VERIFIED MODEL CHECKING OF TIMED AUTOMATA

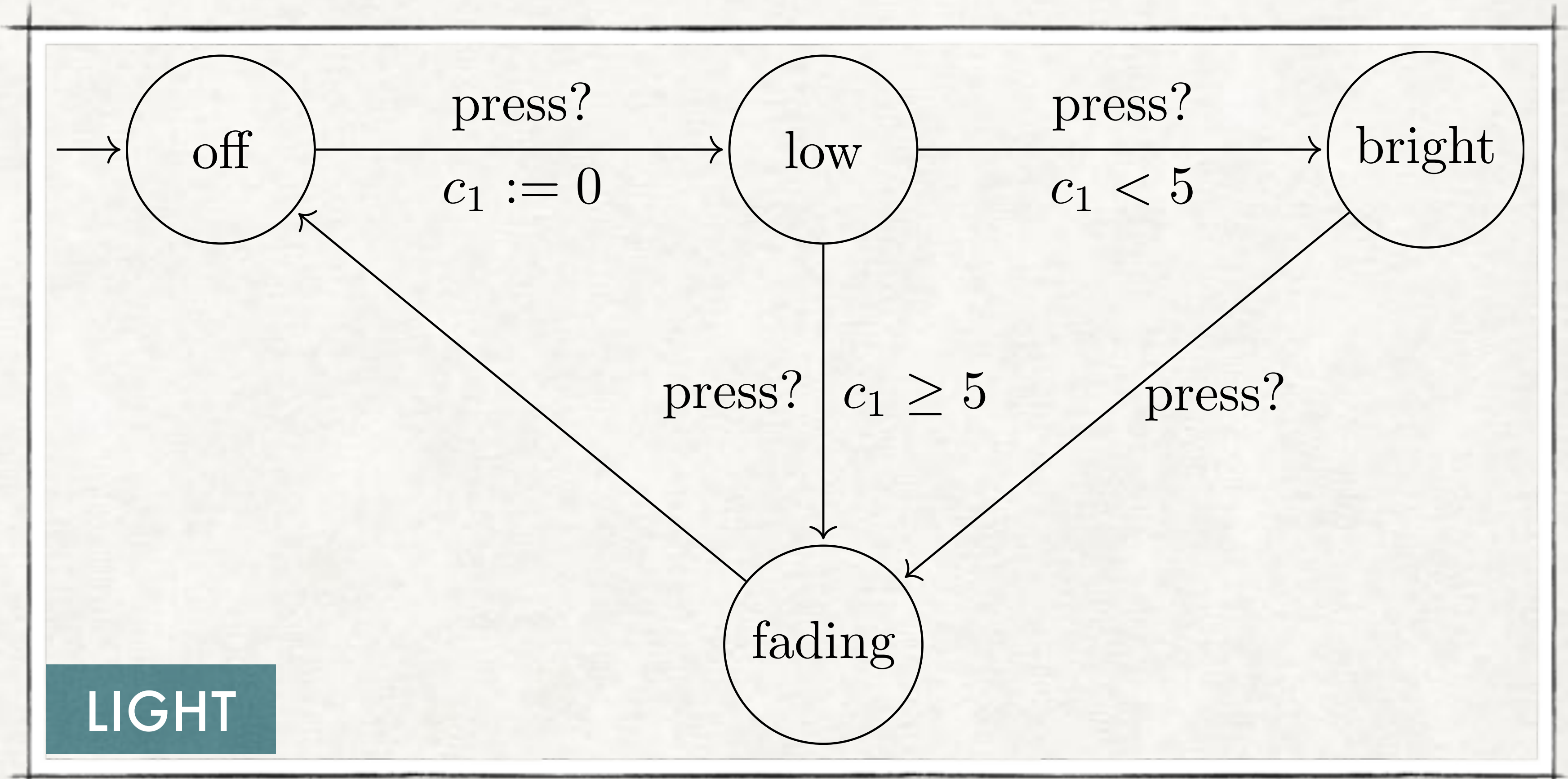
SIMON WIMMER AND PETER LAMMICH

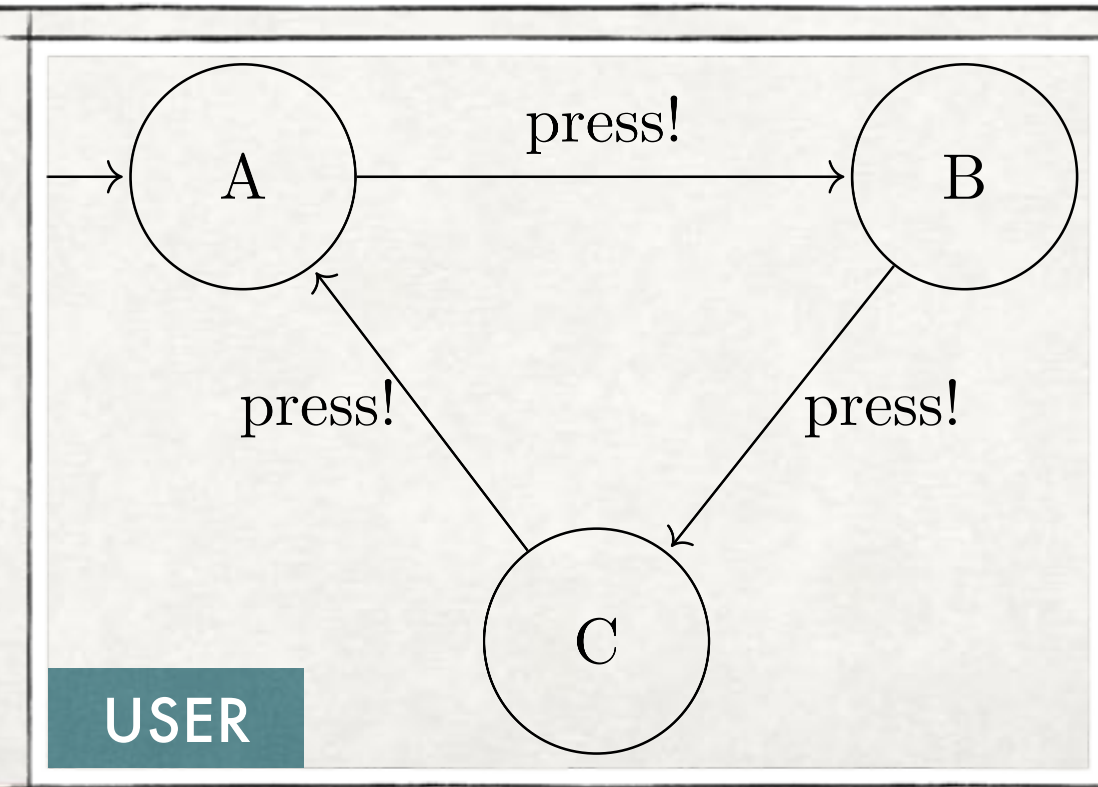
FAKULTÄT FÜR INFORMATIK,
TECHNISCHE UNIVERSITÄT MÜNCHEN

TIMED AUTOMATA

LIGHT SWITCH EXAMPLE

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





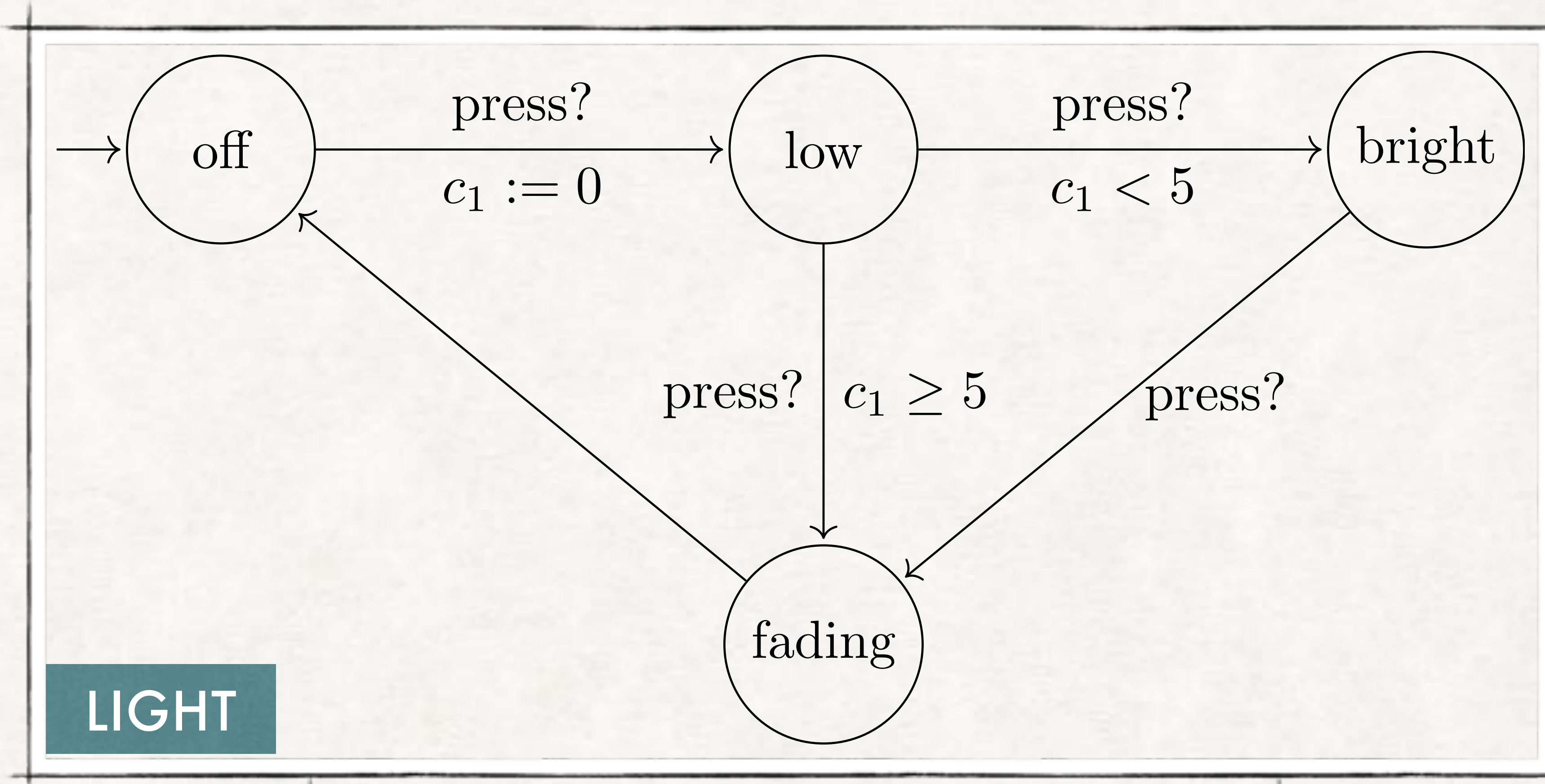
$\mathbf{E}\Diamond\ light.bright$ ✓

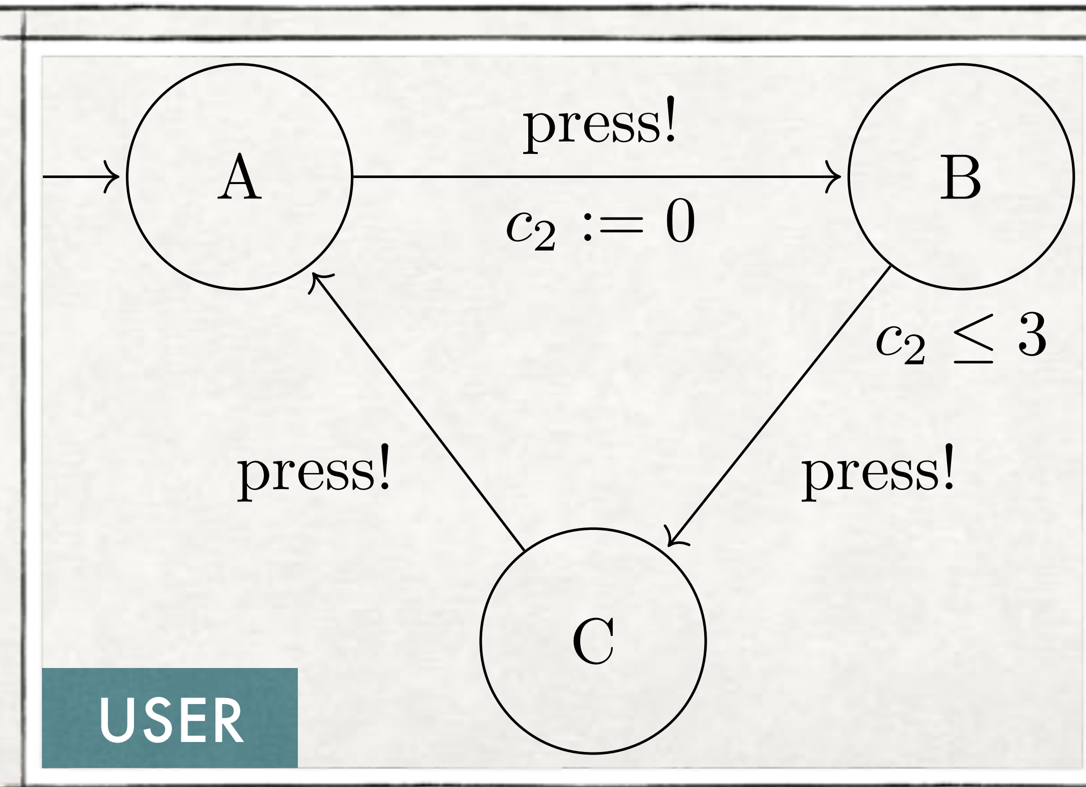
$\mathbf{A}\Diamond\ light.bright$ ✗

TIMED AUTOMATA

LIGHT SWITCH EXAMPLE

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





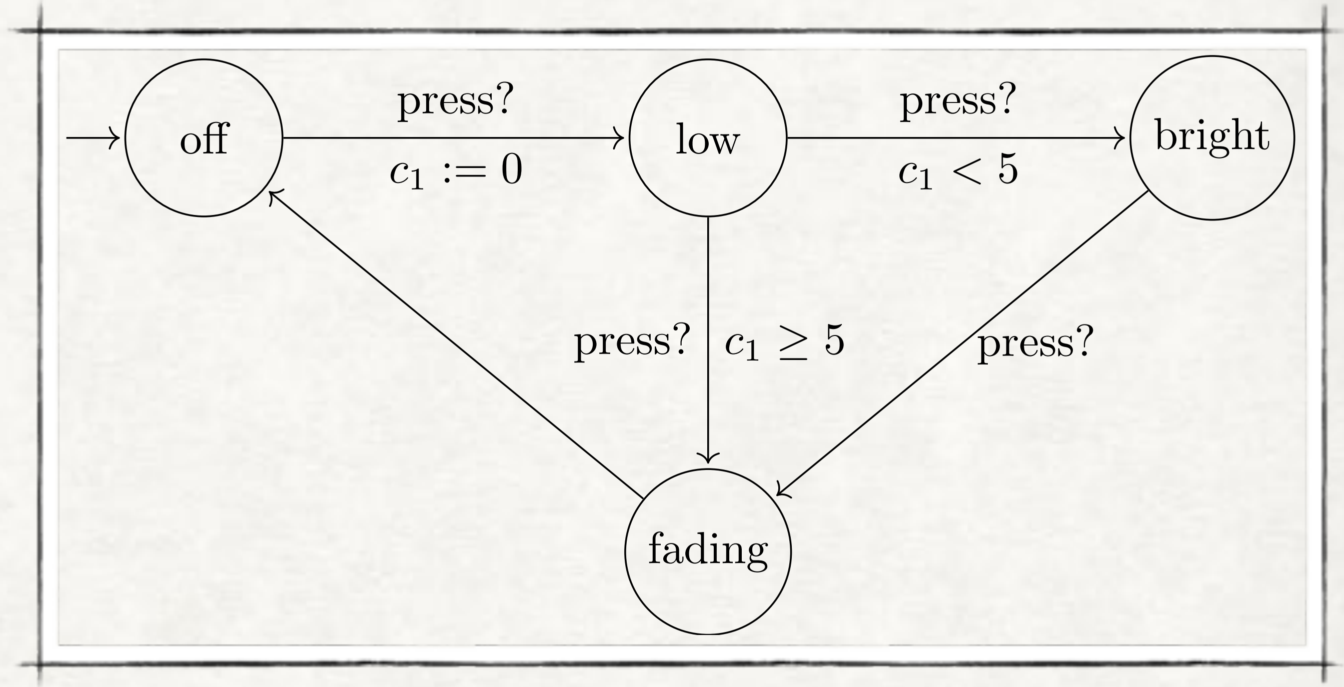
$\mathbf{E}\Diamond\ light.bright$ ✓

$\mathbf{A}\Diamond\ light.bright$ ✓

TIMED AUTOMATA

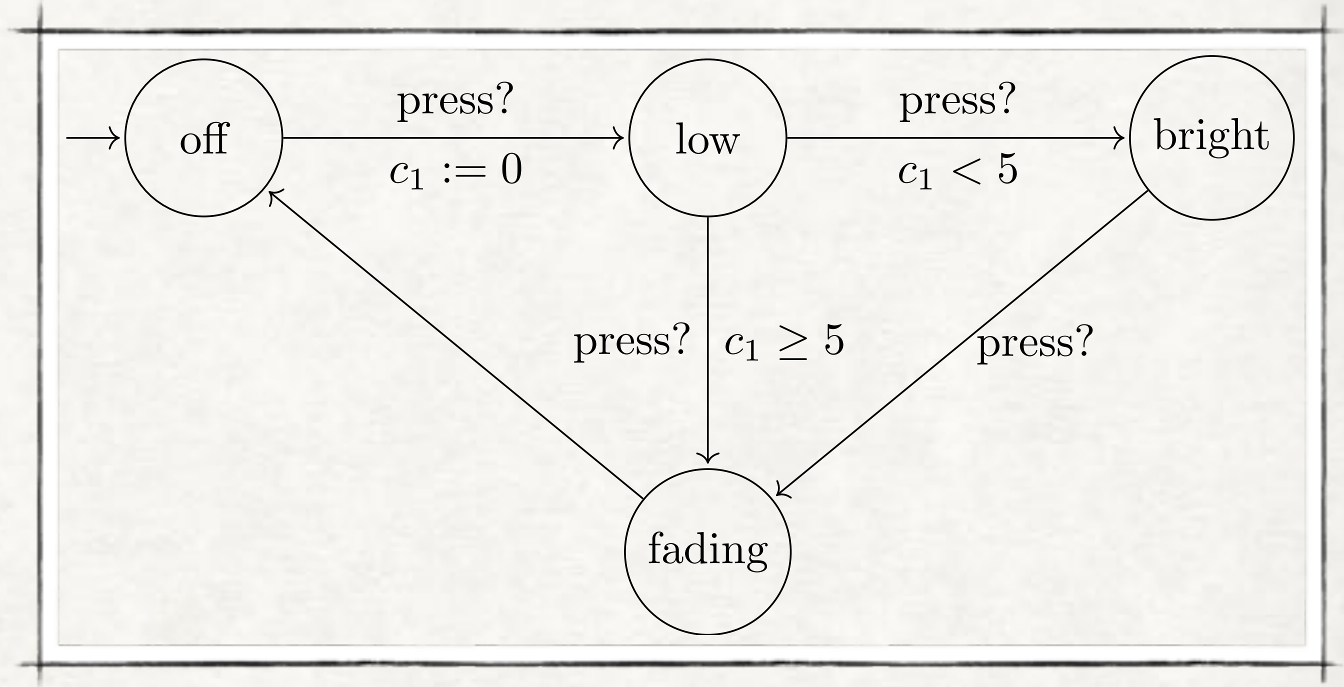
SEMANTICS

- Types of transitions:
  delay and **action**
- Clock valuations: $nat \Rightarrow real$
  → Infinite Semantics
- Clock constraints:
  $(\lambda c.\, 1) \vdash c_1 > 0 \wedge c_2 \leq 3$
  → Invariants on nodes and guards on edges



MODEL CHECKING

- Clock valuations: $nat \Rightarrow real$
  → Infinite Semantics
- Concrete states *(l, u)* to abstract states *(l, Z)*
  - node *l*
  - clock valuation *u*: $nat \Rightarrow real$
  - *Z* a set of clock valuations (zone): $(nat \Rightarrow real)\ set$
- Symbolic computation: zones as clock constraints
  → Difference Bound Matrices (DBMs)



MODEL CHECKING?

- However: number of zones (or DBMs) is infinite
  → **Approximations**: cut-off at largest entry in the automaton
- Getting these approximations right is **hard!**
- Bouyer 2003:
  - Most correctness proofs incomplete/wrong
  - Approximation is **unsound** for general TA
    → Restriction to diagonal-free TA (what we do)
    or represent zones as unions of DBMs

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
  - High feature compatibility with relevant modelling formalisms

ABSTRACT FORMALISATION

THE STARTING POINT

- ITP 2016: Isabelle/HOL formalisation of TA
- Main Results:
  - Approximations of zones are indeed sound
  - **Abstractly**, the typical reachability checking algorithm for **single** TA is sound & complete

WHAT WERE WE MISSING?

- Efficient Algorithms
  - DBMs as imperative arrays with destructive updates
  - Search algorithms with subsumption
- Expressive modelling language
  - Networks of automata with synchronisation
  - State of the art tool Uppaal accepts a C-like language

HOW DO WE GET THERE?

- Efficient Algorithms → Refinement
  - DBMs as imperative arrays with destructive updates
    Imperative Refinement Framework:
    abstract functional impl. → efficient imperative impl.
  - Search algorithms with subsumption
- Expressive modelling language
  - Networks of automata with synchronisation
    Product construction: reduction to single TA model checking
  - State of the art tool Uppaal accepts a C-like language
    Program analysis: not every input constitutes a valid (single) TA

AGENDA

- MAIN THEOREM
- REFINEMENT
- PROGRAM ANALYSIS
- PRODUCT CONSTRUCTION
- EXPERIMENTS
- FUTURE WORK

WHAT DO WE PROVE?

NO DEADLOCK

SUCCESS

HOARE TRIPLE IN IMPERATIVE-HOL

SAT/UNSAT?

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

FAILURE

INPUT IS VALID AND LIES IN THE SUPPORTED FRAGMENT?

REFINEMENT

REFINEMENT BY EXAMPLE

$$up\ M = (\lambda i\ j.$$
$$\quad \text{if } i > 0 \text{ then if } j = 0 \text{ then } \infty$$
$$\quad \text{else } \min(M\ i\ 0 + M\ 0\ j)(M\ i\ j)$$
$$\quad \text{else } M\ i\ j)$$

ASSUME THAT M IS NORMALISED

EXPLICIT PROOF

$$up_1\ M = (\lambda i\ j.$$
$$\quad \text{if } i > 0 \wedge j = 0$$
$$\quad \text{then } \infty$$
$$\quad \text{else } M\ i\ j)$$

REFINEMENT BY EXAMPLE

$$up_1\ M = (\lambda i\ j.\ \text{if } i > 0 \wedge j = 0 \text{ then } \infty \text{ else } M\ i\ j)$$

FUNCTIONAL PROGRAM

EXPLICIT PROOF

$$up_2\ M\ n = fold\ (\lambda i\ M.\ M((i,0) := \infty))\ [1\,..{<}\,n+1]\ M$$

REFINEMENT BY EXAMPLE

$$
\begin{aligned}
&up_2\ M\ n = fold \\
&\quad (\lambda i\ M.\ M((i,0) := \infty)) \\
&\quad [1\,..{<}\,n+1]\ M
\end{aligned}
$$

IMPERATIVE IMPLEMENTATION

EXTRACTED SEMI-AUTOMATICALLY

$$
\begin{aligned}
&up_3\ M\ n = imp\_for'\ 1\ (n+1) \\
&\quad (\lambda i\ M.\ mtx\_set\ (n+1)\ M\ (i,0)\ \infty) \\
&\quad M
\end{aligned}
$$

IMPERATIVE REFINEMENT

WITH THE IMPERATIVE REFINEMENT FRAMEWORK

- Semi-automatically synthesise imp. implementation
  - Parametricity (‘truly polymorphic functions ignore the type’)
  - Separation logic with some automated frame inference
- Proved automatically:

$$\boxed{(up_3, up_2) \in mtx\_assn^d * nat\_assn^k \rightarrow mtx\_assn}$$

CYCLICITY CHECKER

FOR LIVENESS PROPERTIES

'PSEUDOCODE'

ANY RELATION/TS/ GRAPH

SETS

SUBSUMPTION

```
dfs P = do {
    (P, ST, r) ← rec_T (λdfs (P, ST, v) .
        do {
            if ∃v′ ∈ set ST. v′ ≼ v then return (P, ST, True)
            else do {
                if ∃v′ ∈ P. v ≼ v′ then return (P, ST, False)
                else do {
                    let ST = v · ST ;
                    (P, ST′, r) ←
                        foreach {v′ | v → v′} (λ(_, _, b). ¬ b)
                            (λv′ (P, ST, _). dfs (P, ST, v′))
                            (P, ST, False) ;
                    assert (ST′ = ST) ;
                    return (insert v P, tl ST′, r)
                }
            }
        }) (P, [], a_0) ;
    return (r, P)}
```

CYCLICITY CHECKER

LAYERED REFINEMENT

- Non-determinism monad ('give me any x such that …')
- Verification Condition Generator
- Final data structure resembles Uppaal's unified PW list
- Main theorems:

$$\begin{array}{c} \mathit{dfs}\ P \leq \mathsf{SPEC}\ (\lambda(r, P').\ (r \Longrightarrow (\exists x.\ a_0 \rightarrow^* x \wedge x \rightarrow^+ x)) \\ \wedge(\neg\, r \Longrightarrow \neg\,(\exists x.\ a_0 \rightarrow^* x \wedge x \rightarrow^+ x) \wedge \mathit{liveness\_compatible}\ P')) \\ \text{if } \mathit{liveness\_compatible}\ P \end{array}$$

$$(\mathit{dfs\_map}, \mathit{dfs}) \in \mathit{map\_set\_rel} \rightarrow \mathit{Id} \times_r \mathit{map\_set\_rel}$$

PROGRAM ANALYSIS & PRODUCT CONSTRUCTION

PROGRAM ANALYSIS

TO ENSURE THE INPUT IS VALID

- Input: Uppaal bytecode (interpreted with finite fuel)
  Assembler-style language for updates and guards

- Main property: Successful executions only induce conjunctive clock constraints ($c_1 > 0 \wedge c_2 < 3$ but not $c_1 > 0 \vee c_2 < 3$)

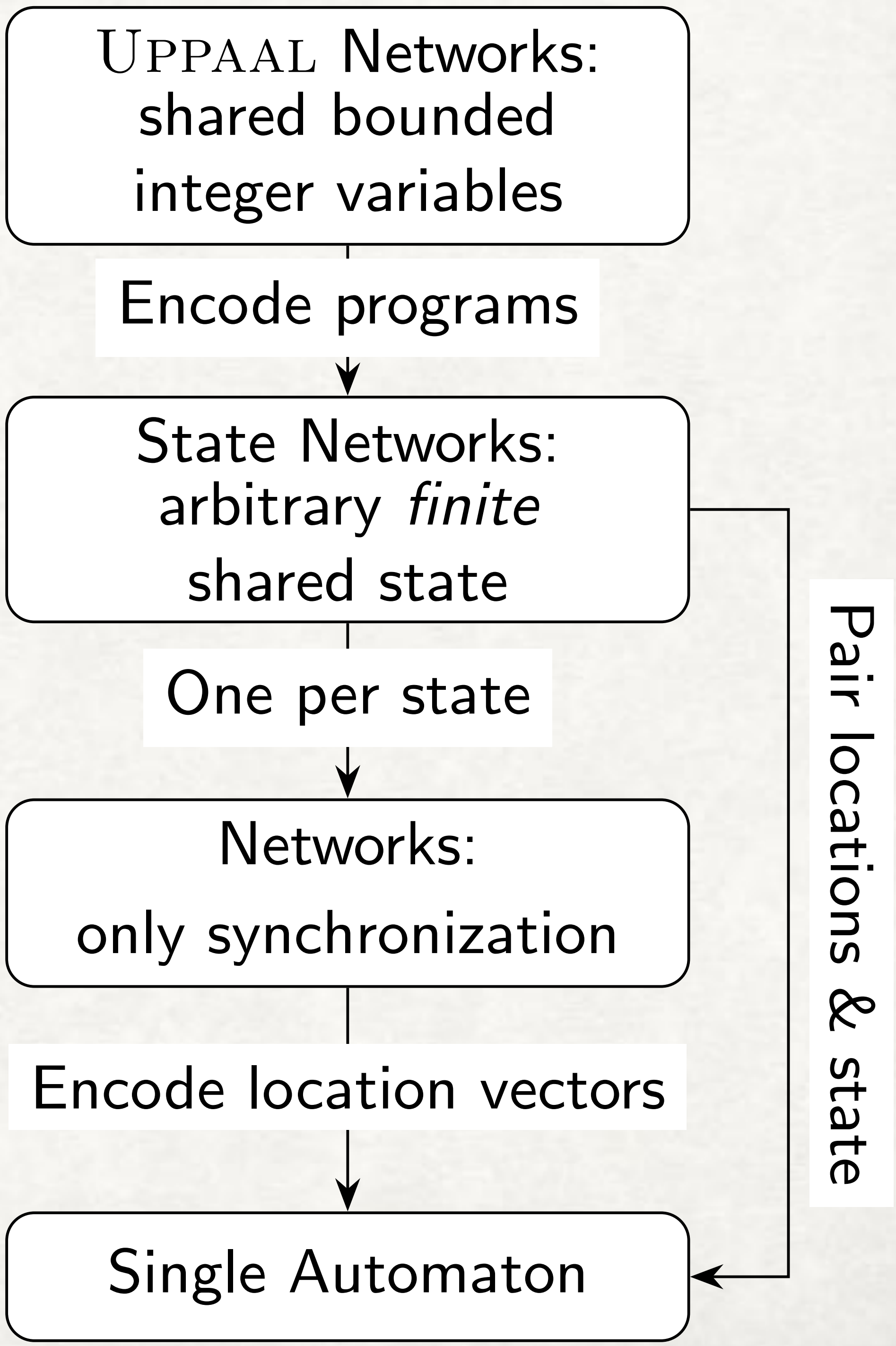
- Very simplistic analysis:

  - Approximate set of reachable instructions for a given guard

  - Check that clock expressions only occur in a ‘conjunction block’

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**



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

FUTURE

- Can we find bugs in actual model checkers?
- 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,**
  (more complex hybrid systems?)
- > 50000 lines of formalisation vs 5403 lines of SML checker

THANK YOU!
QUESTIONS?

wimmers.github.io/munta