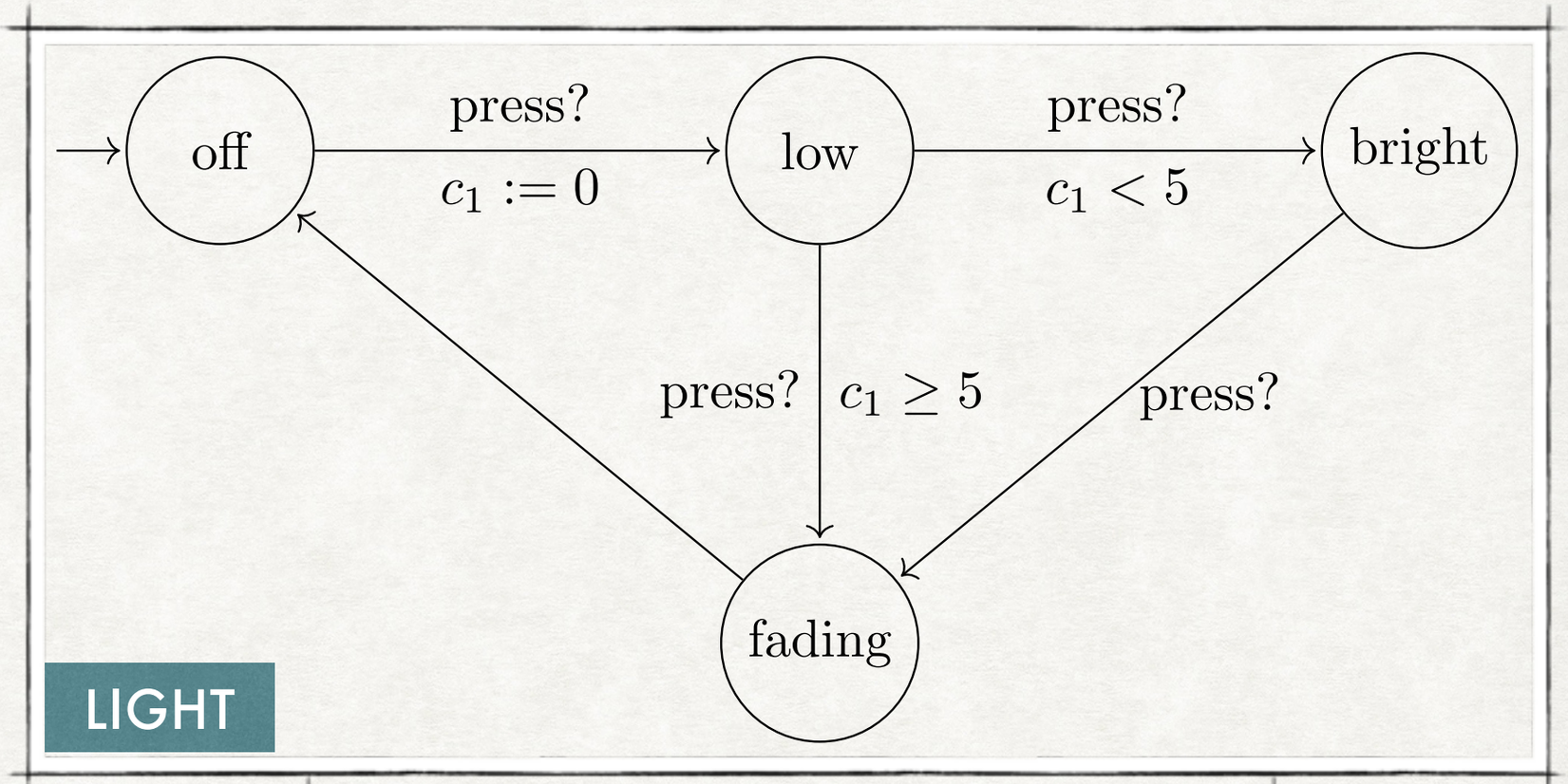


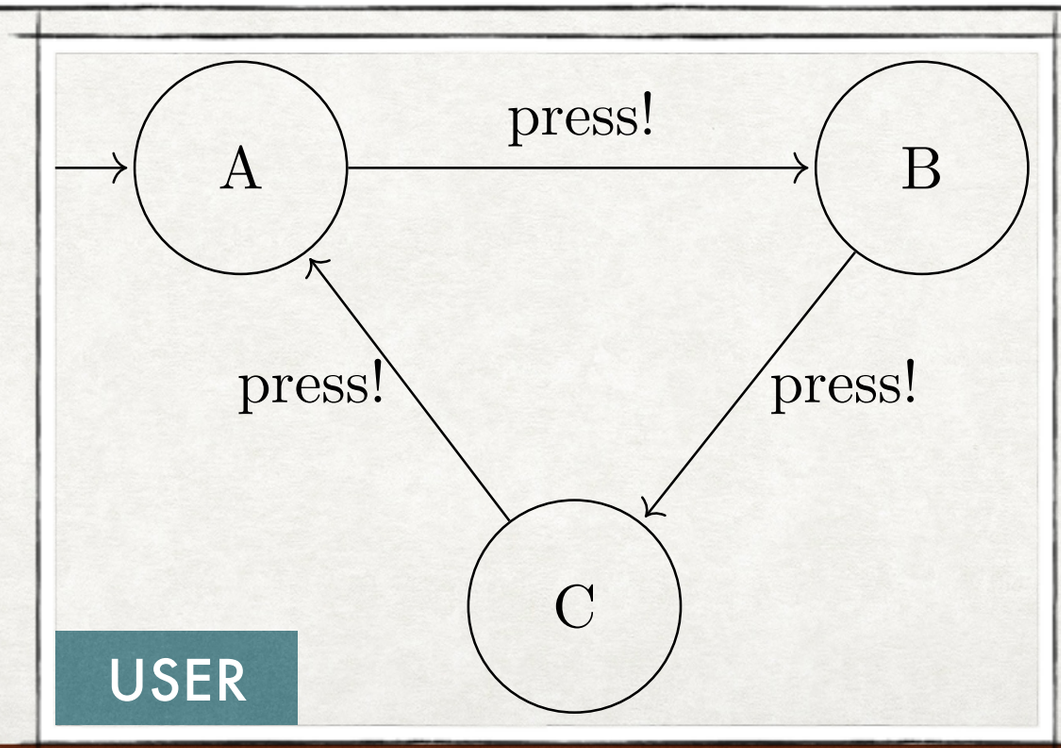
TIMED AUTOMATA

LIGHT SWITCH EXAMPLE

- One press: light turns low
- Two quick presses: light turns bright
- Two slow presses: light turns off

$$\mathbf{E}\Diamond\ light.bright$$

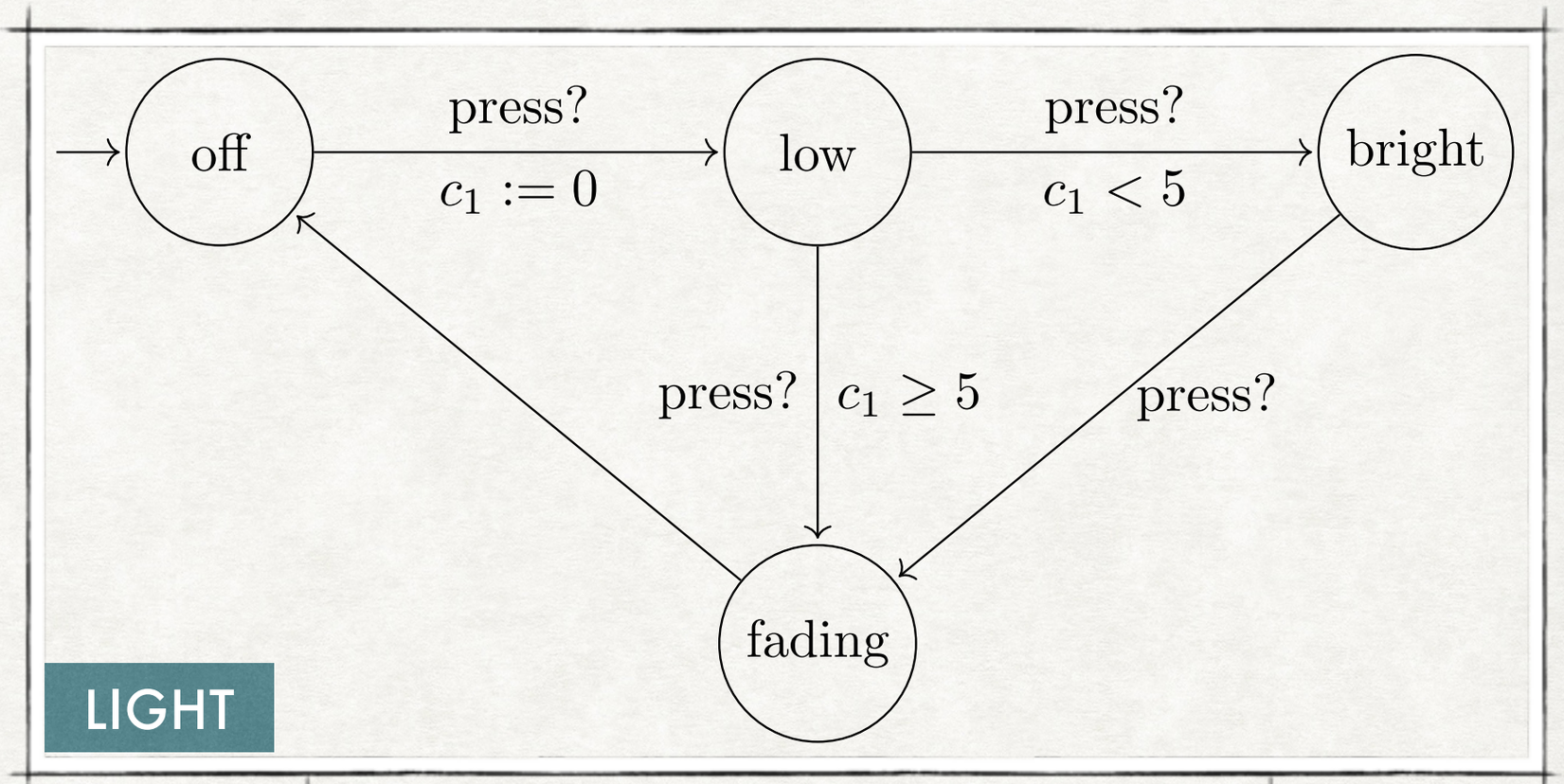




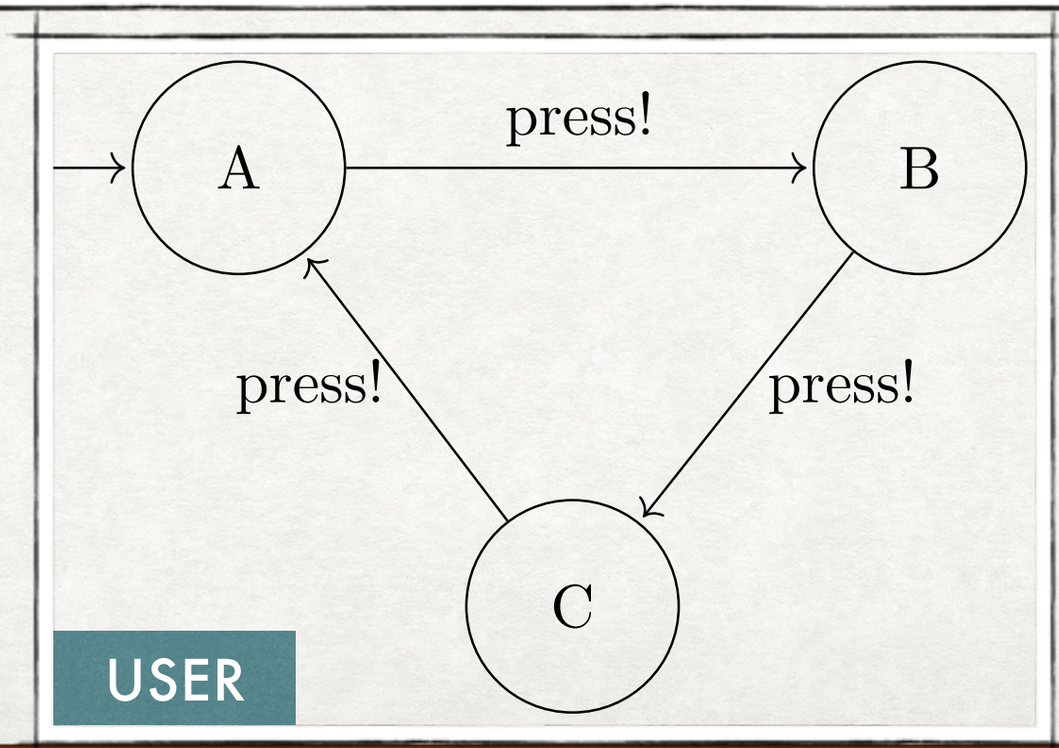
TIMED AUTOMATA

LIGHT SWITCH EXAMPLE

- One press: light turns low
- Two quick presses: light turns bright
- Two slow presses: light turns off



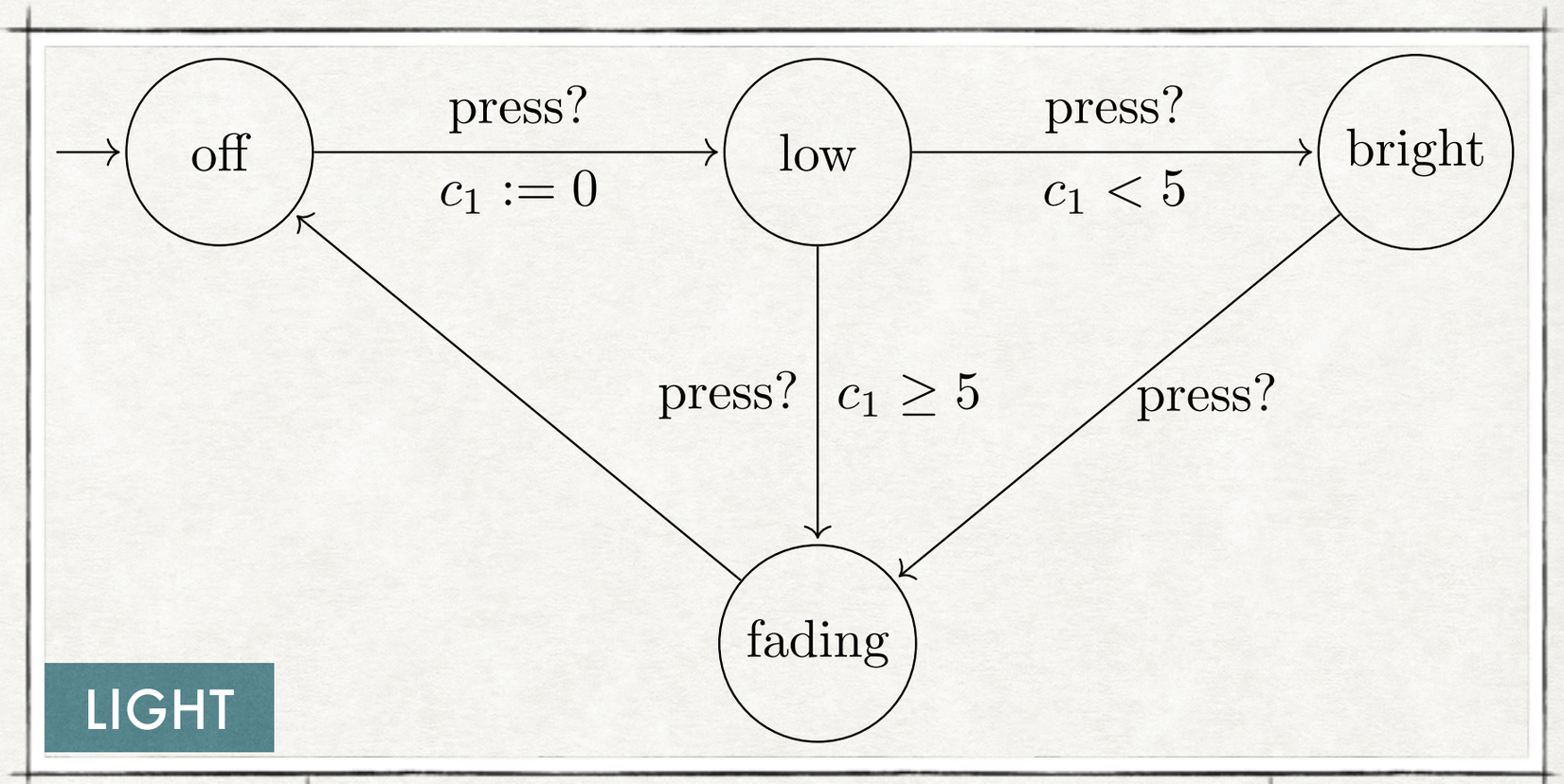
$\mathbf{E}\Diamond\ light.bright$ ✓

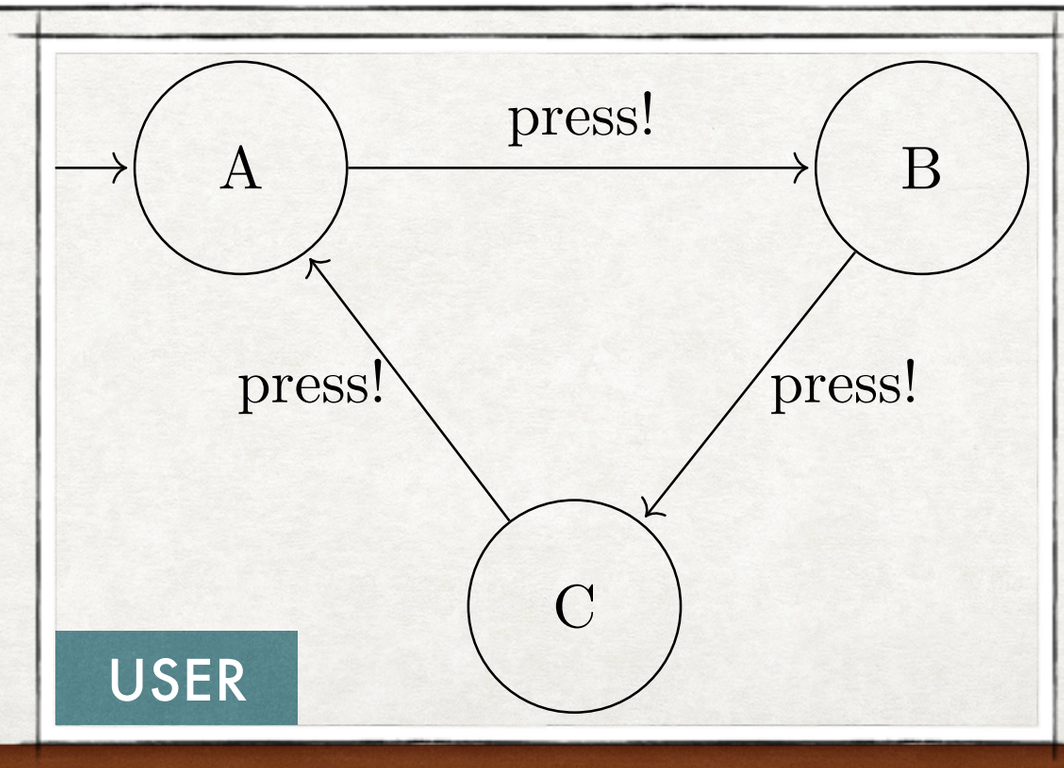


TIMED AUTOMATA

LIGHT SWITCH EXAMPLE

- One press: light turns low
- Two quick presses: light turns bright
- Two slow presses: light turns off





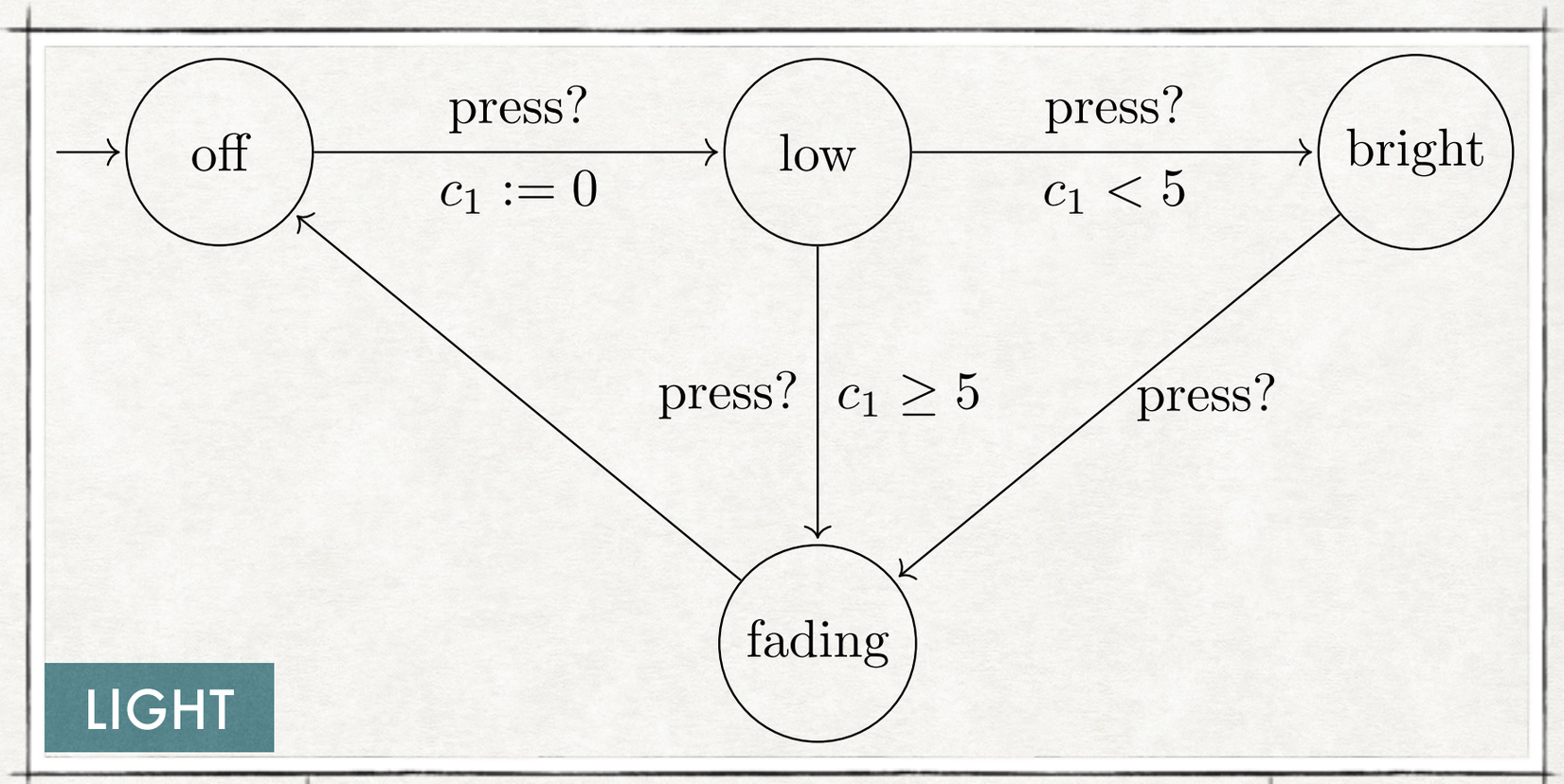
$\mathbf{E}\Diamond\ light.bright$ ✓

$\mathbf{A}\Diamond\ light.bright$

TIMED AUTOMATA

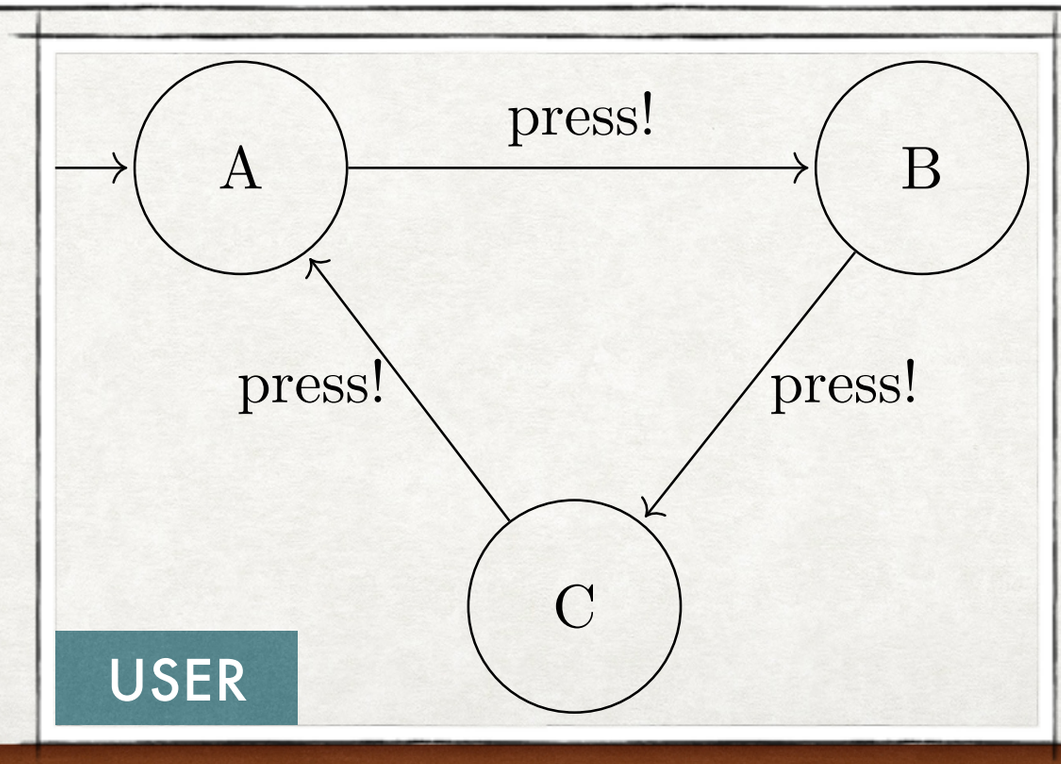
LIGHT SWITCH EXAMPLE

- One press: light turns low
- Two quick presses: light turns bright
- Two slow presses: light turns off



$\mathbf{E}\Diamond\ light.bright$ ✓

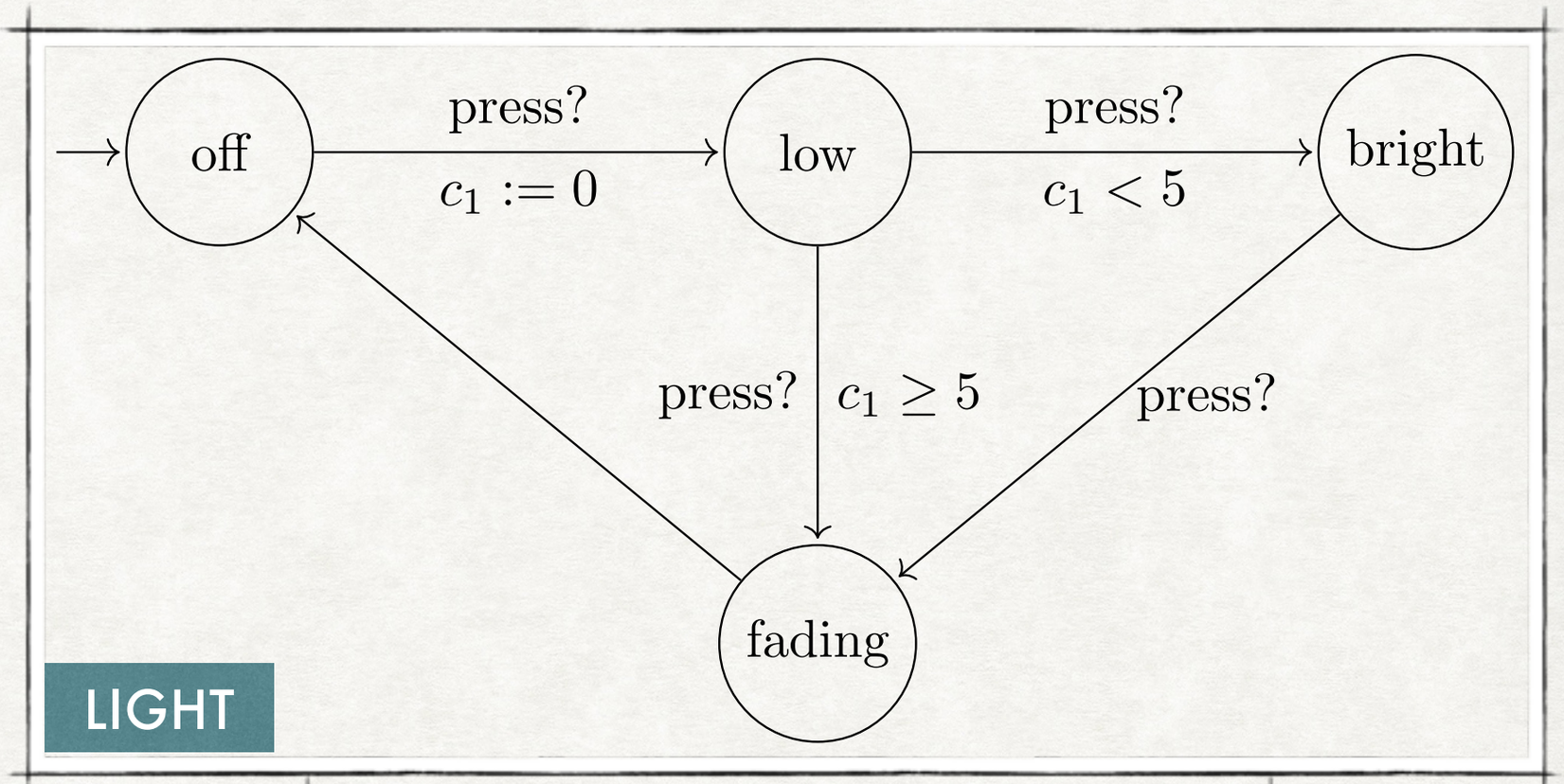
$\mathbf{A}\Diamond\ light.bright$ ✗

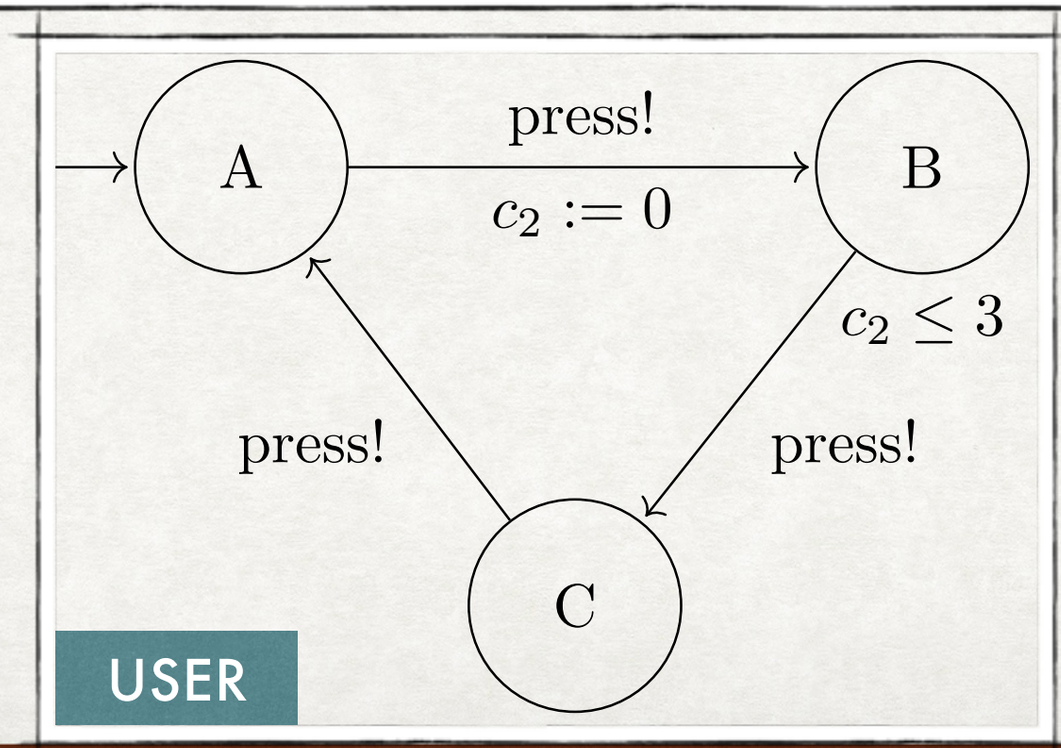


TIMED AUTOMATA

LIGHT SWITCH EXAMPLE

- One press: light turns low
- Two quick presses: light turns bright
- Two slow presses: light turns off





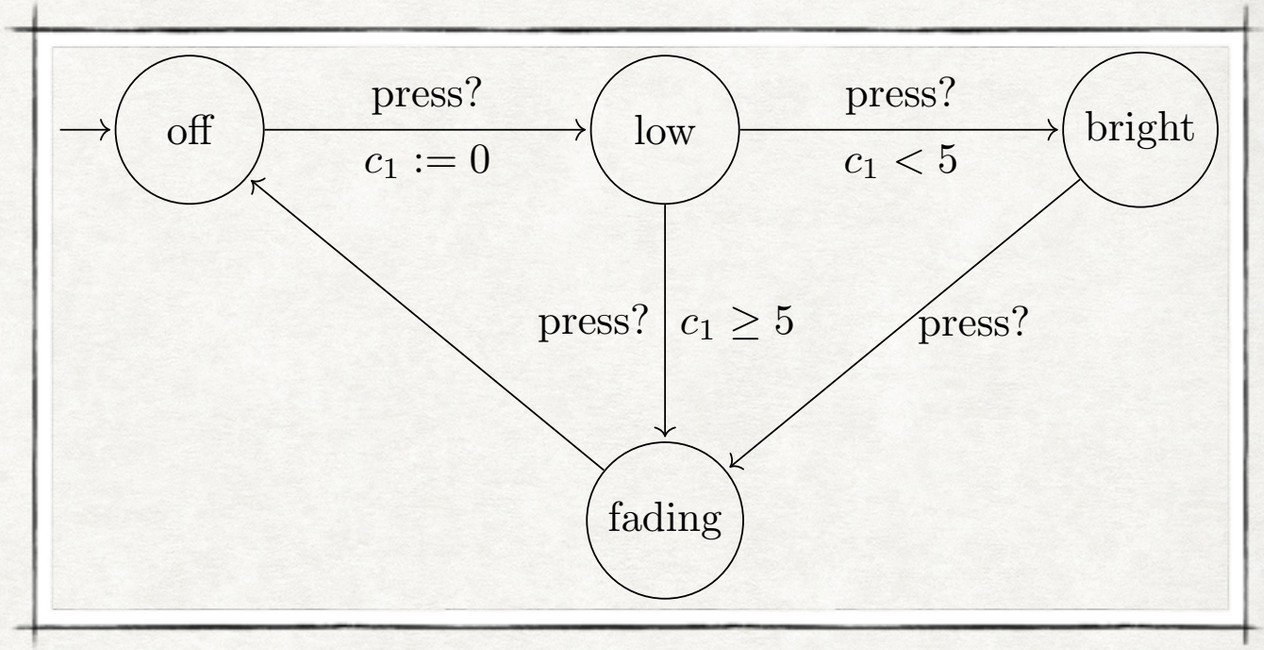
$\mathbf{E}\Diamond\ light.bright$ ✓

$\mathbf{A}\Diamond\ light.bright$ ✓

TIMED AUTOMATA

SEMANTICS

- Types of transitions: **delay** and **action**
- Clock valuations: $\mathbb{N} \Rightarrow \mathbb{R}$
  $\rightarrow$ Infinite Semantics
- Clock constraints:
  $(\cdot \mapsto 1) \vdash c_1 > 0 \wedge c_2 \leq 3$
  $\rightarrow$ Invariants on nodes and guards on edges

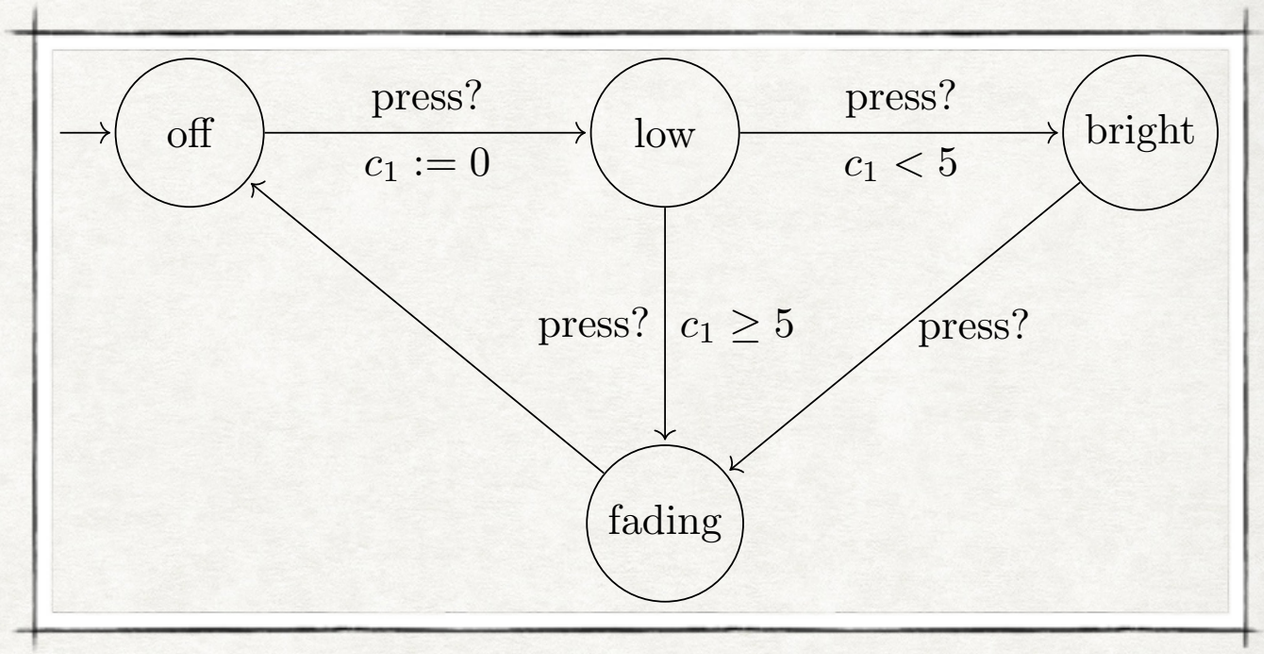


MODEL CHECKING

- Clock valuations: $\mathbb{N} \Rightarrow \mathbb{R}$
  → Infinite Semantics

- Concrete states *(l, u)* to abstract states *(l, Z)*
  - node *l*
  - clock valuation *u*: $\mathbb{N} \Rightarrow \mathbb{R}$
  - *Z* a set of clock valuations (zone): $P(\mathbb{N} \Rightarrow \mathbb{R})$

- Symbolic computation: zones as clock constraints
  → Difference Bound Matrices (DBMs)



OBJECTIVE

OBJECTIVE

- Provide verified reference implementation for TA MC
  - Not meant to replace existing MCs
  - Rather allow validation of existing MCs against it
- Experimentation platform

OBJECTIVE

- Provide verified reference implementation for TA MC
  - Not meant to replace existing MCs
  - Rather allow validation of existing MCs against it
    - Experimentation platform
- Thus we need:
  - Acceptable performance
  - A practical modeling formalism

AGENDA

- FEATURES
- CORRECTNESS CLAIM
- ARCHITECTURE
- EXPERIMENTS
- FUTURE WORK

MODELING LANGUAGE

WHAT CAN WE MODEL?

MODELING LANGUAGE

WHAT CAN WE MODEL?

- TA networks w/ sync. over channels incl. broadcast channels

MODELING LANGUAGE

WHAT CAN WE MODEL?

- TA networks w/ sync. over channels incl. broadcast channels
- Shared finite state
  - Set of integer variables
  - Boolean & arithmetic expressions for guards and updates

MODELING LANGUAGE

WHAT CAN WE MODEL?

- TA networks w/ sync. over channels incl. broadcast channels
- Shared finite state
  - Set of integer variables
    - Boolean & arithmetic expressions for guards and updates
- Urgent and committed locations

MODELING LANGUAGE

WHAT CAN WE MODEL?

- TA networks w/ sync. over channels incl. broadcast channels
- Shared finite state
  - Set of integer variables
    - Boolean & arithmetic expressions for guards and updates
- Urgent and committed locations
- **Restrictions**: diagonal free TA & clock updates to 0

MODELING LANGUAGE

WHAT CAN WE MODEL?

- TA networks w/ sync. over channels incl. broadcast channels
- Shared finite state
  - Set of integer variables
    - Boolean & arithmetic expressions for guards and updates
- Urgent and committed locations
- **Restrictions**: diagonal free TA & clock updates to 0
- Formalized semantics: ~150 lines of Isabelle text

MODEL CHECKING CAPABILITIES

WHAT CAN WE CHECK?

MODEL CHECKING CAPABILITIES

WHAT CAN WE CHECK?

- Input format: JSON
  - Easy data exchange and parsing
  - No templating: to improve inter-operability

MODEL CHECKING CAPABILITIES

WHAT CAN WE CHECK?

- Input format: JSON
  - Easy data exchange and parsing
  - No templating: to improve inter-operability
- CTL subset corresponding to TCTL subset supported by UPPAAL

  $\mathbf{E}\Diamond$, $\mathbf{A}\Diamond$, $\mathbf{E}\Box$, $\mathbf{A}\Box$, $\dashrightarrow$

MODEL CHECKING CAPABILITIES

WHAT CAN WE CHECK?

- Input format: JSON
  - Easy data exchange and parsing
  - No templating: to improve inter-operability
- CTL subset corresponding to TCTL subset supported by UPPAAL

  $\mathbf{E}\Diamond$, $\mathbf{A}\Diamond$, $\mathbf{E}\Box$, $\mathbf{A}\Box$, $\dashrightarrow$
- Deadlock checking

MODEL CHECKING CAPABILITIES

WHAT CAN WE CHECK?

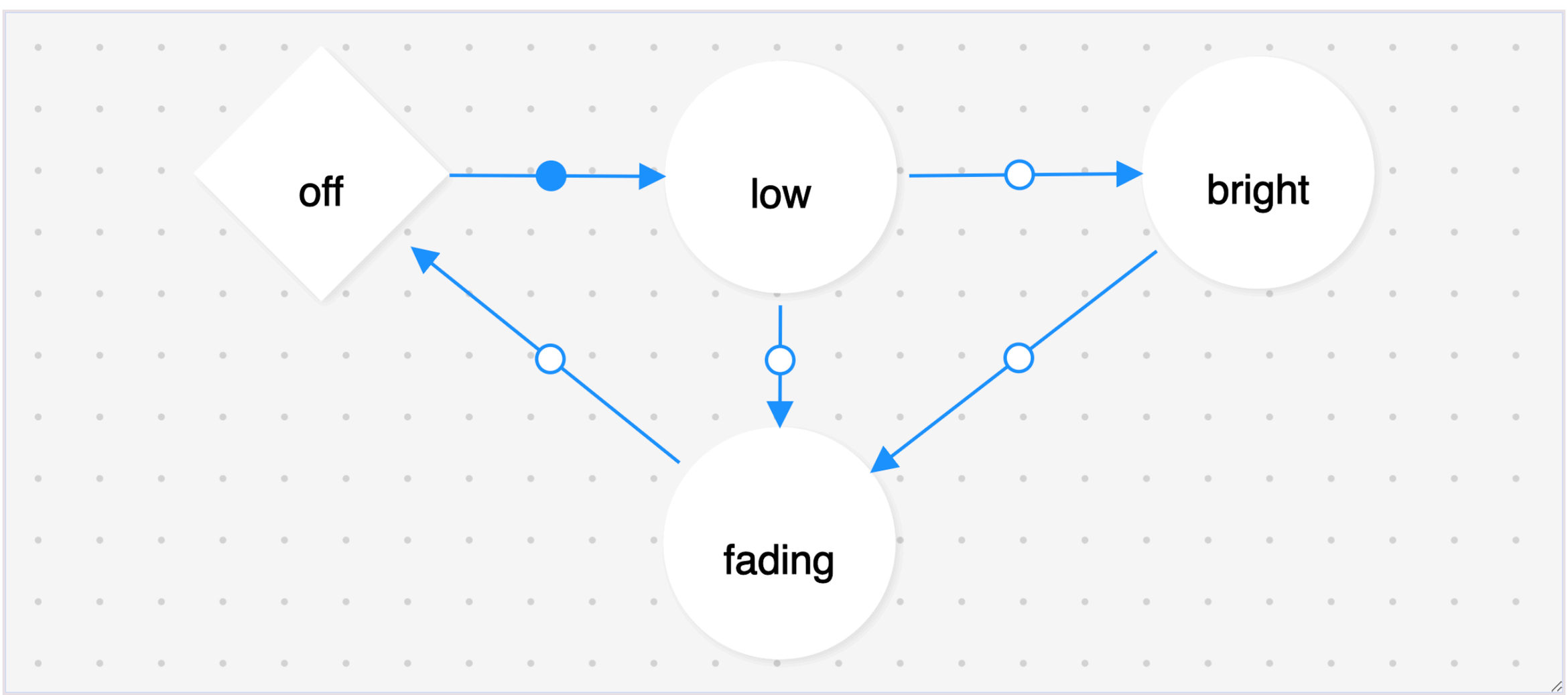
- Input format: JSON
  - Easy data exchange and parsing
  - No templating: to improve inter-operability
- CTL subset corresponding to TCTL subset supported by UPPAAL

  $\mathbf{E}\Diamond$, $\mathbf{A}\Diamond$, $\mathbf{E}\Box$, $\mathbf{A}\Box$, $\dashrightarrow$
- Deadlock checking
- Report full set of reachable states

MODEL CHECKING CAPABILITIES

WHAT CAN WE CHECK?

- Input format: JSON
  - Easy data exchange and parsing
  - No templating: to improve inter-operability
- CTL subset corresponding to TCTL subset supported by UPPAAL
  $\mathbf{E}\Diamond$, $\mathbf{A}\Diamond$, $\mathbf{E}\Box$, $\mathbf{A}\Box$, $\dashrightarrow$
- Deadlock checking
- Report full set of reachable states
- **Graphical user interface** (experimental): in the browser



Clocks:

c, x

Variables:

Declarations of integer variables
Example: x[-10:10], y[0:3]

Update:

c := 0

Guard:

Edge Guard

Label:

press?

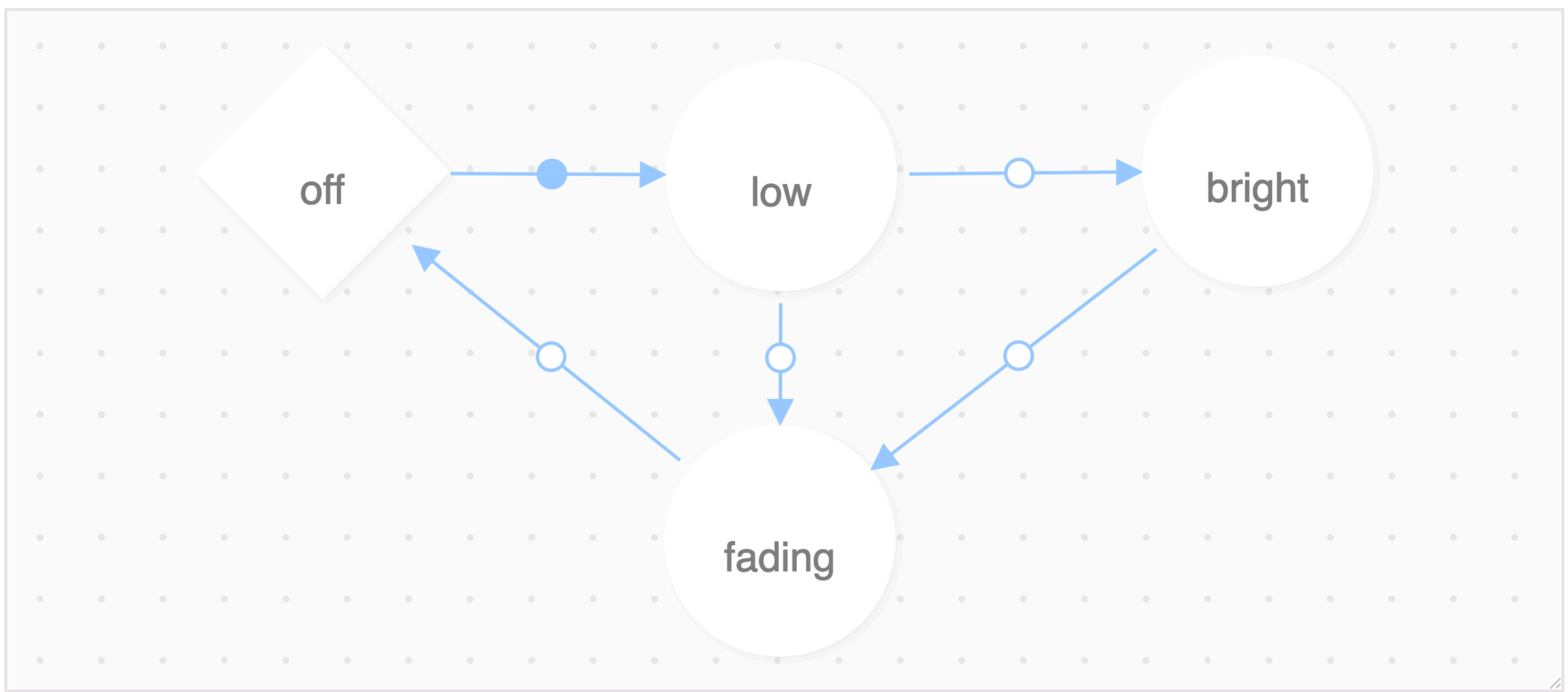
Automata:

Lamp | User

Formula:

A<> Lamp.bright

Save | Check input | Verify | Verify in your browser



Clocks:

c, x

Variables:

Declarations of integer variables
Example: x[-10:10], y[0:3]

Update:

c := 0

Guard:

Edge Guard

Label:

press?

Automata:

Lamp | User

Formula:

A<> Lamp.bright

Save | Check input | Verify | Verify in your browser

GRAPHICAL USER INTERFACE

MAKING IT ACCESSIBLE

GRAPHICAL USER INTERFACE

MAKING IT ACCESSIBLE

- Programmed in ReasonML: runs in the browser

GRAPHICAL USER INTERFACE

MAKING IT ACCESSIBLE

- Programmed in ReasonML: runs in the browser
- Interfacing with Munta
  - Model checking in the browser
  - Verification queries sent to local (primitive) verification server

GRAPHICAL USER INTERFACE

MAKING IT ACCESSIBLE

- Programmed in ReasonML: runs in the browser
- Interfacing with Munta
  - Model checking in the browser
    - Verification queries sent to local (primitive) verification server
- Consistency of graphical model and verified model
  - **Parse-print-parse** loop to produce input JSON
  - Possibility to manually inspect input JSON

CORRECTNESS CLAIM

ISABELLE/HOL

HOW DO WE VERIFY?

ISABELLE/HOL

HOW DO WE VERIFY?

- Isabelle: **LCF-style** proof assistant (or interactive theorem prover)
  - Any valid theorem has to pass through a small logical core
  - Tactics: bigger proofs from primitive logical inferences

ISABELLE/HOL

HOW DO WE VERIFY?

- Isabelle: **LCF-style** proof assistant (or interactive theorem prover)
  - Any valid theorem has to pass through a small logical core
  - Tactics: bigger proofs from primitive logical inferences
- Generic: many object logics supported
- Isabelle/HOL: most prominent object logic
  Higher Order Logic: functional programming + math

ISABELLE/HOL

HOW DO WE VERIFY?

- Isabelle: **LCF-style** proof assistant (or interactive theorem prover)
  - Any valid theorem has to pass through a small logical core
  - Tactics: bigger proofs from primitive logical inferences
- Generic: many object logics supported
- Isabelle/HOL: most prominent object logic
  Higher Order Logic: functional programming + math
- Isar: structured proof language for human-readable proofs
- Powerful automation: e.g. term rewriting and external ATPs

WHAT DO WE PROVE?

$$
\begin{array}{l}
\{emp\} \\
\quad precond\_mc\ p\ m\ k\ max\_steps\ I\ T\ prog\ formula\ bounds\ P\ s_0 \\
\{\lambda Some\ r \Rightarrow valid\_input\ p\ m\ max\_steps\ I\ T\ prog\ bounds\ P\ s_0\ na\ k \wedge \\
\quad (\neg\, deadlock\ (conv\ N)\ (init, s_0, u_0) \Longrightarrow \\
\qquad r = conv\ N, (init, s_0, u_0) \vDash_{max\_steps} formula) \\
\mid None \Rightarrow \neg\, valid\_input\ p\ m\ max\_steps\ I\ T\ prog\ bounds\ P\ s_0\ na\ k\}
\end{array}
$$

WHAT DO WE PROVE?

SEPARATION LOGIC HOARE TRIPLE IN IMPERATIVE HOL

$$\{emp\}$$
$$precond\_mc\ p\ m\ k\ max\_steps\ I\ T\ prog\ formula\ bounds\ P\ s_0$$
$$\{\lambda Some\ r \Rightarrow valid\_input\ p\ m\ max\_steps\ I\ T\ prog\ bounds\ P\ s_0\ na\ k \wedge$$
$$(\neg\ deadlock\ (conv\ N)\ (init, s_0, u_0) \Longrightarrow$$
$$r = conv\ N, (init, s_0, u_0) \vDash_{max\_steps} formula)$$
$$\mid None \Rightarrow \neg\ valid\_input\ p\ m\ max\_steps\ I\ T\ prog\ bounds\ P\ s_0\ na\ k\}$$

WHAT DO WE PROVE?

$$
\begin{array}{l}
\{emp\} \\
\quad precond\_mc\ p\ m\ k\ max\_steps\ I\ T\ prog\ formula\ bounds\ P\ s_0 \\
\{\lambda Some\ r \Rightarrow valid\_input\ p\ m\ max\_steps\ I\ T\ prog\ bounds\ P\ s_0\ na\ k \wedge \\
\quad (\neg\, deadlock\ (conv\ N)\ (init, s_0, u_0) \Longrightarrow \\
\qquad r = conv\ N, (init, s_0, u_0) \vDash_{max\_steps} formula) \\
\mid None \Rightarrow \neg\, valid\_input\ p\ m\ max\_steps\ I\ T\ prog\ bounds\ P\ s_0\ na\ k\}
\end{array}
$$

WHAT DO WE PROVE?

SUCCESS

$$
\begin{array}{l}
\{emp\} \\
\quad precond\_mc\ p\ m\ k\ max\_steps\ I\ T\ prog\ formula\ bounds\ P\ s_0 \\
\{\lambda Some\ r \Rightarrow valid\_input\ p\ m\ max\_steps\ I\ T\ prog\ bounds\ P\ s_0\ na\ k \wedge \\
\quad (\neg\ deadlock\ (conv\ N)\ (init, s_0, u_0) \Longrightarrow \\
\qquad r = conv\ N, (init, s_0, u_0) \vDash_{max\_steps} formula) \\
\mid None \Rightarrow \neg\ valid\_input\ p\ m\ max\_steps\ I\ T\ prog\ bounds\ P\ s_0\ na\ k\}
\end{array}
$$

FAILURE

WHAT DO WE PROVE?

SUCCESS

$$
\begin{array}{l}
\{emp\} \\
\quad precond\_mc\ p\ m\ k\ max\_steps\ I\ T\ prog\ formula\ bounds\ P\ s_0 \\
\{\lambda Some\ r \Rightarrow valid\_input\ p\ m\ max\_steps\ I\ T\ prog\ bounds\ P\ s_0\ na\ k \wedge \\
\quad (\neg\ deadlock\ (conv\ N)\ (init, s_0, u_0) \Longrightarrow \\
\qquad r = conv\ N, (init, s_0, u_0) \vDash_{max\_steps} formula) \\
\mid None \Rightarrow \neg\ valid\_input\ p\ m\ max\_steps\ I\ T\ prog\ bounds\ P\ s_0\ na\ k\}
\end{array}
$$

FAILURE

IS INPUT VALID AND WITHIN THE SUPPORTED FRAGMENT?

WHAT DO WE PROVE?

NO DEADLOCK

SUCCESS

SAT/UNSAT?

$$
\begin{array}{l}
\{emp\} \\
\quad precond\_mc\ p\ m\ k\ max\_steps\ I\ T\ prog\ formula\ bounds\ P\ s_0 \\
\{\lambda Some\ r \Rightarrow valid\_input\ p\ m\ max\_steps\ I\ T\ prog\ bounds\ P\ s_0\ na\ k \wedge \\
\quad (\neg\, deadlock\ (conv\ N)\ (init, s_0, u_0) \Longrightarrow \\
\qquad r = conv\ N, (init, s_0, u_0) \vDash_{max\_steps} formula) \\
\mid None \Rightarrow \neg\, valid\_input\ p\ m\ max\_steps\ I\ T\ prog\ bounds\ P\ s_0\ na\ k\}
\end{array}
$$

FAILURE

IS INPUT VALID AND WITHIN THE SUPPORTED FRAGMENT?

TRUSTED CODE BASE

HOW VERIFIED IS 'VERIFIED'?

- Formalization of the Modeling Language Semantics
- Formalization of Imperative HOL and its corresponding separation logic
- Isabelle/HOL's logical kernel
- Isabelle/HOL's code generator
- The target language's compiler and runtime system

TRUSTED CODE BASE

HOW VERIFIED IS 'VERIFIED'?

- Formalization of the Modeling Language Semantics ← MANUAL INSPECTION ↓
- Formalization of Imperative HOL and its corresponding separation logic
- Isabelle/HOL's logical kernel → generally assumed to be correct
- Isabelle/HOL's code generator
- The target language's compiler and runtime system

TRUSTED CODE BASE

HOW VERIFIED IS ‘VERIFIED’?

TRUSTED CODE BASE

HOW VERIFIED IS 'VERIFIED'?

- Formalization of the Modeling Language Semantics
- Formalization of Imperative HOL and its corresponding separation logic
- Isabelle/HOL's logical kernel
- Isabelle/HOL's code generator → Hupel & Nipkow 2018
- The target language's compiler and runtime system → CakeML

TRUSTED CODE BASE

HOW VERIFIED IS 'VERIFIED'?

- Formalization of the Modeling Language Semantics
- Formalization of Imperative HOL and its corresponding separation logic
- Isabelle/HOL's logical kernel
- Isabelle/HOL's code generator → Hupel & Nipkow 2018
- The target language's compiler and runtime system → CakeML
- CakeML: dialect of ML with a verified compiler and runtime system
- Hupel & Nipkow 2018: provably correct code extraction from HOL to CakeML

ARCHITECTURE

Single TA model checking

Forward exploration with
local clock ceiling extraploation

DBMs

Zones

Deadlock

TA

Regions

Parser

Error handling

Diagnostics

Product construction

Modeling language semantics

Clock ceiling

Renaming

HOL

JavaScript ← OCaml ← Code generator → SML —MLton→ Server

SINGLE AUTOMATON MODEL CHECKER

PRODUCT CONSTRUCTION

TRUSTED HOL CODE

MODELING LANGUAGE SEMANTICS

CERTIFIED HOL CODE

HOL

EXTRACTION OF EXECUTABLE CODE

SINGLE AUTOMATON MODEL CHECKER

PRODUCT CONSTRUCTION

TRUSTED HOL CODE

MODELING LANGUAGE SEMANTICS

CERTIFIED HOL CODE

EXTRACTION OF EXECUTABLE CODE

SINGLE AUTOMATON MODEL CHECKER

PRODUCT CONSTRUCTION

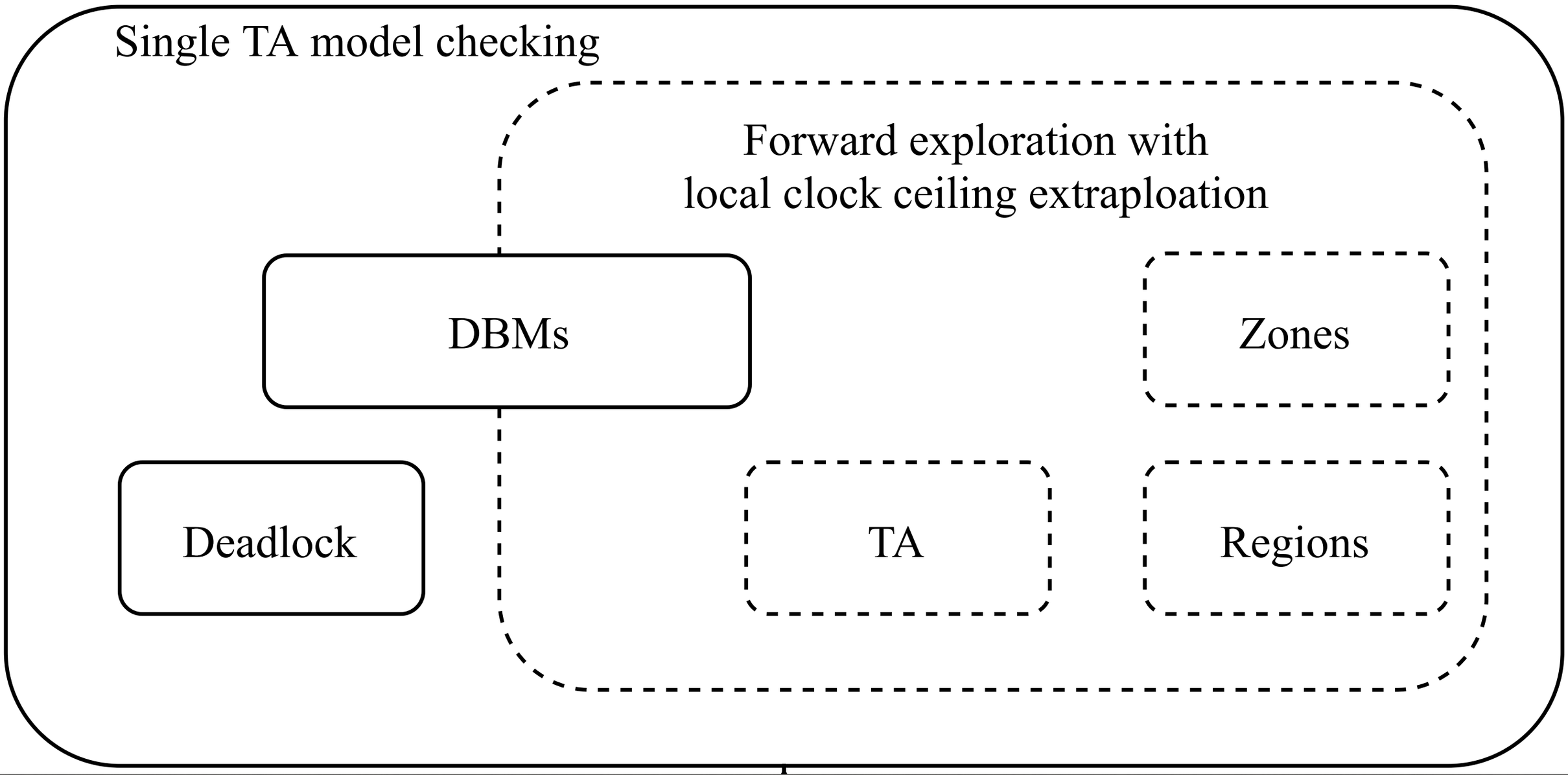
TRUSTED HOL CODE

MODELING LANGUAGE SEMANTICS

CERTIFIED HOL CODE

EXTRACTION OF EXECUTABLE CODE

SINGLE TA MODEL CHECKING



EFFICIENT IMPLEMENTATIONS

- Efficient Algorithms
  - DBMs as imperative arrays with destructive updates
  - Search algorithms with subsumption
- Expressive modelling language
  - Efficient on-the-fly product construction

EFFICIENT IMPLEMENTATIONS

- Efficient Algorithms → Refinement
  - DBMs as imperative arrays with destructive updates
    Imperative Refinement Framework:
    abstract functional impl. → efficient imperative impl.
  - Search algorithms with subsumption
    Stepwise refinement
- Expressive modelling language
  - Efficient on-the-fly product construction
    Refinement

SINGLE AUTOMATON MODEL CHECKER

PRODUCT CONSTRUCTION

TRUSTED HOL CODE

MODELING LANGUAGE SEMANTICS

CERTIFIED HOL CODE

EXTRACTION OF EXECUTABLE CODE

SINGLE AUTOMATON MODEL CHECKER

PRODUCT CONSTRUCTION

TRUSTED HOL CODE

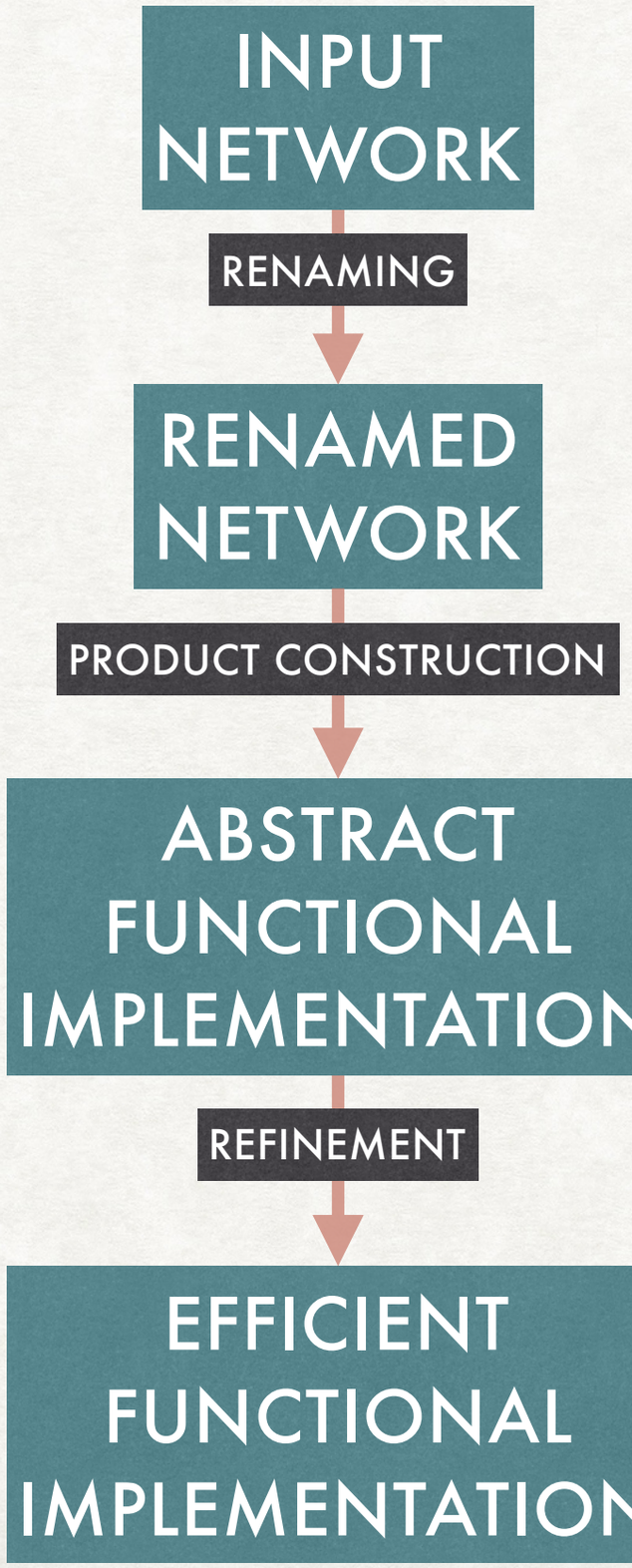
MODELING LANGUAGE SEMANTICS

CERTIFIED HOL CODE

EXTRACTION OF EXECUTABLE CODE

PRODUCT CONSTRUCTION

- From networks to single TA MC
  - Shared bounded integer variables
  - Networks with sync. over channels
- Retains ability to do MC **on the fly**
  - Result: transitions and invariants as functional programs



HOW CAN WE BUILD A REAL MODEL CHECKER?

SINGLE AUTOMATON MODEL CHECKER

PRODUCT CONSTRUCTION

TRUSTED HOL CODE

MODELING LANGUAGE SEMANTICS

CERTIFIED HOL CODE

EXTRACTION OF EXECUTABLE CODE

SINGLE AUTOMATON MODEL CHECKER

PRODUCT CONSTRUCTION

TRUSTED HOL CODE

MODELING LANGUAGE SEMANTICS

CERTIFIED HOL CODE

EXTRACTION OF EXECUTABLE CODE

CODE EXTRACTION

CONSTRUCTING A REAL TOOL

- **Code generator**: HOL specification of Munta → SML & OCaml
- Target languages
  - SML: fast executables with MLton
  - OCaml: compilation to JS via BuckleScript

SINGLE AUTOMATON MODEL CHECKER

PRODUCT CONSTRUCTION

TRUSTED HOL CODE

MODELING LANGUAGE SEMANTICS

CERTIFIED HOL CODE

EXTRACTION OF EXECUTABLE CODE

TRUSTED HOL CODE

CONSTRUCTING A REAL TOOL

- Missing: parsing, error handling, diagnostics
  → not verified & implemented in directly in HOL
- Considered non-critical: error handling, diagnostics
- Correctness of parser: check **parse-print-parse** loop

SINGLE AUTOMATON MODEL CHECKER

PRODUCT CONSTRUCTION

TRUSTED HOL CODE

MODELING LANGUAGE SEMANTICS

CERTIFIED HOL CODE

EXTRACTION OF EXECUTABLE CODE

CERTIFIED HOL CODE

CONSTRUCTING A REAL TOOL

- Certified:
  - Algorithms not verified but implemented in HOL
  - Results checked for soundness via verified HOL code
- Model relabeling
- Computation of local clock ceilings

EXPERIMENTS

				Our Tool			Uppaal		Ratio	
Model	Prop	SAT	Size	#states	$time_1$	$time_2$	#states	time	$\mathbf{TR_1}$	$\mathbf{TR_2}$
Fischer	R	N	5	38578	6,93	2,14	3739	0,062	**10,83**	**3,35**
	L	Y	5	42439	7,87	2,24	8149	0,112	**13,49**	**3,84**
		Y	6	697612	373	132	67325	1,94	**18,56**	**6,57**
FDDI	R	N	8	6720	35,1	8,92	5416	0,789	**35,85**	**9,11**
		N	10	29759	173	33,2	24120	6,64	**21,12**	**4,05**
	L	Y	6	2083	9,38	2,69	2439	0,159	**69,08**	**19,81**
		Y	7	3737	18,1	5,74	4944	0,406	**58,98**	**18,70**
CSMA/CD	R	N	5	9959	5,29	1,18	2769	0,102	**14,42**	**3,22**
		N	6	81463	72	15,6	17939	2,18	**7,27**	**1,58**
	L	Y	5	11526	5,81	1,28	3867	0,091	**21,33**	**4,70**
		Y	6	96207	76,4	16,6	23454	2,13	**8,74**	**1,90**

throughput = #states/time

Single TA model checking

Forward exploration with
local clock ceiling extraploation

DBMs

Zones

Deadlock

TA

Regions

Parser

Error handling

Diagnostics

Product construction

Modeling language semantics

Clock ceiling

Renaming

HOL

JavaScript ← OCaml ← Code generator → SML —MLton→ Server

>50000

Single TA

13000

Forward exploratio

local clock ceiling extr

8300

2000

DBMs

5100

Zones

650

Deadlock

4000

TA

Regions

6500

Parser

600

Diagnostics

Product construction

5000

Modeling language semantics

150

Clock ceiling

800

HOL

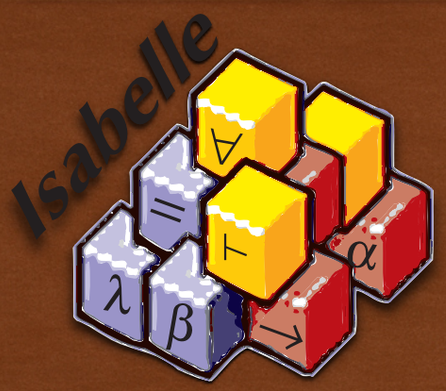
JavaScript ← OCaml ← Code generator

5400

MLton → Server

FUTURE

- Reduce trusted code base: CakeML
- Improve performance: verification w.r.t. C or LLVM
- Can we certify model checking results **efficiently**?
  TA MC uses **subsumption**: final invariant may be much smaller than total number of explored states
- Extensions: **Probabilistic Timed Automata**, LTL model checking, input formalism



THANK YOU!
QUESTIONS?

wimmers.github.io/munta