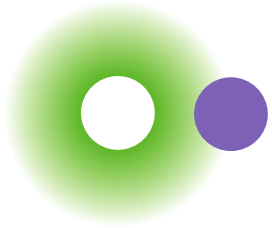
From propositional to first-order monitoring

Jan-Christoph Küster

Joint work with Andreas Bauer and Gil Vegliach



NICTA

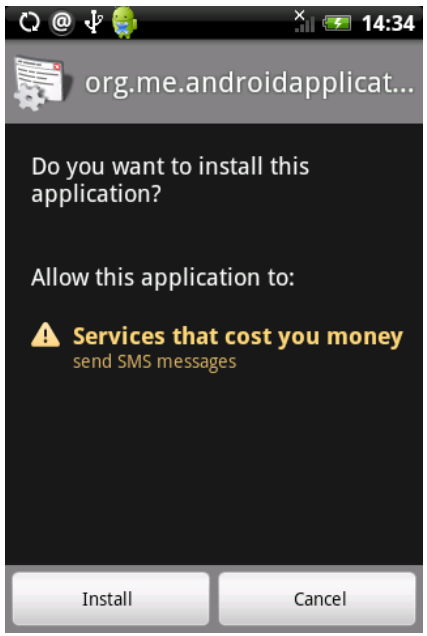
NICTA Funding and Supporting Members and Partners

Australian Government
Department of Broadband, Communications and the Digital Economy
Australian Research Council

Australian National University
UNSW THE UNIVERSITY OF NEW SOUTH WALES
NSW Government Trade & Investment
Victoria The Place To Be
The University of Melbourne
The University of Sydney
Queensland Government
Griffith University
QUT
The University Of Queensland AUSTRALIA

Why first-order monitoring?

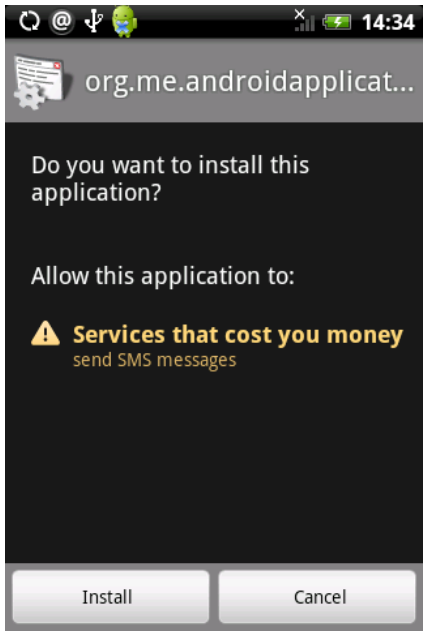
Checking security properites of **systems with data**, e.g., smartphones.



- Trojan-SMS.AndroidOS.FakePlayer.A
- While displaying *“Wait, sought access to video library. . . ”*, app sends string *“798657”* in SMS to Russian premium numbers.

Why first-order monitoring?

Checking security properites of **systems with data**, e.g., smartphones.



- Trojan-SMS.AndroidOS.FakePlayer.A
- While displaying *"Wait, sought access to video library. . . "*, app sends string *"798657"* in SMS to Russian premium numbers.

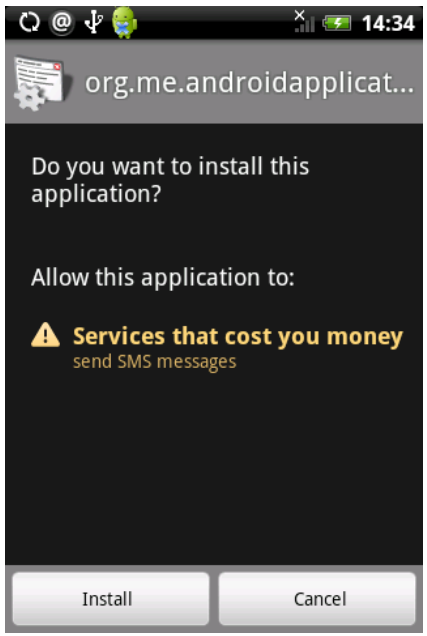
Security property (syntax)

*"Don't send an SMS to **anyone** not in my contact book"*:

$$\mathbf{G} \forall x : sms.\ contact(x)$$

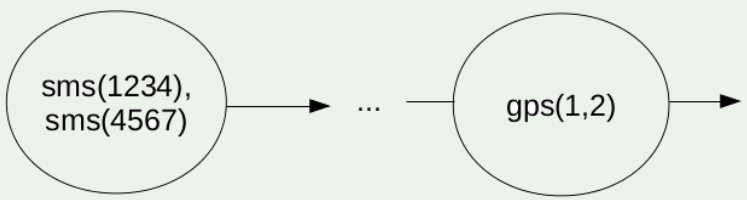
Why first-order monitoring?

Checking security properites of **systems with data**, e.g., smartphones.



- Trojan-SMS.AndroidOS.FakePlayer.A
- While displaying *"Wait, sought access to video library. . . "*, app sends string *"798657"* in SMS to Russian premium numbers.

Trace (semantics)



Spawning automaton

$$\mathcal{L}(\mathcal{A}_\varphi) = \mathcal{L}(\varphi)$$

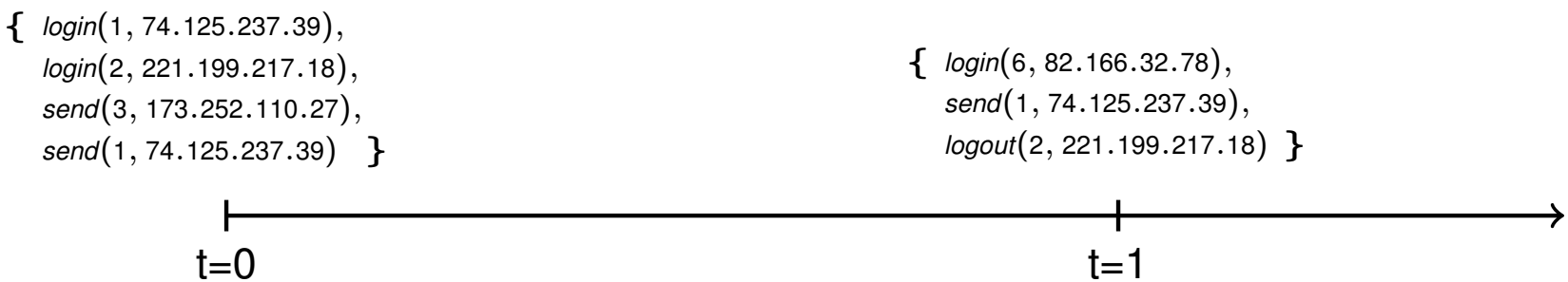
for $\varphi \in \mathrm{LTL}^{\mathrm{FO}}$

Spawning automaton

Example

Once **user** *u* has **logged in** to the system from **IP-address** *ip*, she must **not send** anything from an IP-address **other than ip** until **logged out**.

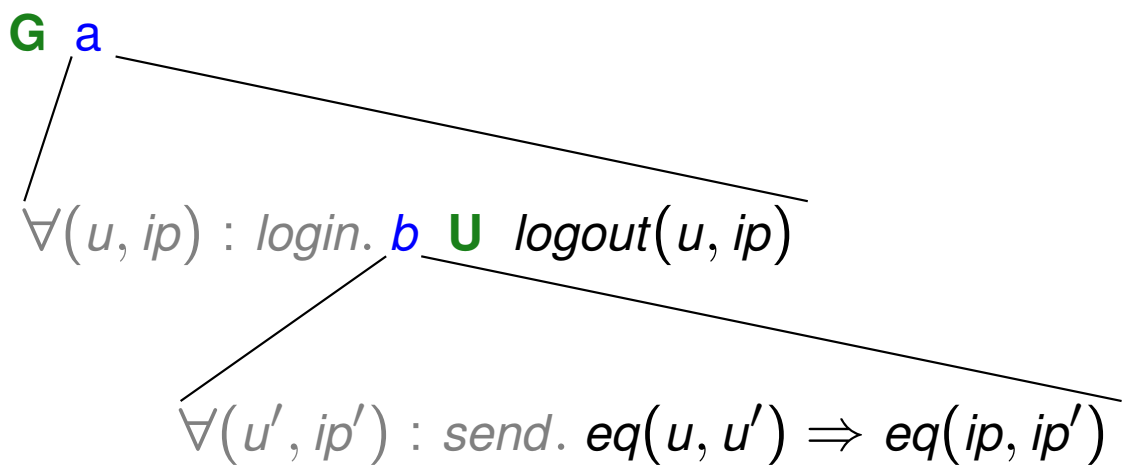
$$\mathbf{G}\ \forall(u, ip) : login.\ \forall(u', ip') : send.\ \big(eq(u, u') \Rightarrow eq(ip, ip')\big)\ \mathbf{U}\ logout(u, ip)$$



Spawning automaton

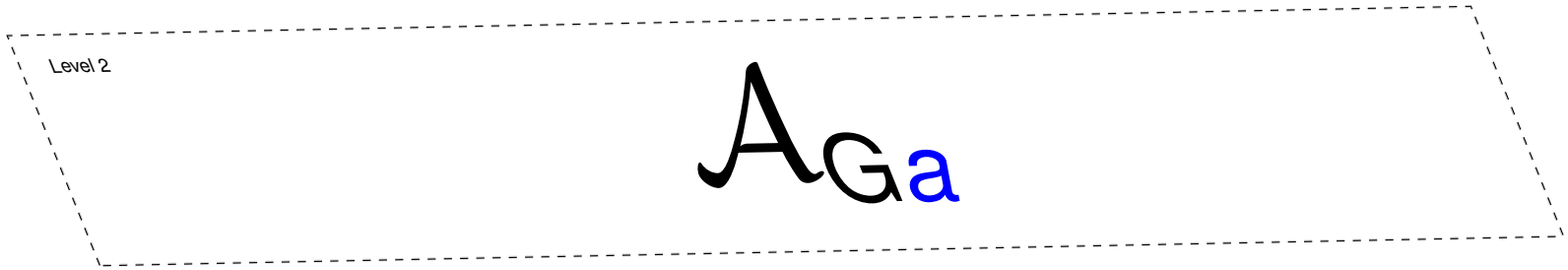
Example

Once **user** *u* has **logged in** to the system from **IP-address** *ip*, she must **not send** anything from an IP-address **other than ip** until **logged out**.

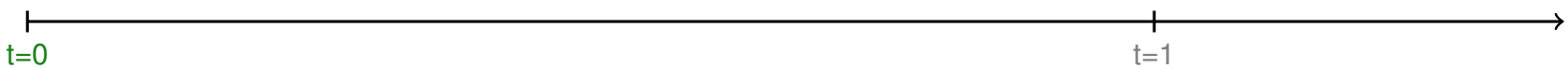


Spawning automaton

Example

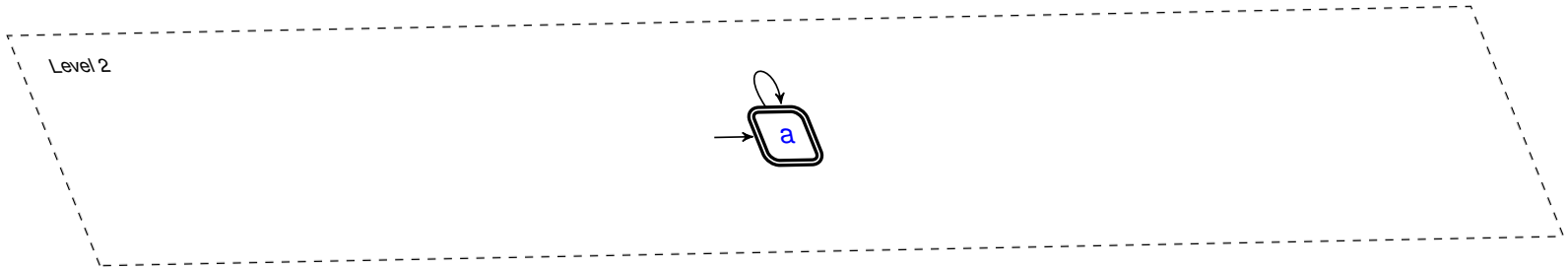


{ *login*(1, 74.125.237.39),
login(2, 221.199.217.18),
send(3, 173.252.110.27),
send(1, 74.125.237.39) }

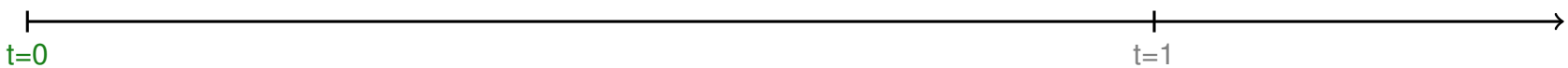


Spawning automaton

Example

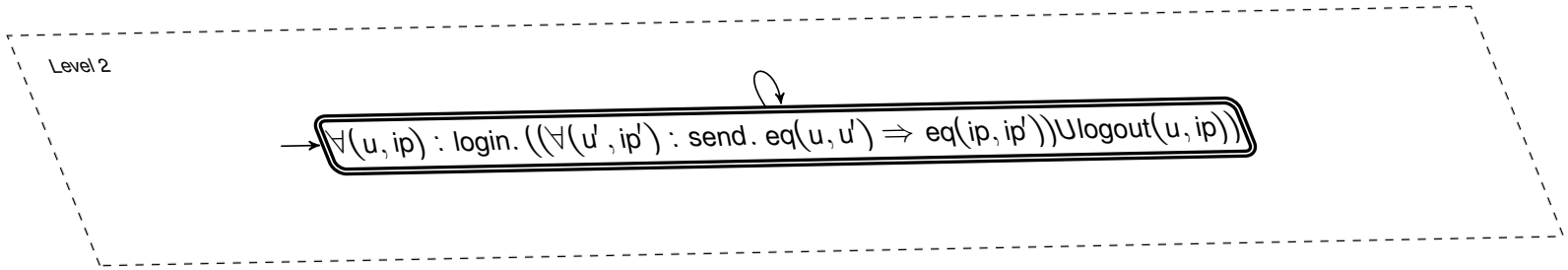


$\{$ *login*(1, 74.125.237.39),
login(2, 221.199.217.18),
send(3, 173.252.110.27),
send(1, 74.125.237.39) $\}$

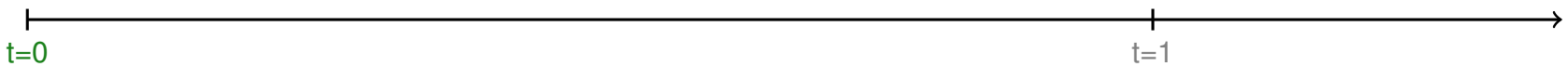


Spawning automaton

Example

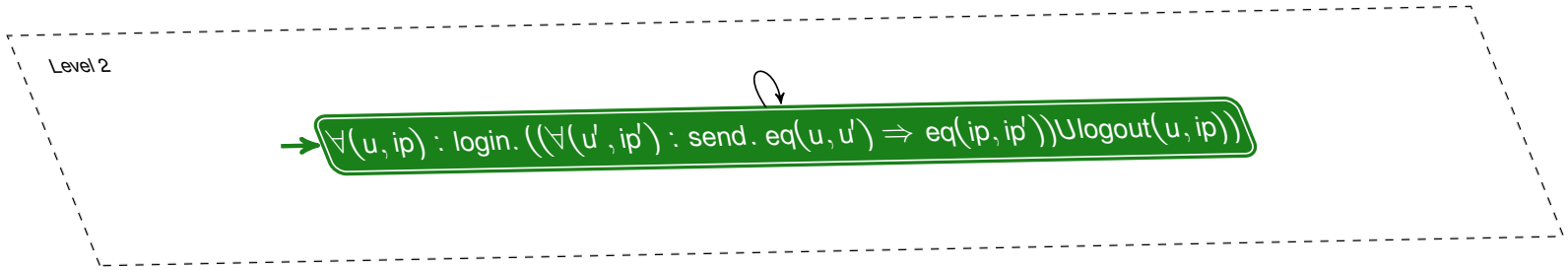


$\{$ *login*$(1, 74.125.237.39)$,
login$(2, 221.199.217.18)$,
send$(3, 173.252.110.27)$,
send$(1, 74.125.237.39)$ $\}$

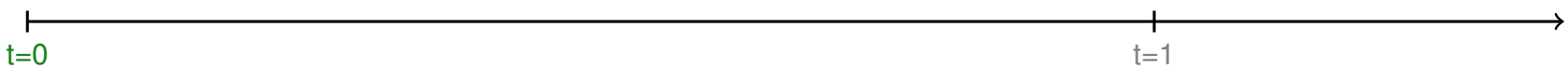


Spawning automaton

Example

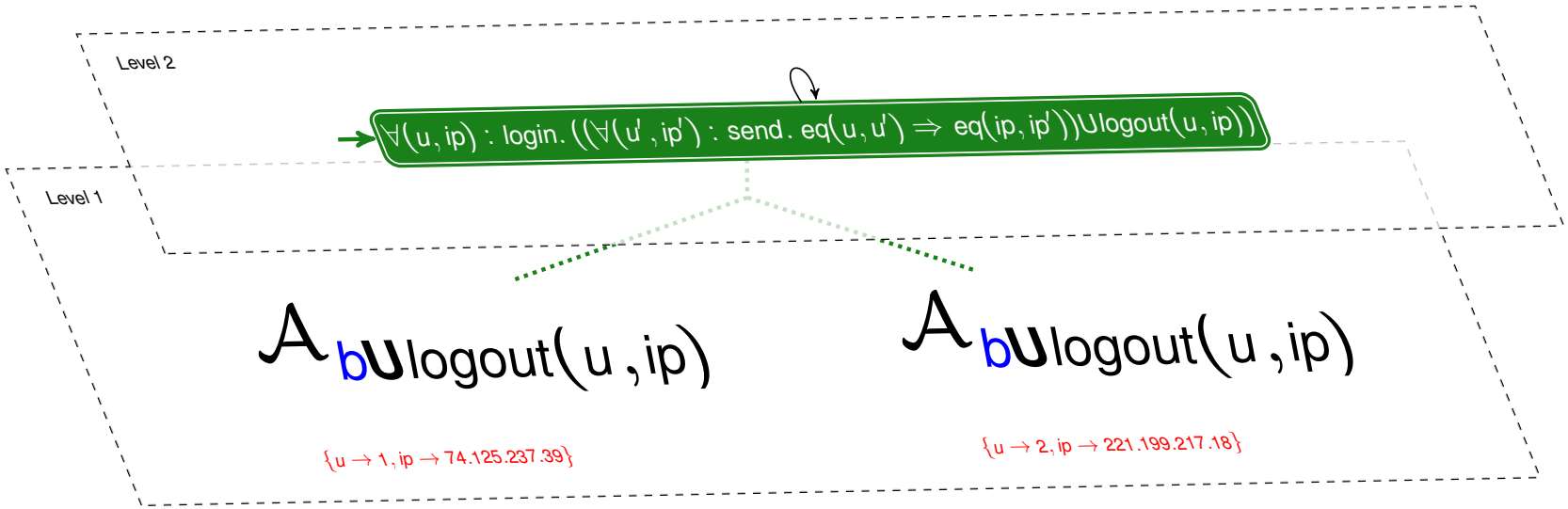


$\{\ \mathit{login}(1, 74.125.237.39),$
$\mathit{login}(2, 221.199.217.18),$
$\mathit{send}(3, 173.252.110.27),$
$\mathit{send}(1, 74.125.237.39)\ \}$

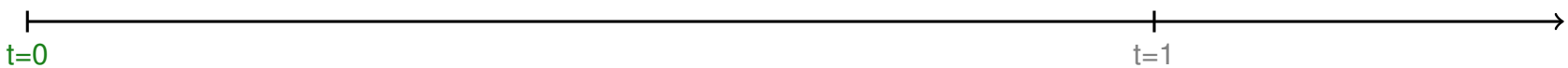


Spawning automaton

Example

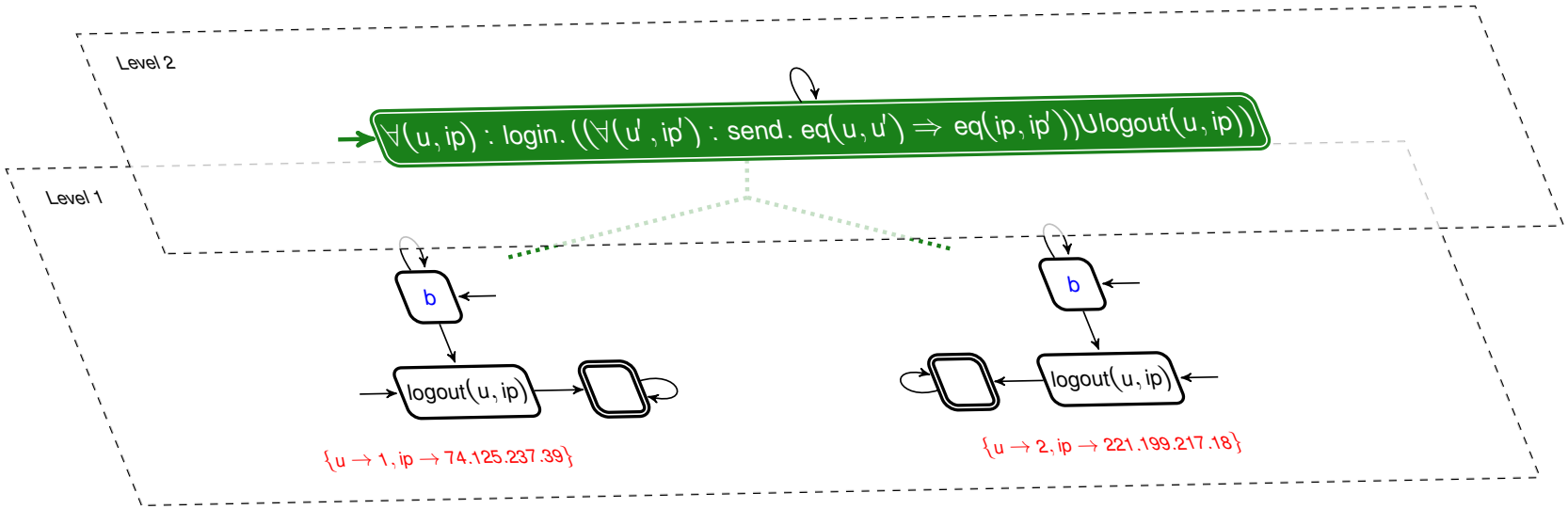


{ *login*(1, 74.125.237.39),
login(2, 221.199.217.18),
send(3, 173.252.110.27),
send(1, 74.125.237.39) }

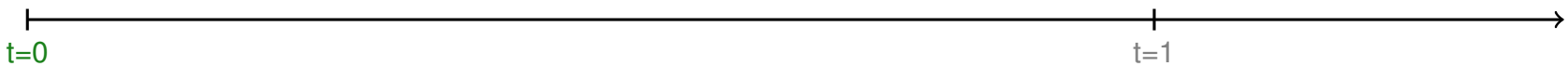


Spawning automaton

Example

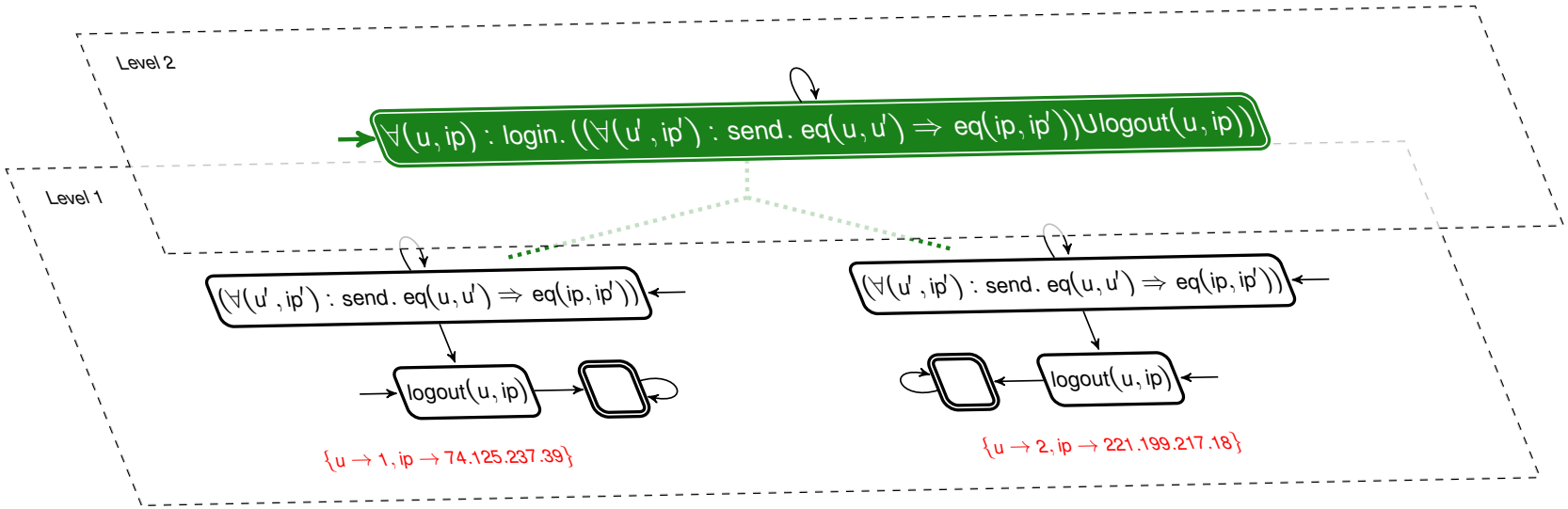


{ *login*(1, 74.125.237.39),
login(2, 221.199.217.18),
send(3, 173.252.110.27),
send(1, 74.125.237.39) }

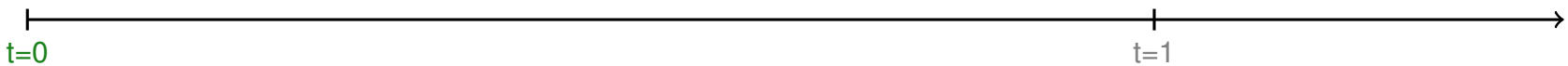


Spawning automaton

Example

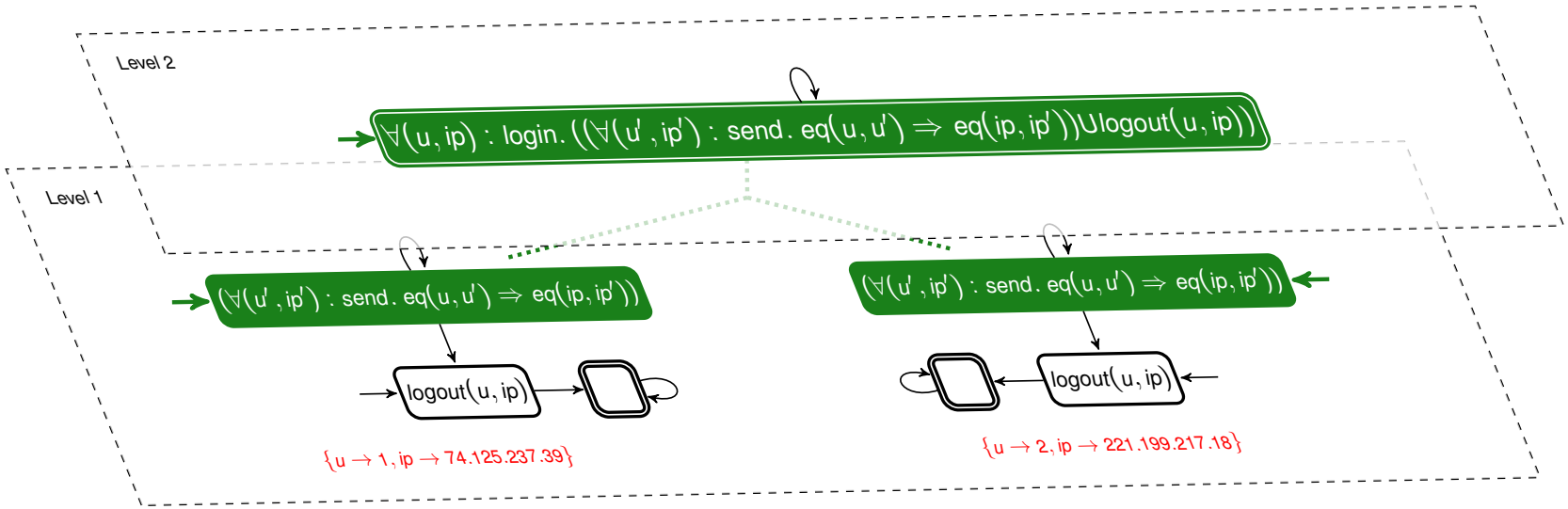


{ *login*(1, 74.125.237.39),
login(2, 221.199.217.18),
send(3, 173.252.110.27),
send(1, 74.125.237.39) }

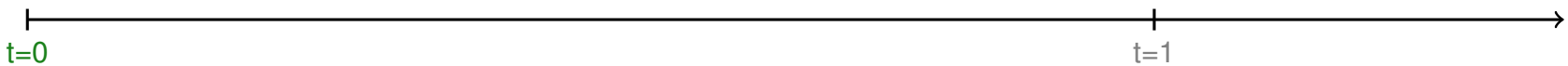


Spawning automaton

Example

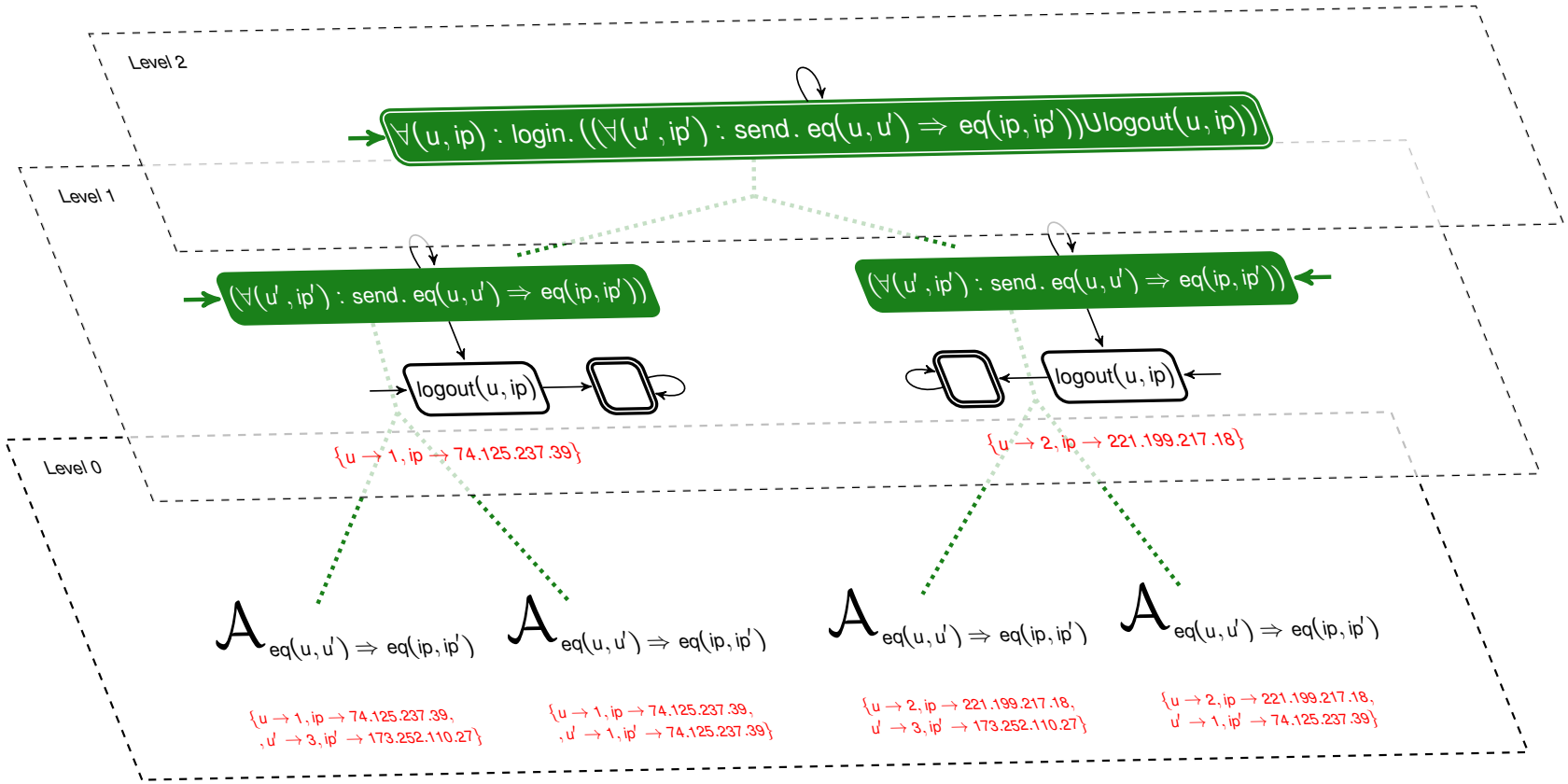


$\{$ *login*(1, 74.125.237.39),
login(2, 221.199.217.18),
send(3, 173.252.110.27),
send(1, 74.125.237.39) $\}$



Spawning automaton

Example

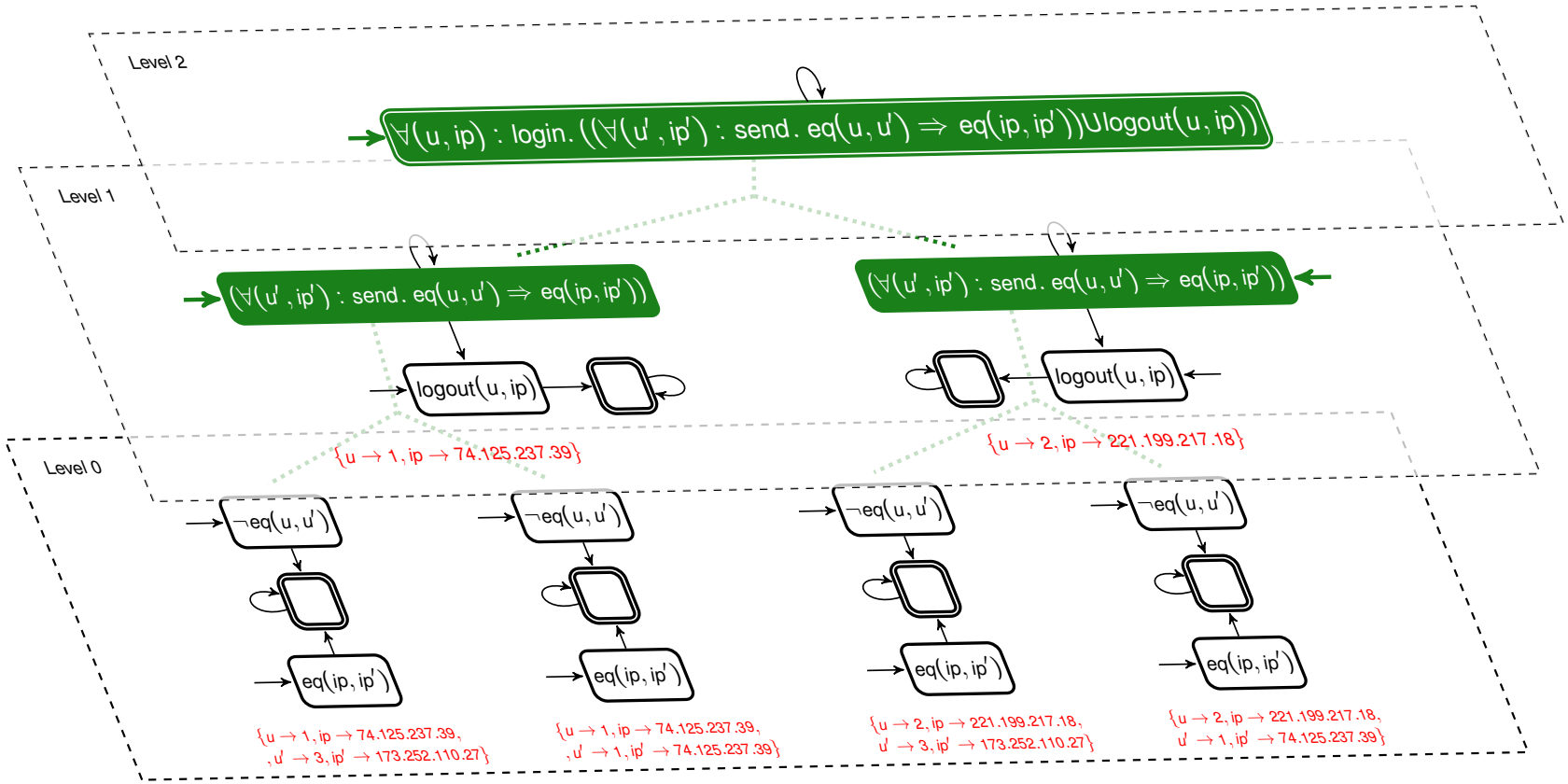


$\{ login(1, 74.125.237.39), login(2, 221.199.217.18), send(3, 173.252.110.27), send(1, 74.125.237.39) \}$

t=0 t=1

Spawning automaton

Example

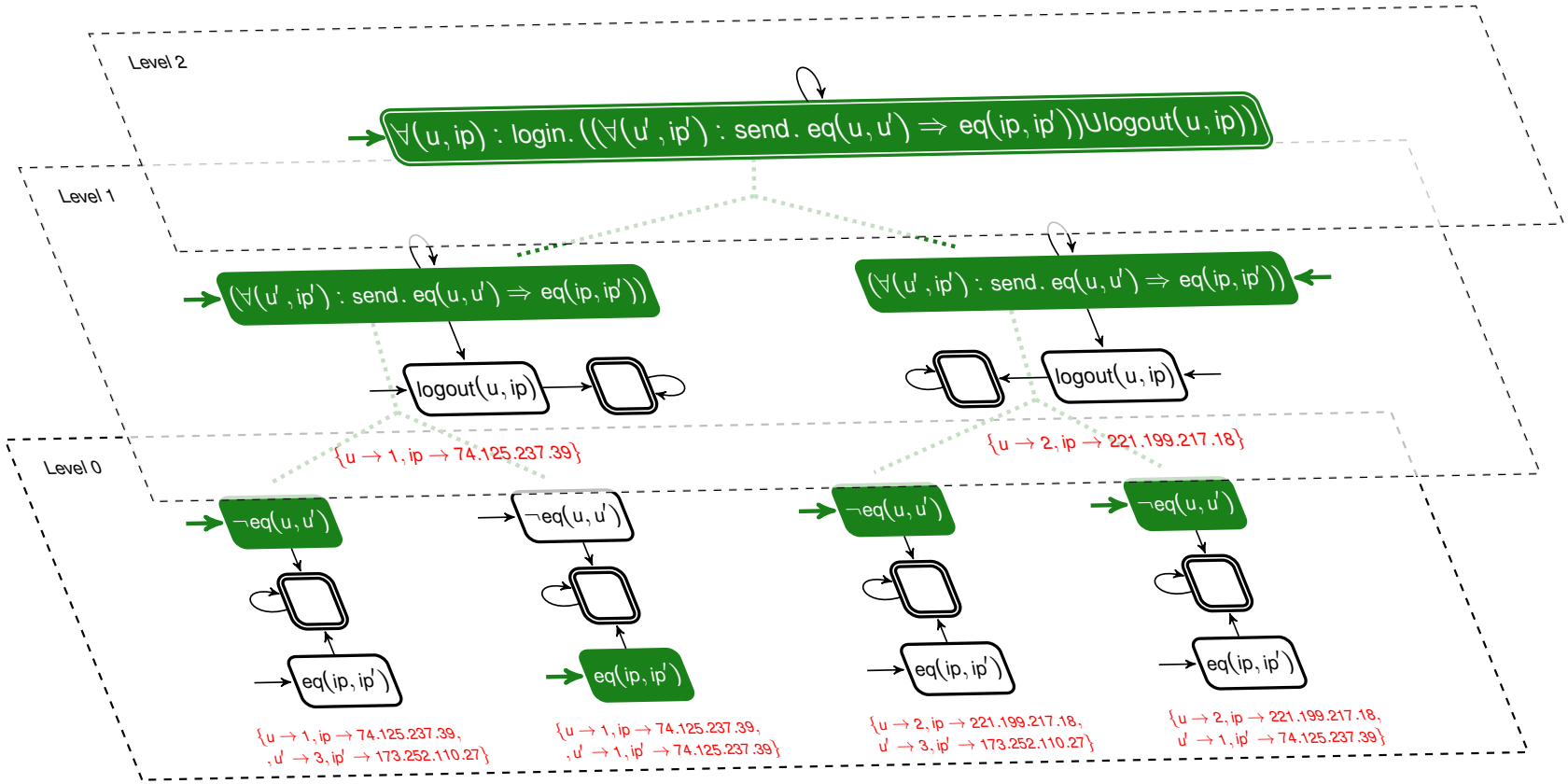


$\{ login(1, 74.125.237.39), login(2, 221.199.217.18), send(3, 173.252.110.27), send(1, 74.125.237.39) \}$

t=0 t=1

Spawning automaton

Example

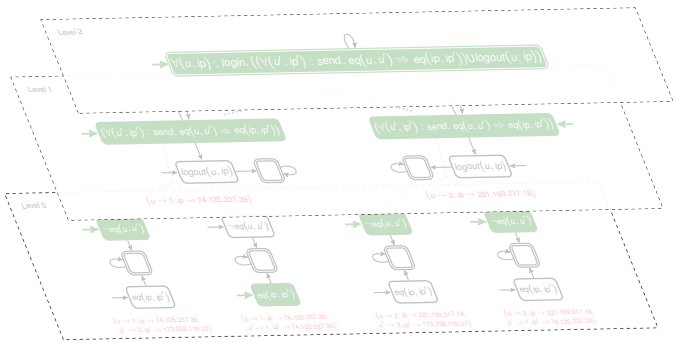


{ $login(1, 74.125.237.39)$,
$login(2, 221.199.217.18)$,
$send(3, 173.252.110.27)$,
$send(1, 74.125.237.39)$ }

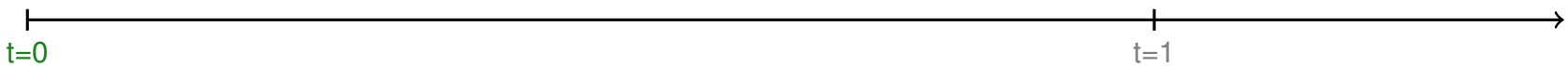
t=0 t=1

Spawning automaton

Example

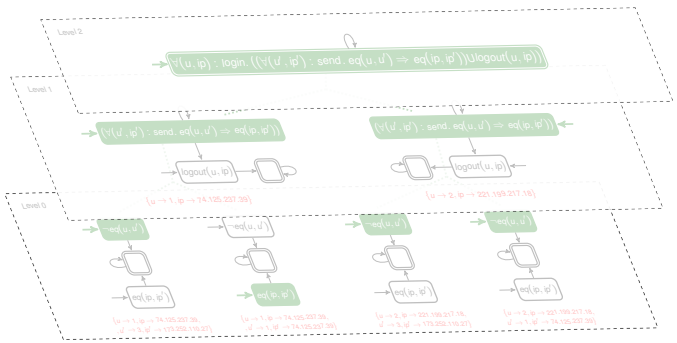


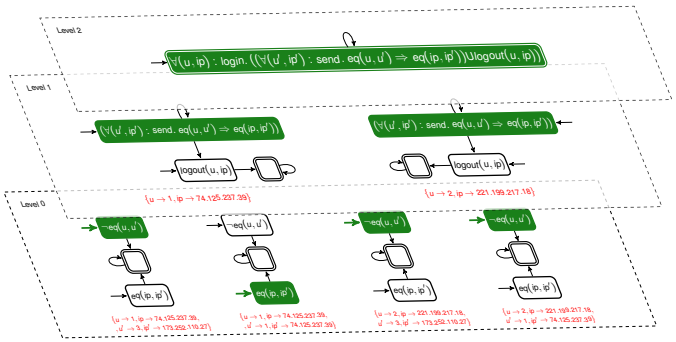
$\{\ login(1, 74.125.237.39),$
$login(2, 221.199.217.18),$
$send(3, 173.252.110.27),$
$send(1, 74.125.237.39)\ \}$



Spawning automaton

Example





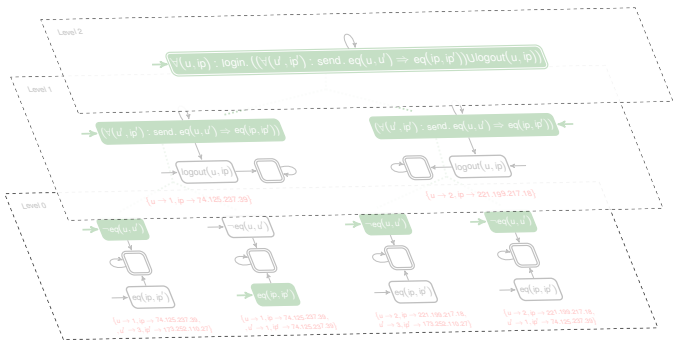
{ *login*(1, 74.125.237.39),
login(2, 221.199.217.18),
send(3, 173.252.110.27),
send(1, 74.125.237.39) }

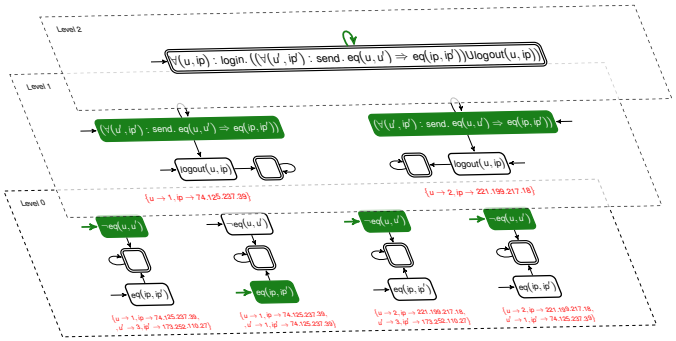
{ *login*(6, 82.166.32.78),
logout(2, 221.199.217.18),
send(1, 74.125.237.39) }

t=0 — t=1

Spawning automaton

Example





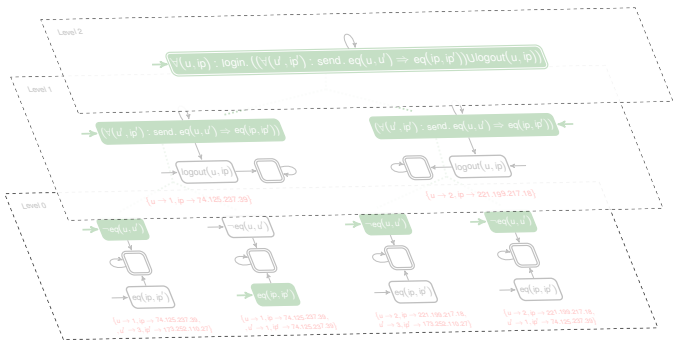
{ *login*(1, 74.125.237.39),
login(2, 221.199.217.18),
send(3, 173.252.110.27),
send(1, 74.125.237.39) }

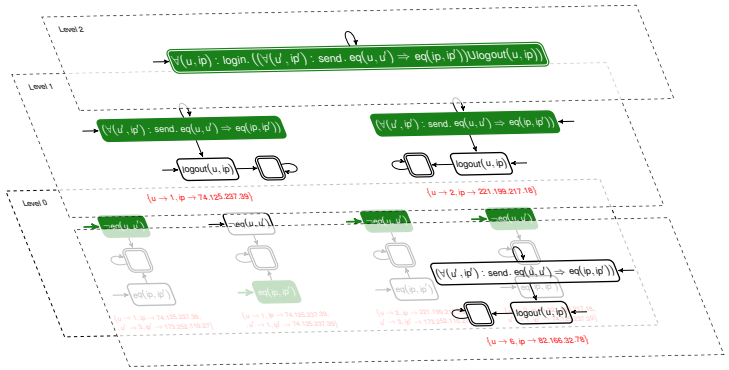
{ *login*(6, 82.166.32.78),
logout(2, 221.199.217.18),
send(1, 74.125.237.39) }

t=0 t=1

Spawning automaton

Example





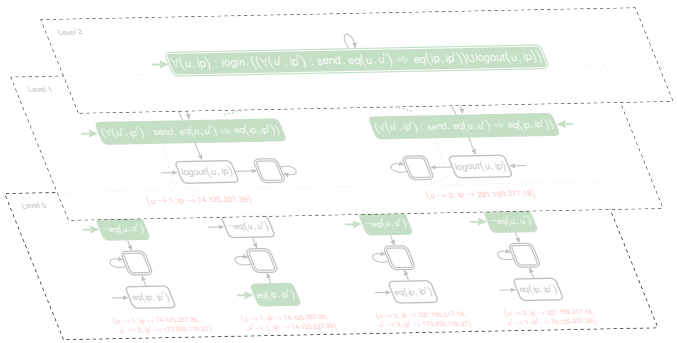
{ *login*(1, 74.125.237.39),
login(2, 221.199.217.18),
send(3, 173.252.110.27),
send(1, 74.125.237.39) }

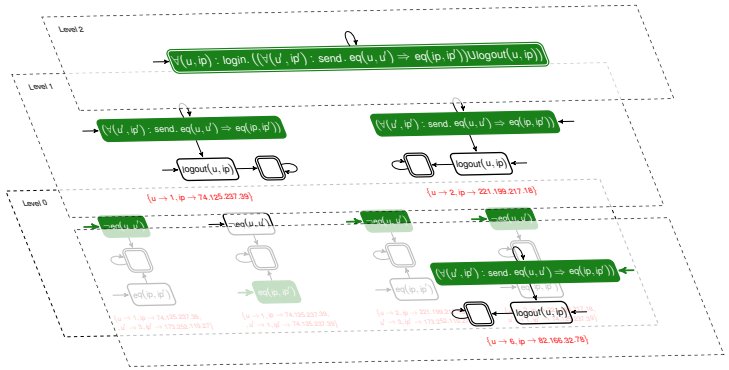
{ *login*(6, 82.166.32.78),
logout(2, 221.199.217.18),
send(1, 74.125.237.39) }

t=0 — t=1

Spawning automaton

Example





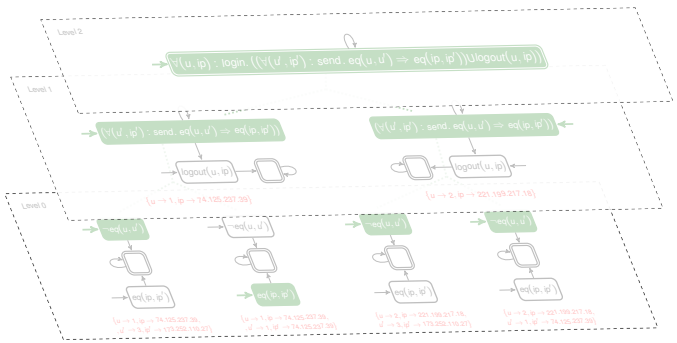
{ *login*(1, 74.125.237.39),
login(2, 221.199.217.18),
send(3, 173.252.110.27),
send(1, 74.125.237.39) }

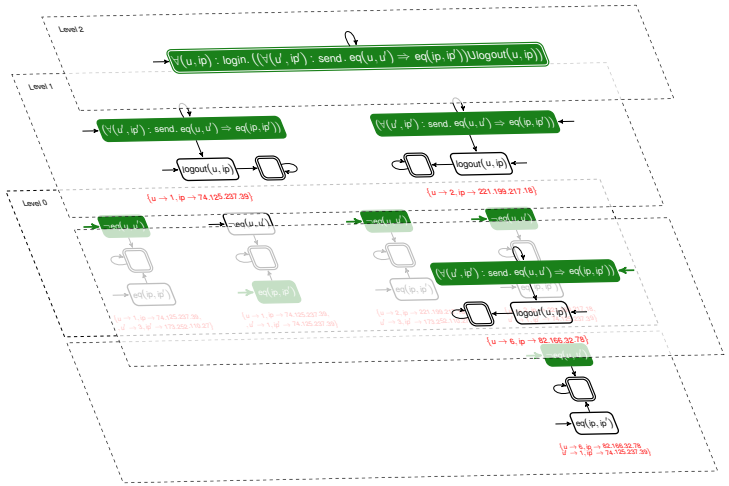
{ *login*(6, 82.166.32.78),
logout(2, 221.199.217.18),
send(1, 74.125.237.39) }

t=0 t=1

Spawning automaton

Example





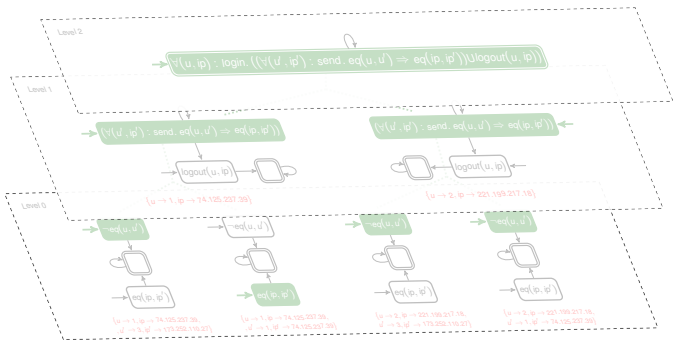
{ *login*(1, 74.125.237.39),
login(2, 221.199.217.18),
send(3, 173.252.110.27),
send(1, 74.125.237.39) }

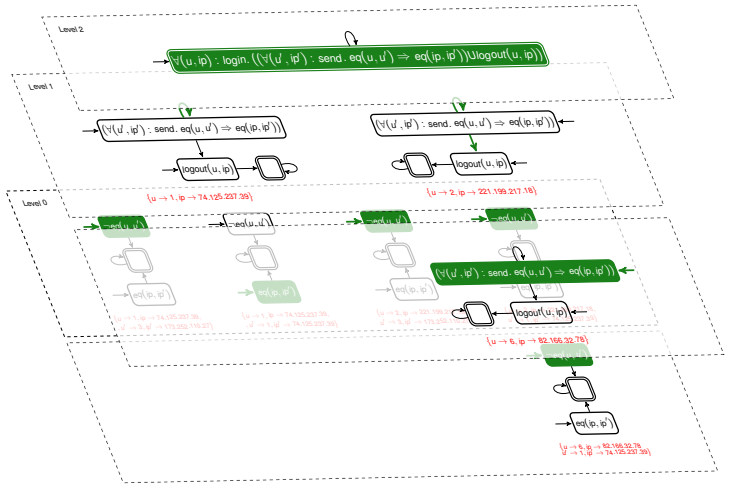
{ *login*(6, 82.166.32.78),
logout(2, 221.199.217.18),
send(1, 74.125.237.39) }

t=0 t=1

Spawning automaton

Example





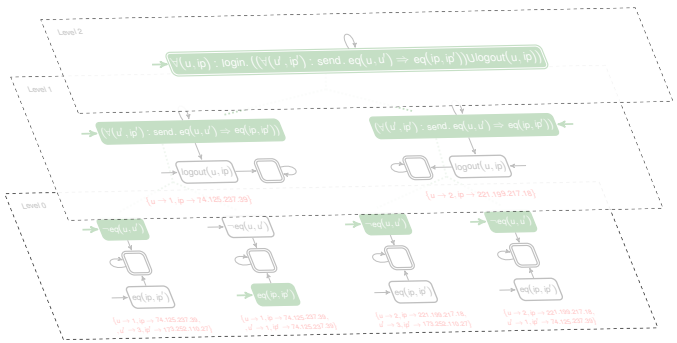
{ *login*(1, 74.125.237.39),
login(2, 221.199.217.18),
send(3, 173.252.110.27),
send(1, 74.125.237.39) }

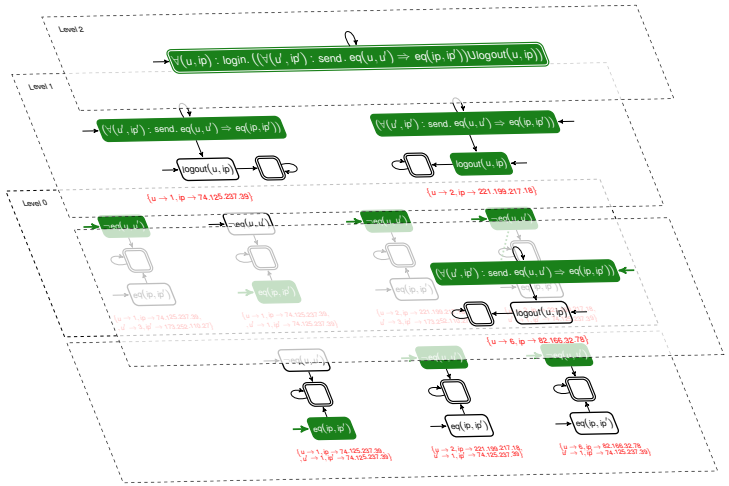
{ *login*(6, 82.166.32.78),
logout(2, 221.199.217.18),
send(1, 74.125.237.39) }

t=0 — t=1

Spawning automaton

Example





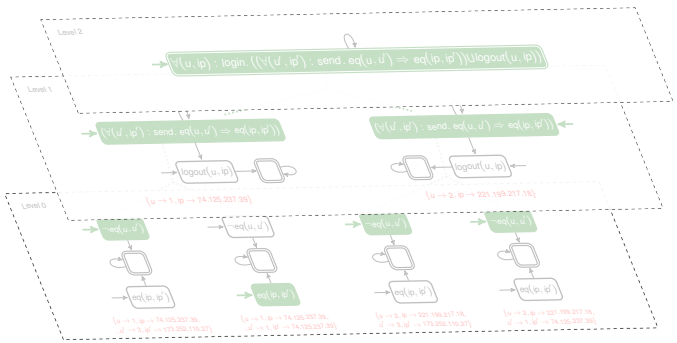
{ *login*(1, 74.125.237.39),
login(2, 221.199.217.18),
send(3, 173.252.110.27),
send(1, 74.125.237.39) }

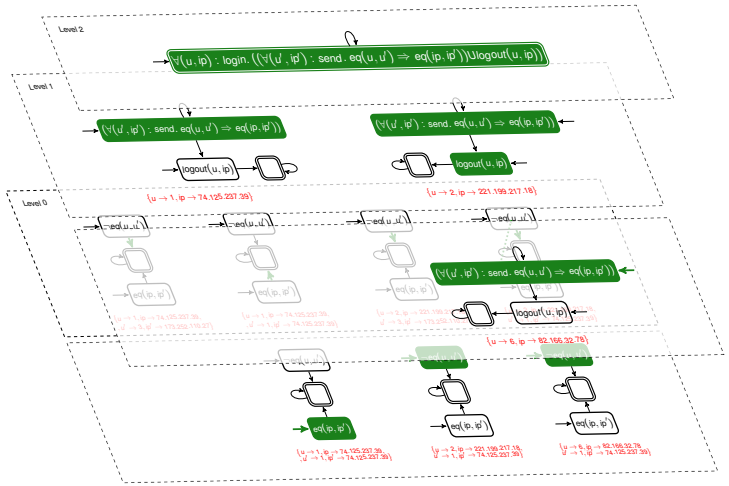
{ *login*(6, 82.166.32.78),
logout(2, 221.199.217.18),
send(1, 74.125.237.39) }

t=0 — t=1

Spawning automaton

Example





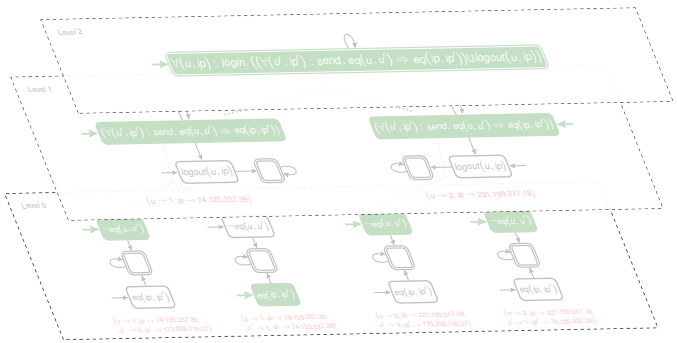
{ *login*(1, 74.125.237.39),
login(2, 221.199.217.18),
send(3, 173.252.110.27),
send(1, 74.125.237.39) }

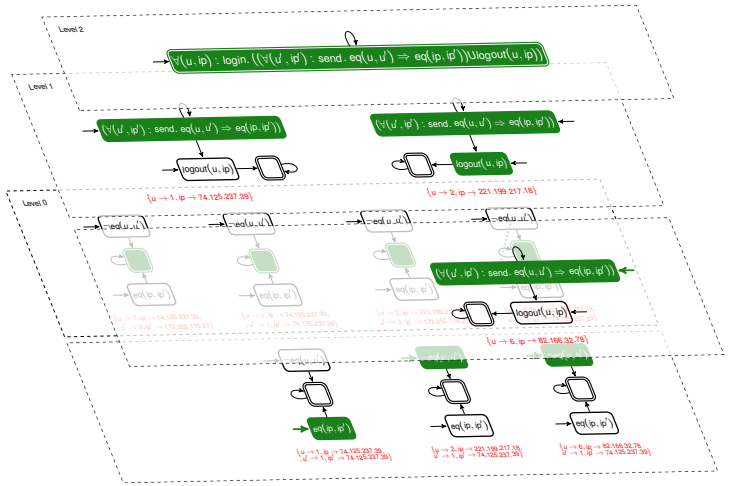
{ *login*(6, 82.166.32.78),
logout(2, 221.199.217.18),
send(1, 74.125.237.39) }

t=0 — t=1

Spawning automaton

Example





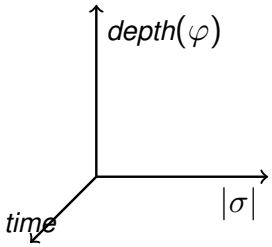
{ *login*(1, 74.125.237.39),
login(2, 221.199.217.18),
send(3, 173.252.110.27),
send(1, 74.125.237.39) }

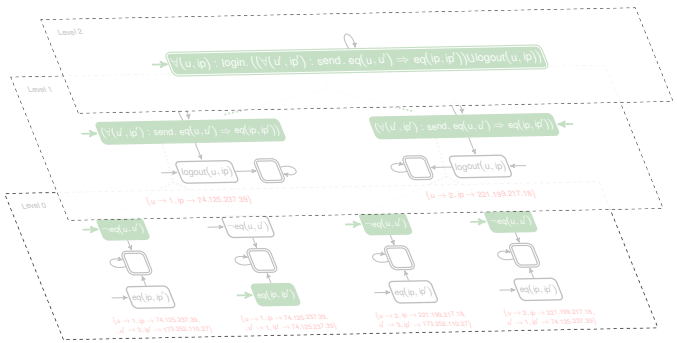
{ *login*(6, 82.166.32.78),
logout(2, 221.199.217.18),
send(1, 74.125.237.39) }

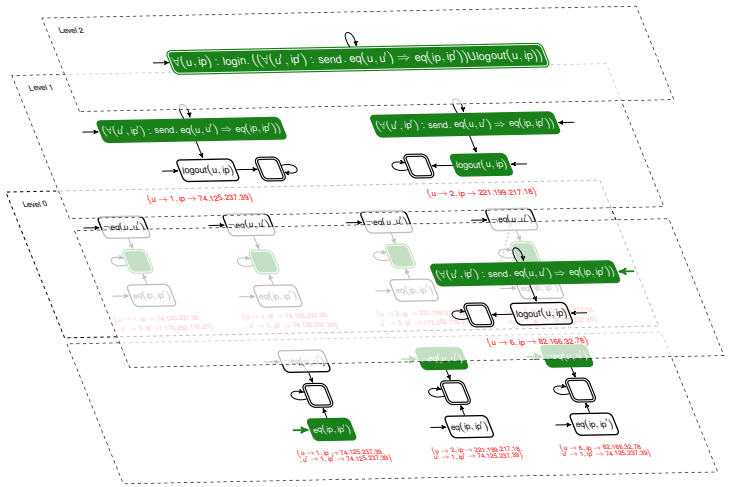
t=0 — t=1

Spawning automaton

Example







{ *login*(1, 74.125.237.39),
login(2, 221.199.217.18),
send(3, 173.252.110.27),
send(1, 74.125.237.39) }

{ *login*(6, 82.166.32.78),
logout(2, 221.199.217.18),
send(1, 74.125.237.39) }

t=0 — t=1

Spawning automaton

Definition

$\mathcal{A} = (\Sigma, Q, Q_0, \delta_{\rightarrow}, \mathcal{F}, \quad)$

- Σ an alphabet
- Q a finite set of states.
- $Q_0 \subseteq Q$ a set of distinguished initial states.
- $\delta_{\rightarrow} : Q \times \Sigma \longrightarrow 2^Q$ a transition relation.
- $\mathcal{F} = \{F_1, \ldots, F_n \mid F_i \subseteq Q\}$ an acceptance condition.

Spawning automaton

Definition

$\mathcal{A} = (\Sigma, Q, Q_0, \delta_{\rightarrow}, \mathcal{F}, \ell, \delta_{\downarrow})$

- Σ an alphabet (countable set).
- Q a finite set of states.
- $Q_0 \subseteq Q$ a set of distinguished initial states.
- $\delta_{\rightarrow} : Q \times \Sigma \longrightarrow 2^Q$ a transition relation.
- $\mathcal{F} = \{F_1, \ldots, F_n \mid F_i \subseteq Q\}$ an acceptance condition.
- $\ell \in \mathbb{N}_0$ is the level of $\mathcal{A}$.
- $\delta_{\downarrow} : Q \times \Sigma \longrightarrow \mathcal{B}^+(\mathcal{A}^{<\ell})$ a spawning function.

Spawning automaton

Definition

$\mathcal{A} = (\Sigma, Q, Q_0, \delta_{\to}, \mathcal{F}, \ell, \delta_{\downarrow})$

- Σ an alphabet (countable set).
- Q a finite set of states.
- $Q_0 \subseteq Q$ a set of distinguished initial states.
- $\delta_{\to} : Q \times \Sigma \longrightarrow 2^Q$ a transition relation.
- $\mathcal{F} = \{F_1, \ldots, F_n \mid F_i \subseteq Q\}$ an acceptance condition.
- $\ell \in \mathbb{N}_0$ is the level of $\mathcal{A}$.
- $\delta_{\downarrow} : Q \times \Sigma \longrightarrow \mathcal{B}^+(\mathcal{A}^{<\ell})$ a spawning function.

Constructing $\delta_{\downarrow}^{\varphi}$ for $\varphi \in \mathrm{LTL}^{\mathrm{FO}}$

$$\delta_{\downarrow}(q, (\mathfrak{A}, \sigma)) = \left(\bigwedge_{\forall \mathbf{x}:p.\psi \in q} \left(\bigwedge_{(p,\mathbf{d}) \in \sigma} \mathcal{A}_{\psi, v(\mathbf{x},\mathbf{d})} \right) \right) \wedge \left(\bigwedge_{\neg \forall \mathbf{x}:p.\psi \in q} \left(\bigvee_{(p,\mathbf{d}) \in \sigma} \mathcal{A}_{\neg\psi, v(\mathbf{x},\mathbf{d})} \right) \right)$$

Spawning automaton

When is a run accepting?

Local acceptance (= Büchi acceptance)

$\mathcal{A}$ accepts exactly those runs in which the set of infinitely often occurring states contains at least a state from each $F_1, \ldots, F_n$.

(Global) acceptance

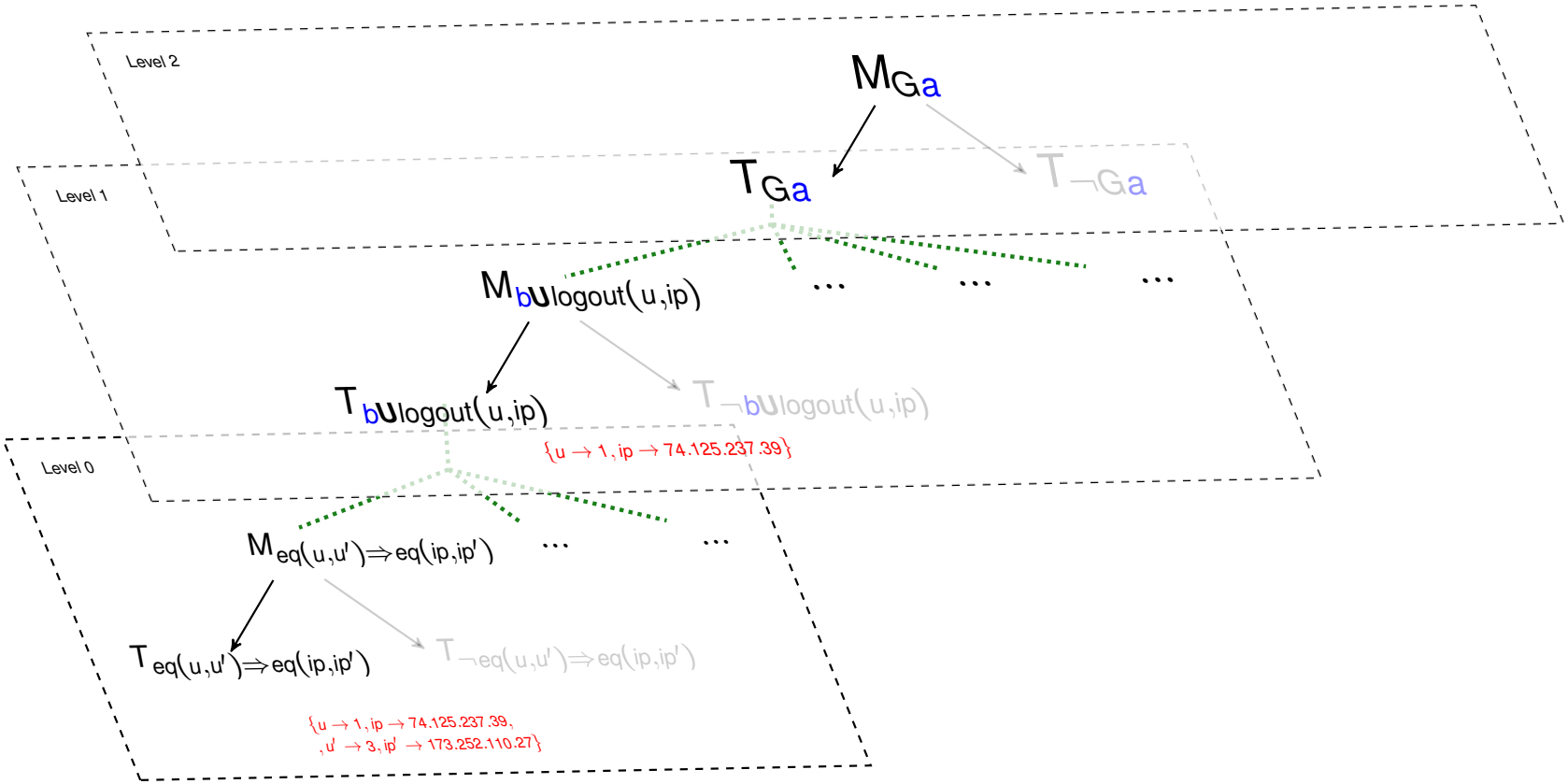
- if $\ell = 0$: local acceptance = acceptance
- if $\ell > 0$: run is local accepting and for all $i \in \mathbb{N}_0$ there is a set $Y \subseteq \mathcal{A}^{<\ell}$, s.t. $Y \models \delta_{\downarrow}(\rho(i), w_i)$ and all $\mathcal{A} \in Y$ have an accepting run over w^i.

Monitor

$$M_\varphi(\overline{\mathfrak{A}}, u) = \top \Rightarrow (\overline{\mathfrak{A}}, u) \in \mathsf{good}(\varphi)$$

(resp. for $\bot$ and $\mathsf{bad}(\varphi)$)

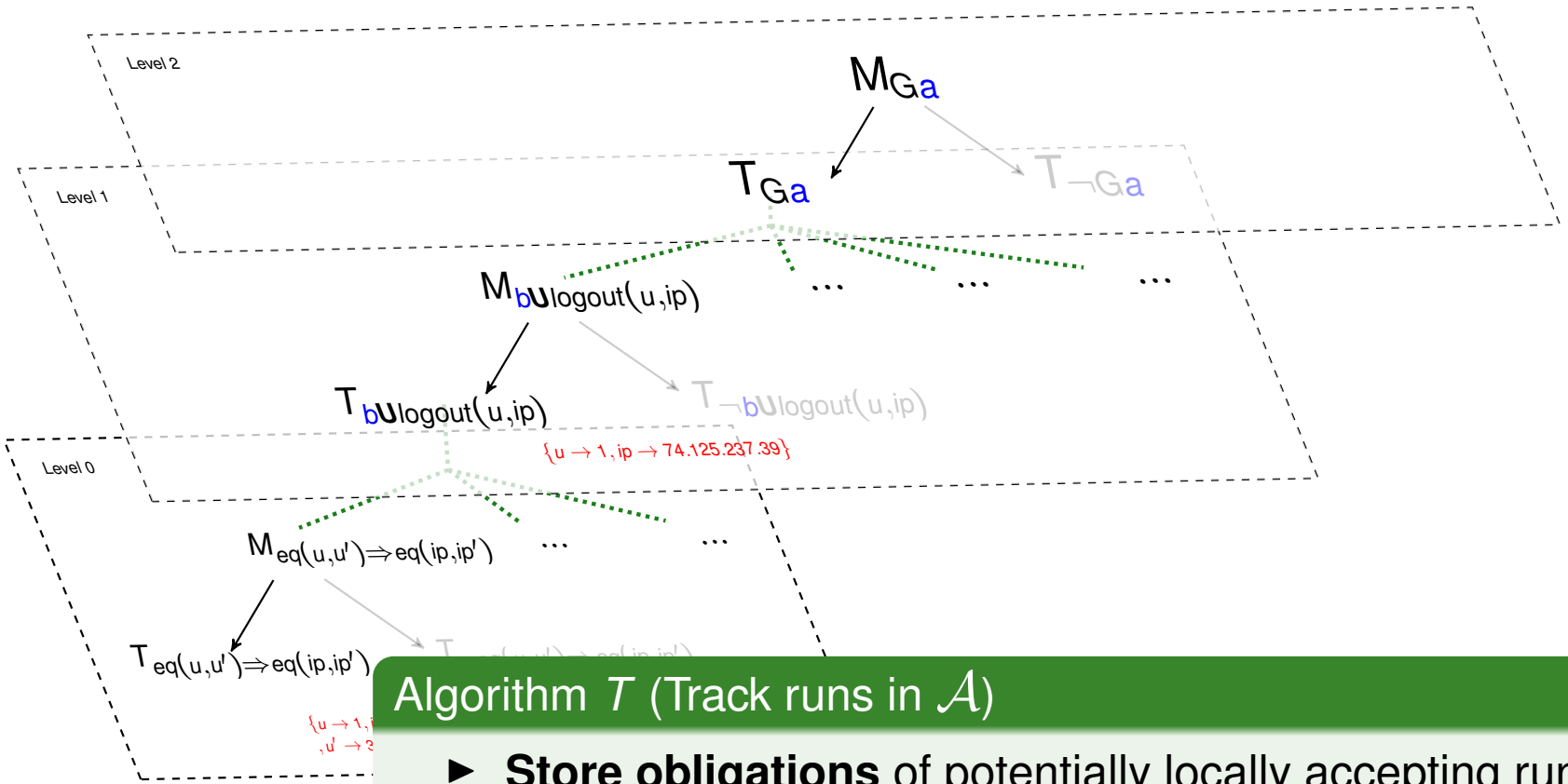
A monitor for LTL^{FO}



Algorithm *M* (Monitor)

- **Create** two instances of **Algorithm T** (for φ and $\neg\varphi$)
- **Forward event** to $T_{\varphi,v}$ and $T_{\neg\varphi,v}$
- **Return** $\bot$ (resp. $\top$) **if** $T_{\varphi,v}$ (resp. $T_{\neg\varphi,v}$) has **"no runs"**, **otherwise print "?"**

A monitor for LTL^{FO}



Algorithm T (Track runs in $\mathcal{A}$)

- **Store obligations** of potentially locally accepting runs: $\{[\ldots, \delta_{\downarrow}(\rho(i), \sigma_i), \ldots], \ldots\}$
- **Execute** M for each $\mathcal{A}$ in $\delta_{\downarrow}(\rho(i), \sigma_i)$ and **wait for result**
- **Remove runs** if $\delta_{\downarrow}(\rho(i), \sigma_i) = \bot$
- **Remove obligations** if $\delta_{\downarrow}(\rho(i), \sigma_i) = \top$

Monitor properties

Trace-length independence

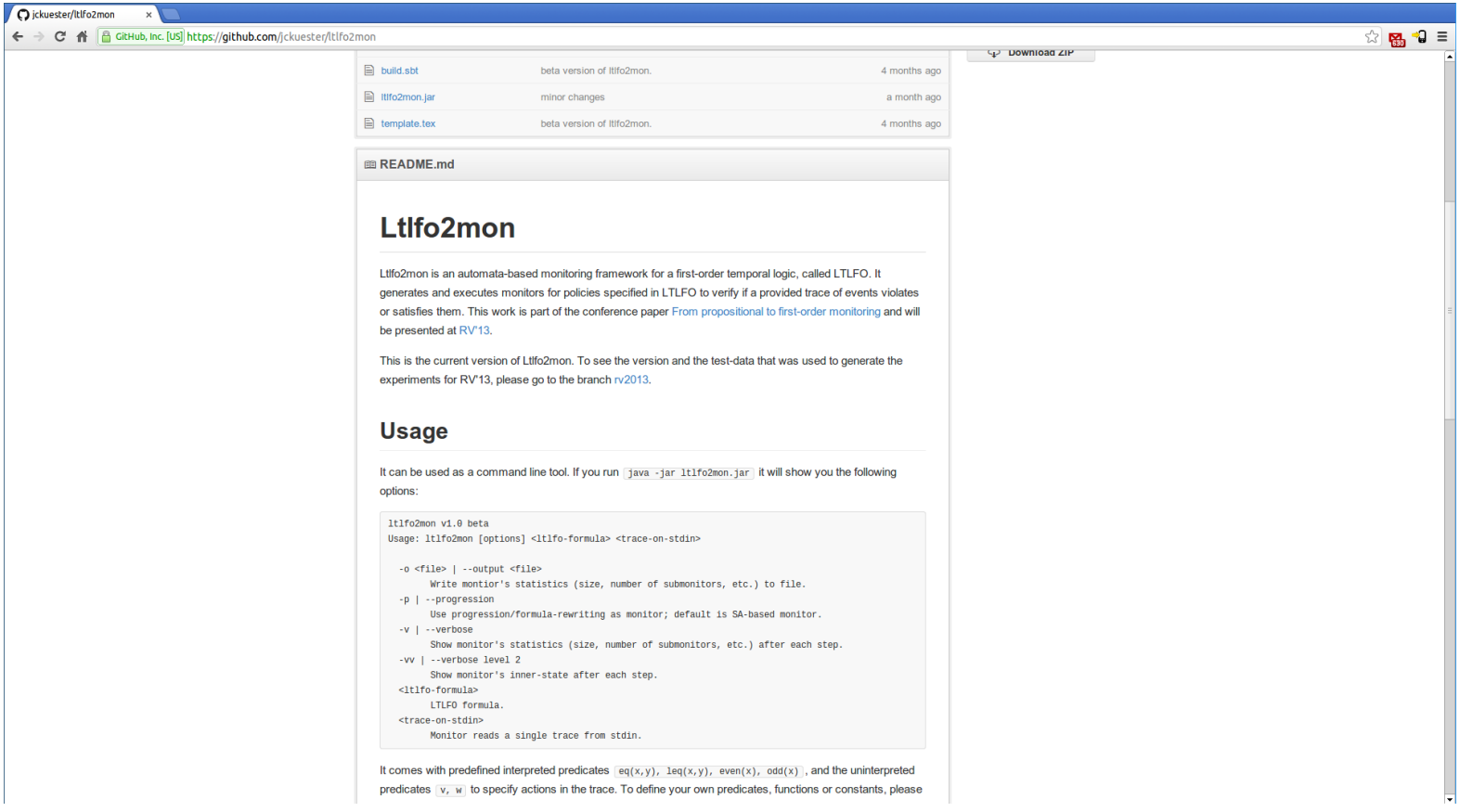
- If a formula is not trace-length dependent, then the monitor is trace-length independent.
- Effciency does not decline with an increasing number of observations.

Monotonic

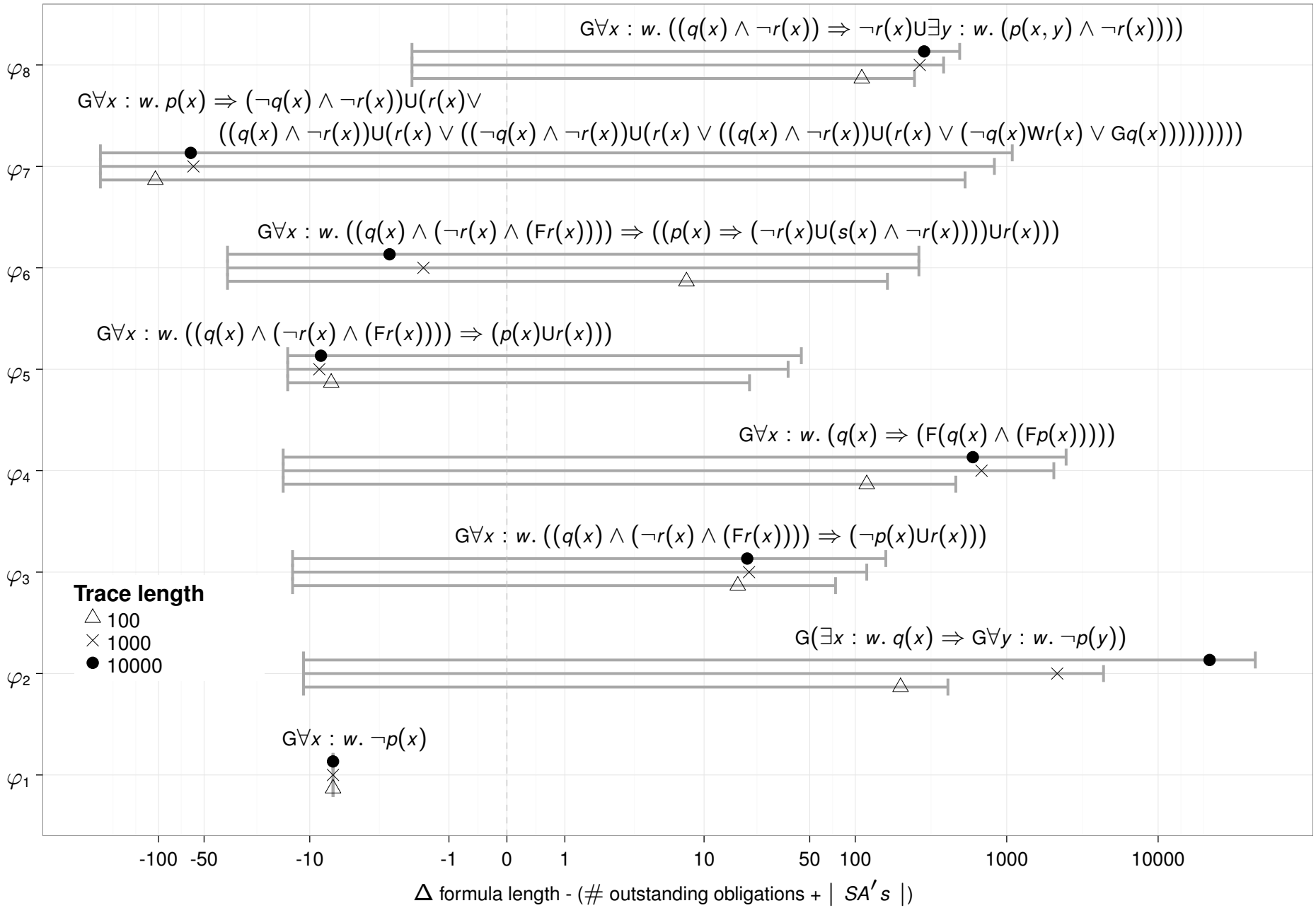
- Once the monitor **returns** $\bot$ to the user, i.e., has **found a bad prefix**, additional observations do not lead to it **returning** $\top$ (and vice versa).

Tool: Ltlfo2Mon

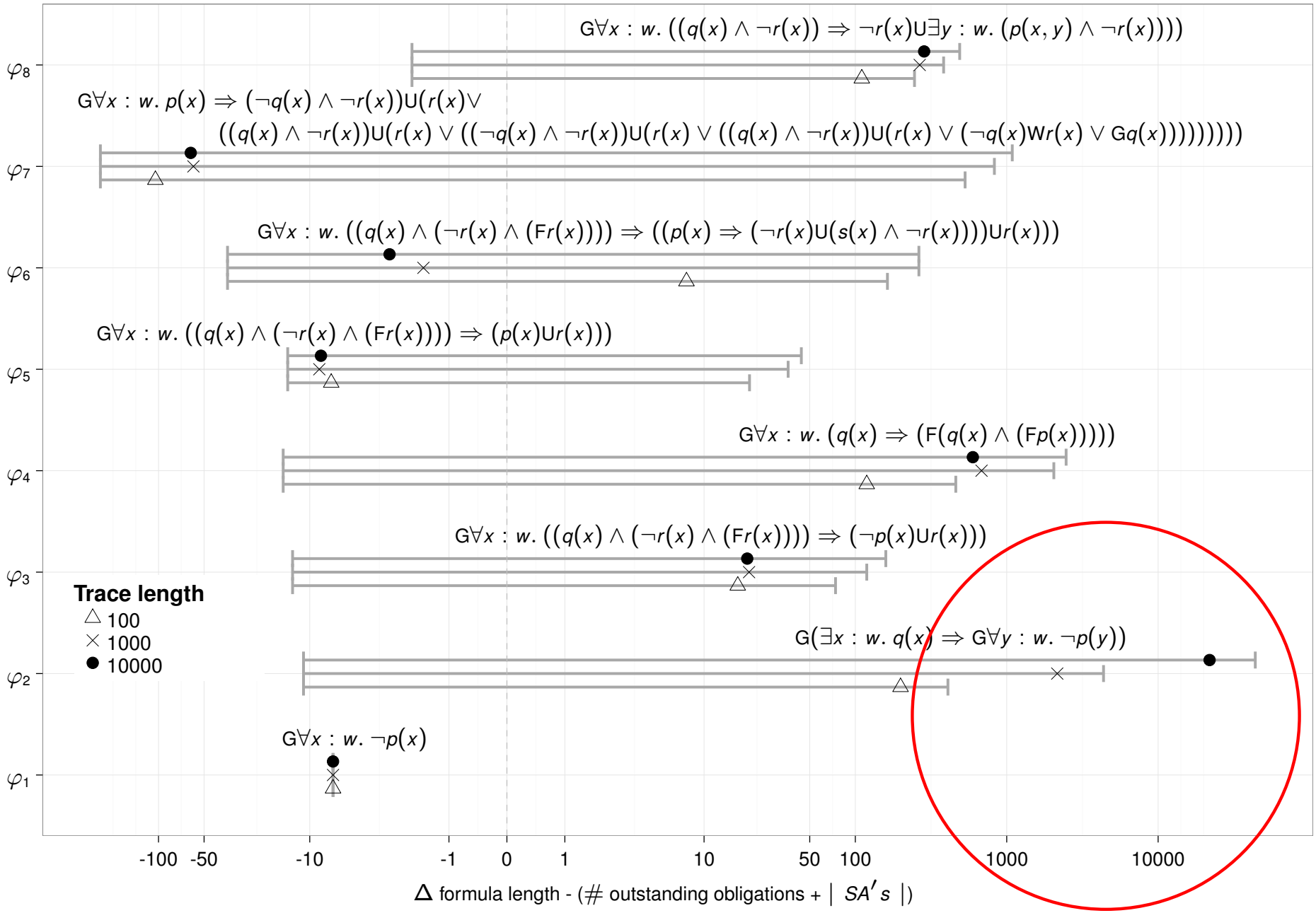
https://github.com/jckuester/ltlfo2mon



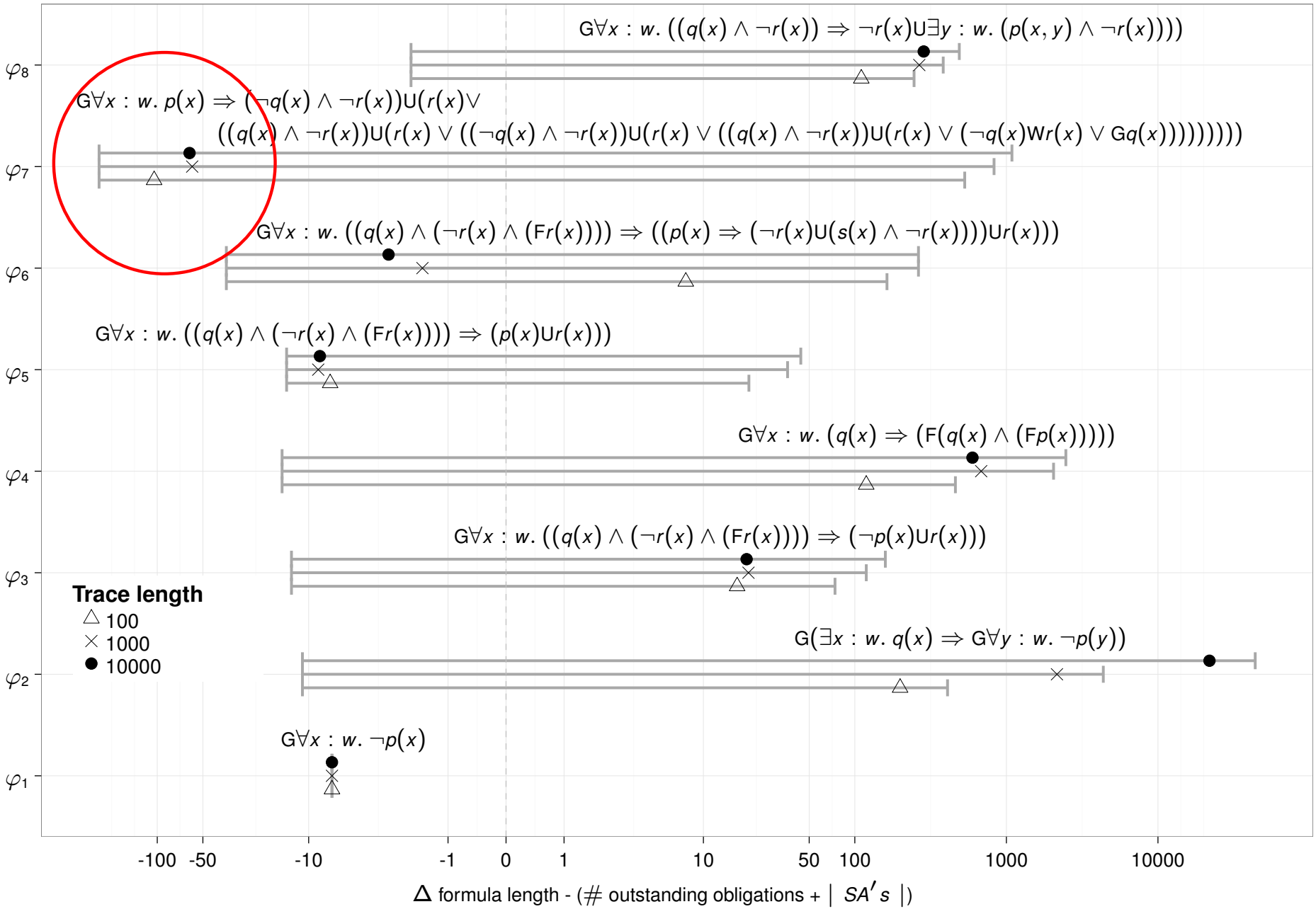
SA-based monitor vs. formula rewriting



SA-based monitor vs. formula rewriting



SA-based monitor vs. formula rewriting



Complexity results: LTL vs. LTL^{FO}

	Satisfiability	**Word problem**	**Model checking**	**Prefix problem**
LTL	PSpace-complete	$<$ Bilinear-time	PSpace-complete	PSpace-complete

Complexity results: LTL vs. LTL^{FO}

	Satisfiability	**Word problem**	**Model checking**	**Prefix problem**
LTL	PSpace-complete	$<$ Bilinear-time	PSpace-complete	PSpace-complete
LTL^{FO}	Undecidable	PSpace-complete	ExpSpace-membership, PSpace-hard	Undecidable

Summary

- A first-order temporal logic, called LTL^{FO}, suitable for reasoning about systems with data at runtime.
- New automaton model called spawning automaton.
- Monitor algorithm for LTL^{FO} based on spawning automaton.
  - https://github.com/jckuester/ltlfo2mon
- Complexity results LTL vs. LTL^{FO}