Symbolic ω-automata with Obligations

Luca Di Stefano

TU Wien, Austria

HIGHLIGHTS, September 2026

A Bit of Context

These days, many automata-based approaches require data-awareness

E.g., reactive synthesis: we need finite presentations for ∞-state arenas.

Typical solution: store values & test them later (RA-style)

A Bit of Context

These days, many automata-based approaches require data-awareness

E.g., reactive synthesis: we need finite presentations for ∞-state arenas.

Typical solution: store values & test them later (RA-style)

State of the art: Bespoke temporal logics with assignments (eg., TSLMT), automata with variables (eg., RPG), or a combination (eg., Issy, Sweap)

A Bit of Context

These days, many automata-based approaches require data-awareness

E.g., reactive synthesis: we need finite presentations for ∞-state arenas.

Typical solution: store values & test them later (RA-style)

State of the art: Bespoke temporal logics with assignments (eg., TSLMT), automata with variables (eg., RPG), or a combination (eg., Issy, Sweap)

Still, I find registers a bit annoying: spooky action at a distance!
Several useful things get complicated:

- Nondeterminism
- Emerson-Lei acceptance
- Alternation

A Bit of Context

These days, many automata-based approaches require data-awareness

E.g., reactive synthesis: we need finite presentations for ∞-state arenas.

Typical solution: store values & test them later (RA-style)

State of the art: Bespoke temporal logics with assignments (eg., TSLMT), automata with variables (eg., RPG), or a combination (eg., Issy, Sweap)

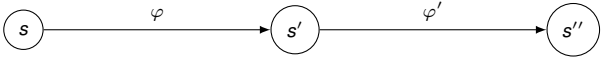
Still, I find registers a bit annoying: spooky action at a distance!
Several useful things get complicated:

- Nondeterminism
- Emerson-Lei acceptance
- Alternation

What if we leave memory to the world outside the automaton?

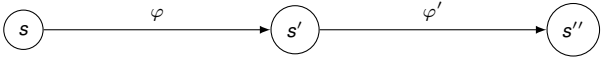
Transitions with Obligations

Traditional symbolic automaton over V: when in s, read a valuation $v \in \mathcal{V}als(V)$ & test it against φ. If $v \models \varphi$, transition to s' is possible.

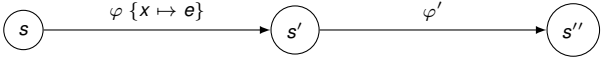


Transitions with Obligations

Traditional symbolic automaton over V: when in s, read a valuation $v \in \mathcal{V}als(V)$ & test it against φ. If $v \models \varphi$, transition to s' is possible.

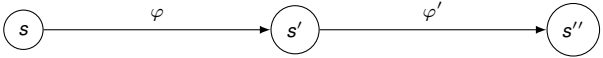


For $x \in V$ we can add an expression $e : \mathcal{V}als \to \mathrm{dom}(x)$ mapped to x

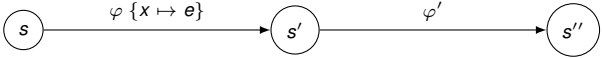


Transitions with Obligations

Traditional symbolic automaton over V: when in s, read a valuation $v \in \mathcal{V}als(V)$ & test it against φ. If $v \models \varphi$, transition to s' is possible.



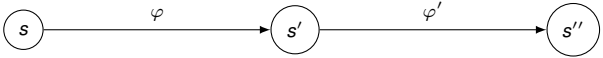
For $x \in V$ we can add an expression $e : \mathcal{V}als \to \mathrm{dom}(x)$ mapped to x



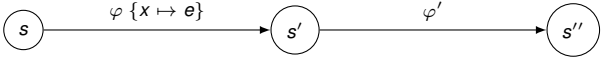
Assume that we are in s, read v, test $v \models \varphi$, reach s'. Compute $k = e(v)$.

Transitions with Obligations

Traditional symbolic automaton over V: when in s, read a valuation $v \in \mathcal{V}als(V)$ & test it against φ. If $v \models \varphi$, transition to s' is possible.



For $x \in V$ we can add an expression $e : \mathcal{V}als \to \mathrm{dom}(x)$ mapped to x

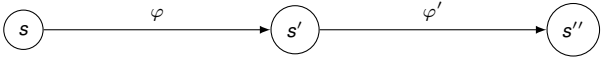


Assume that we are in s, read v, test $v \models \varphi$, reach s'. Compute $k = e(v)$.

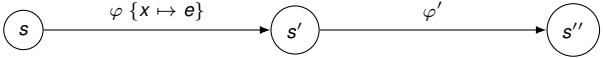
Now, to reach s'', the new valuation must satisfy φ' and also $x = k$.

Transitions with Obligations

Traditional symbolic automaton over V: when in s, read a valuation $v \in \mathcal{V}als(V)$ & test it against φ. If $v \models \varphi$, transition to s' is possible.



For $x \in V$ we can add an expression $e : \mathcal{V}als \to \mathrm{dom}(x)$ mapped to x



Assume that we are in s, read v, test $v \models \varphi$, reach s'. Compute $k = e(v)$.

Now, to reach s'', the new valuation must satisfy φ' and also $x = k$.

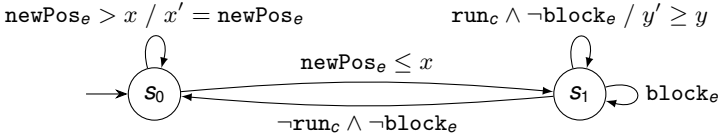
Somewhat more of a finite-height PDA without a separate stack alphabet. Relatively easy to define nondeterminism, E-L acceptance, and universal branching on top of this

Example: Infinite Race in Sweap

Symbolic automaton for arena, LTL (+ predicates) for goal

Automaton reads valuations over $\mathbb{E} \cup \mathbb{C}$

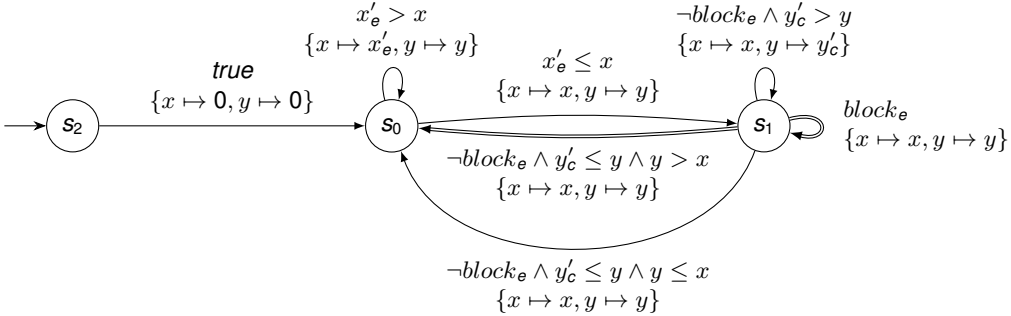
Arena has nondet. assignments to variables V



$$V = \{x \overset{\text{init}}{=} 0, y \overset{\text{init}}{=} 0\}; \quad \mathbb{E} = \{\texttt{newPos}_e, \texttt{block}_e\}; \quad \mathbb{C} = \{\texttt{run}_c\}$$

Goal: $GF(s_1 \wedge \neg\texttt{block}) \implies GF(s_0 \wedge (y > x))$

Infinite Race, with Obligations



Automaton reads valuations over $\{block_e, x, x'_e, y, y'_c\}$

Büchi acceptance

WIP: Emptiness of Büchi Obligation Automata

Assume 1 accepting transition f, 1 initial state s_0

Undecidable in general (can encode any 2-CM). Anyway:

WIP: Emptiness of Büchi Obligation Automata

Assume 1 accepting transition f, 1 initial state s_0

Undecidable in general (can encode any 2-CM). Anyway:

1. Decompose $\mathcal{A}$ into SCCs. Let K_0 the SCC with s_0 and K_f the one with f
2. In K_f disprove that f is taken finitely many times (persistence checking). The counter-proof gives a weakest precondition wp s.t. f is taken ∞ often.
3. If $K_0 \neq K_f$, prove a path $K_0, K_1, \ldots, K_f$ such that wp holds upon reaching K_f (weakest pre-/strongest postcondition checking)

Decidability for a family $\mathbb{A}$ (with obligation expressions from a theory $\mathcal{T}$) is entirely determined by the decidability of steps 2. and 3. in $\mathbb{A}$.

Conclusion

No spooky action at a distance $\Rightarrow$ Easier compositional reasoning?

Practical side:

- Operations like automata product, . . . on top of Spot (precise, reversible lowering into symbolic automata)
- General emptiness checking via IC3

Future applications/directions: stateful arenas for synthesis, data-aware runtime monitors, logics with obligation modalities . . .

Conclusion

No spooky action at a distance $\Rightarrow$ Easier compositional reasoning?

Practical side:

- Operations like automata product, . . . on top of Spot (precise, reversible lowering into symbolic automata)
- General emptiness checking via IC3

Future applications/directions: stateful arenas for synthesis, data-aware runtime monitors, logics with obligation modalities . . .

Preprint (bit outdated)

Prototype tool

Backup slides

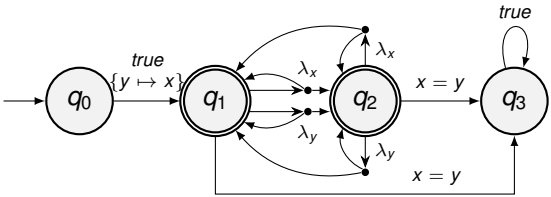
Example: Non-repeating Values

Language $\textsc{Nonce}_x$ of data words where x never takes the same value twice

Example: Non-repeating Values

Language NONCE_x of data words where x never takes the same value twice

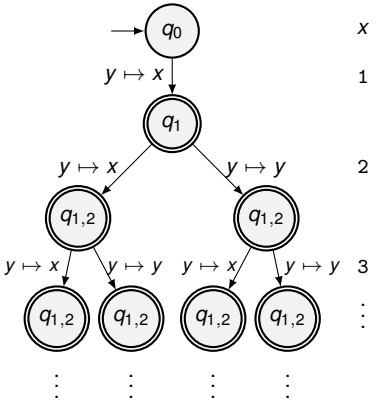
We need universal branching + a support variable y:



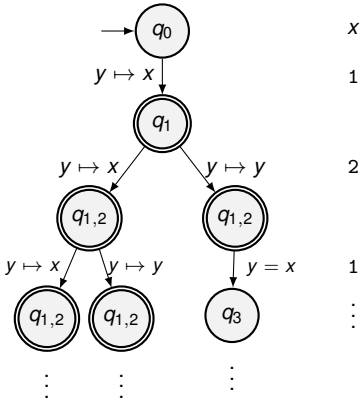
with $\lambda_x = (x \neq y, \{y \mapsto x\})$, $\lambda_y = (x \neq y, \{y \mapsto y\})$

Intuitively: if the word repeats the value of x then a branch in the run-tree reaches q_3 and never leaves

Run-trees from slide 2



(a) A word where x always changes is accepted.



(b) A word where x takes the same value twice is rejected: the right-most branch can never leave q_3.

Example: Repeating Values

Typical example in register automata

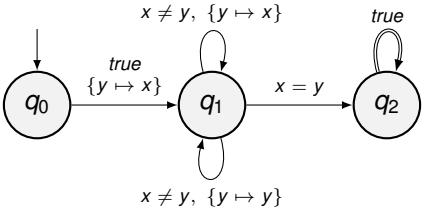
Recognise the language $\textsc{Twice}_x$ of data words where x takes the same value twice (or more)

Example: Repeating Values

Typical example in register automata

Recognise the language $\textsc{Twice}_x$ of data words where x takes the same value twice (or more)

Support variable playing the part of the register + Nondeterminism



Semantics

Obligation $o = \{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$, with t_i from a theory over V

$[\![o]\!]v = \bigwedge_i (x_i = [\![t_i]\!]v)$. For the empty obligation, $[\![\varepsilon]\!]v = \mathit{true}$, for every v.

Word = sequence of valuations $\mathbf{v} = v_1 v_2 \ldots \in \mathcal{V}als^\omega$.

Run over a word $\mathbf{v}$ = sequence of edges $(q_i, \gamma_i, o_i, q'_i)$, $i \in \mathbb{N}$
such that q_1 is an initial state, and for every i:

1. $q'_i = q_{i+1}$;
2. $v_i \models \gamma_i$; and
3. $v_{i+1} \models [\![o_i]\!]v_i$.

Semantics (Universal Branching)

And-state = non-empty set of states (conjunction). Run over a word $\mathbf{v}$ = infinite DAG. Nodes labelled by and-states, (unique) root must be initial.

Arcs labelled $(\gamma, o, \mathbf{f})$, where γ is a predicate, o an obligation, and $\mathbf{f}$ a set of indices referring to accepting sets (accepting indices)

For a node labelled by $Z = \{q_1, \ldots, q_n\}$, an arc $Z \xrightarrow{\gamma, o, \mathbf{f}} Z'$ exists iff:

There exist n edges $e_j = (q_j, \gamma_j, o_j, Z'_j)$ such that $e_j \in R_{q_j}$ for every j;
$\gamma = \bigwedge_j \gamma_j$; $o = o_1 \times \ldots \times o_n$; $Z' = \bigcup_j Z'_j$; $\mathbf{f} = \{k \mid \exists j. e_j \in F_k\}$, where F_k is the k-th accepting set of the automaton;
$v_i \models \gamma_i$, for every i; $v_{i+1} \models [\![o_j]\!] v_i$, for every i, j.

A branch of a run is a maximal path starting from its root. For any branch, let $\mathbf{f}_1\mathbf{f}_2\ldots$ be the sequence of accepting indices on its arcs. Then, the branch is accepting for $\mathrm{Fin}(k)$ iff the set $\{i \mid k \in \mathbf{f}_i\}$ is finite; otherwise, the branch is accepting for $\mathrm{Inf}(k)$. A run is accepting iff all its branches are accepting.