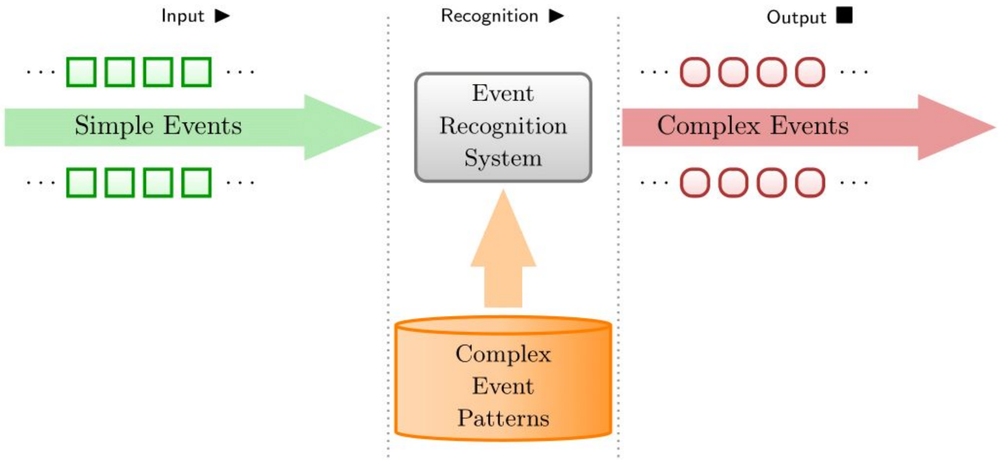
Answer Set Automata: A Learnable Pattern Specification Framework for Complex Event Recognition

Nikos Katzouris, George Paliouras
NCSR "Demokritos"

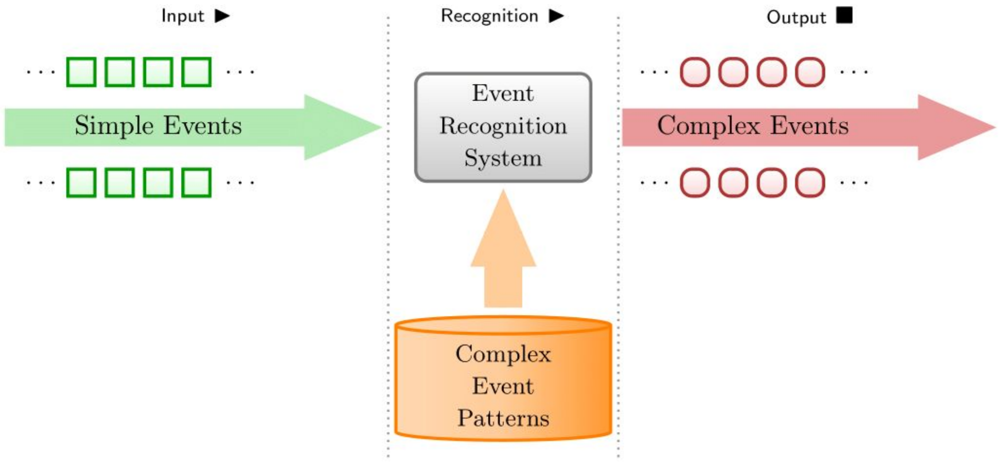
TIME 2023

Complex Event Recognition & Forecasting (CER/F)



- **Recognition:**
  - Matches of the patterns on the input.
- **Forecasting:**
  - Likelihood of future full pattern matches, given observed partial matches.

Complex Event Recognition & Forecasting (CER/F)



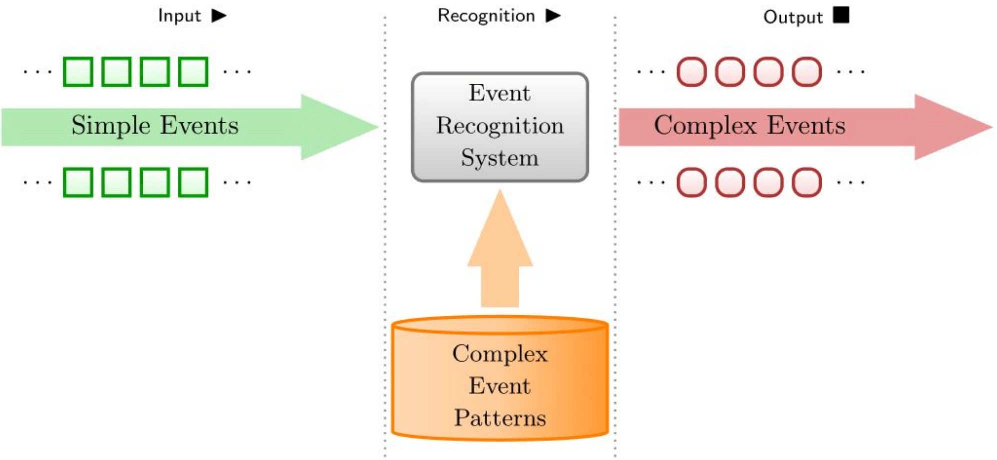
- **Recognition:**
  - Matches of the patterns on the input.
- **Forecasting:**
  - Likelihood of future full pattern matches, given observed partial matches.

Complex event patterns:

- Typically manually authored, specifying situations interest.
- Using **Event Specification Languages (ESLs).**
- Declarative.
- Formal, compositional semantics.

complex event (CE) :=	input event	base case
	FILTER$_p(CE)$	apply predicate p on the variables of CE
	SEQ(CE_1, CE_2)	sequence
	ITER(CE)	Kleene Closure
	CE_1 OR CE_2	disjunction
	CE_1 AND CE_2	conjunction (:= SEQ(CE_1, CE_2) OR SEQ(CE_2, CE_1))
	WITHIN$_{t_1}^{t_2}(CE)$	windowing
	SELECT(CE)	event selection strategies (strict-contiguity, skip-till-next-match...)

Complex Event Recognition & Forecasting (CER/F)



- **Recognition:**
  - Matches of the patterns on the input.
- **Forecasting:**
  - Likelihood of future full pattern matches, given observed partial matches.

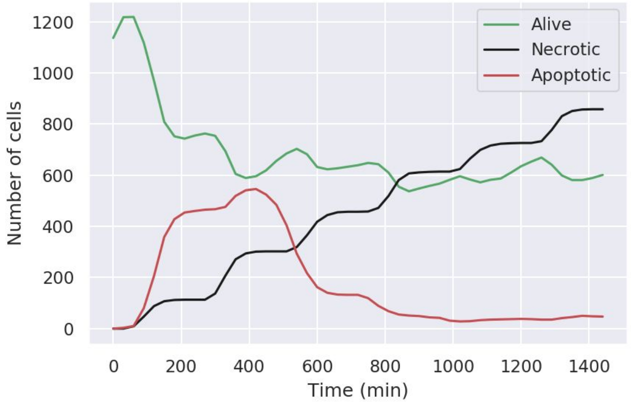
Complex event patterns:

- **Often unknown, or change over time.**
- **Can we learn/revise them from data?**

complex event (CE) :=	input event	|base case
	FILTER$_p(CE)$	|apply predicate p on the variables of CE
	SEQ(CE_1, CE_2)	|sequence
	ITER(CE)	|Kleene Closure
	CE_1 OR CE_2	|disjunction
	CE_1 AND CE_2	|conjunction (:= SEQ(CE_1, CE_2) OR SEQ(CE_2, CE_1))
	WITHIN$_{t_1}^{t_2}(CE)$	|windowing
	SELECT(CE)	|event selection strategies (strict-contiguity, skip-till-next-match...)

Example

Domain: simulation of tumor evolution in response to a drug



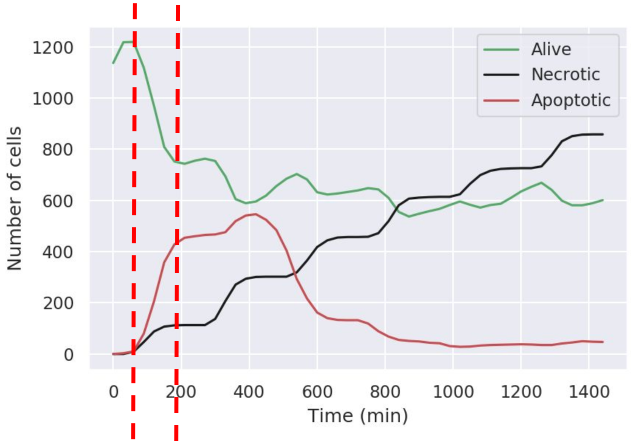
Complex Event Pattern

PATTERN	SEQ(ITER(X_t), ITER(Y_t), ITER(Z_t))
FILTER	X_t.alive $<$ X_{t-1}.alive
AND	X_t.apoptotic $>$ X_{t-1}.apoptotic
AND	Y_t.alive $<$ 800
AND	Z_t.alive $<$ Z_t.necrotic

complex event (CE) :=	input event	|base case
	FILTER$_p(CE)$	|apply predicate p on the variables of CE
	SEQ(CE_1, CE_2)	|sequence
	ITER(CE)	|Kleene Closure
	CE_1 OR CE_2	|disjunction
	CE_1 AND CE_2	|conjunction (:= SEQ(CE_1, CE_2) OR SEQ(CE_2, CE_1))
	WITHIN$_{t_1}^{t_2}(CE)$	|windowing
	SELECT(CE)	|event selection strategies (strict-contiguity, skip-till-next-match...)

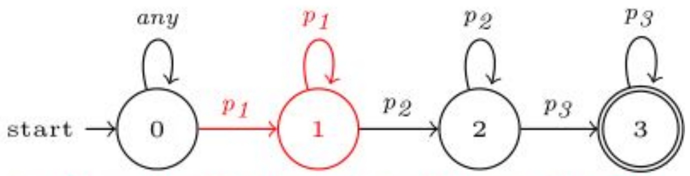
Event Specification Languages & Automata

Domain: simulation of tumor evolution in response to a drug



Complex Event Pattern

PATTERN SEQ(ITER(X_t), ITER(Y_t), ITER(Z_t))
FILTER X_t.alive < X_{t-1}.alive
AND X_t.apoptotic > X_{t-1}.apoptotic
AND Y_t.alive < 800
AND Z_t.alive < Z_t.necrotic



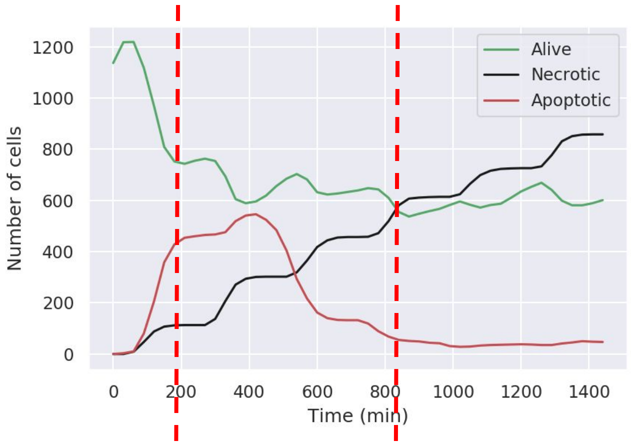
$p_1(T) \leftarrow$ decrease($alive(T)$), increase($apoptotic(T)$).
$p_2(T) \leftarrow$ less_than_val($alive(T)$, 800).
$p_3(T) \leftarrow$ less_than_att($alive(T)$, $necrotic(T)$).

- Patterns usually express **“episodes”** and correspond **to symbolic automata (SFA).**
- SFA: transition guards are predicates, rather than symbols.

Event Specification Languages & Automata

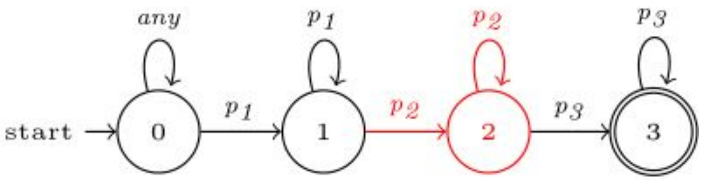
EVENFLOW

Domain: simulation of tumor evolution in response to a drug



Patterns of interesting situations

PATTERN SEQ(ITER(X_t), ITER(Y_t), ITER(Z_t))
FILTER X_t.alive $<$ X_{t-1}.alive
AND X_t.apoptotic $>$ X_{t-1}.apoptotic
AND Y_t.alive $<$ 800
AND Z_t.alive $<$ Z_t.necrotic

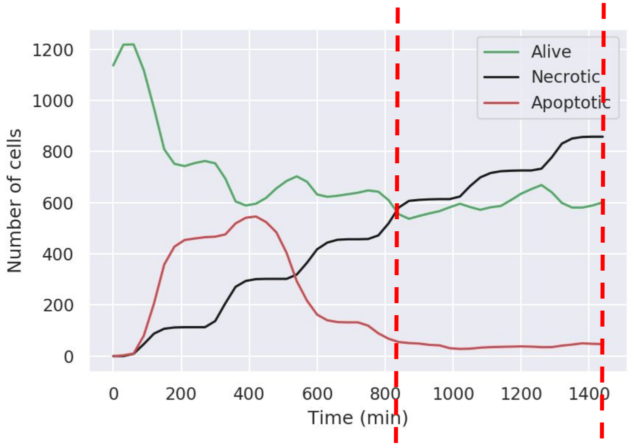


$p_1(T) \leftarrow$ decrease($alive(T)$), increase($apoptotic(T)$).
$p_2(T) \leftarrow$ less_than_val($alive(T)$, 800).
$p_3(T) \leftarrow$ less_than_att($alive(T)$, $necrotic(T)$).

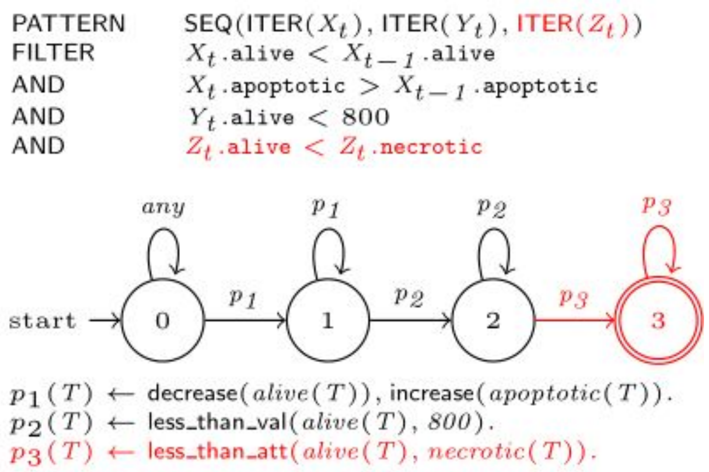
- Patterns usually express **“episodes”** and correspond **to symbolic automata (SFA).**
- SFA: transition guards are predicates, rather than symbols.

Event Specification Languages & Automata

Domain: simulation of tumor evolution in response to a drug



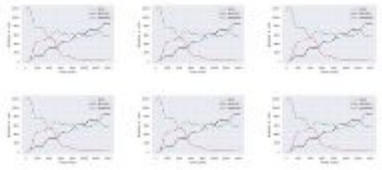
Patterns of interesting situations



- Patterns usually express **“episodes”** and correspond **to symbolic automata (SFA).**
- SFA: transition guards are predicates, rather than symbols.

Approach to Event Pattern Learning

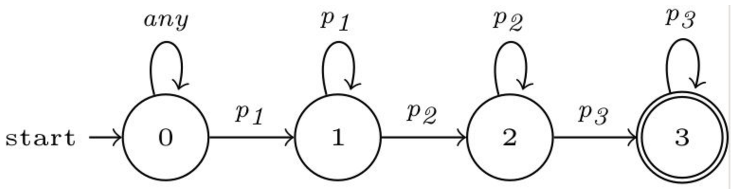
Data



Complex event pattern learning

Patterns of interesting situations

PATTERN	SEQ(ITER(X_t), ITER(Y_t), ITER(Z_t))
FILTER	X_t.alive < X_{t-1}.alive
AND	X_t.apoptotic > X_{t-1}.apoptotic
AND	Y_t.alive < 800
AND	Z_t.alive < Z_t.necrotic



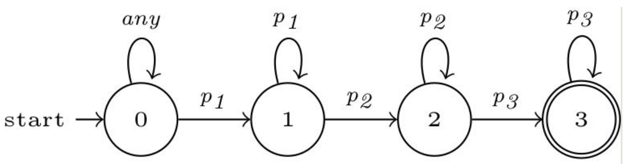
$p_1(T) \leftarrow$ decrease($alive(T)$), increase($apoptotic(T)$).
$p_2(T) \leftarrow$ less_than_val($alive(T)$, 800).
$p_3(T) \leftarrow$ less_than_att($alive(T)$, $necrotic(T)$).

- Approach:
  - Learn **symbolic automata** that correspond to patterns in an event specification language.
- Requirements:
  - **Simultaneously learn the SFA structure and the guards' definitions.**

Answer Set Automata

- Pattern specifications in Answer Set Programming.

Symbolic Automata



$p_1(T) \leftarrow \text{decrease}(alive(T)), \text{increase}(apoptotic(T)).$
$p_2(T) \leftarrow \text{less\_than\_val}(alive(T), 800).$
$p_3(T) \leftarrow \text{less\_than\_att}(alive(T), necrotic(T)).$

Event Specification Languages

PATTERN	SEQ(ITER(X_t), ITER(Y_t), ITER(Z_t))
FILTER	X_t.alive $<$ X_{t-1}.alive
AND	X_t.apoptotic $>$ X_{t-1}.apoptotic
AND	Y_t.alive $<$ 800
AND	Z_t.alive $<$ Z_t.necrotic

Logic Programs

SFA interpreter

$\text{inState}(S_{id}, 0, T) \leftarrow \text{sequence}(S_{id}), \text{start}(T).$
$\text{inState}(S_{id}, S2, T+1) \leftarrow \text{inState}(SeqId, S1, T), \text{transition}(S1, F, S2), \text{holds}(F, SeqId, T).$
$\text{accepted}(S_{id}) \leftarrow \text{inState}(S_{id}, X, T), \text{accepting}(X), \text{seqEnd}(S_{id}, T).$

BK predicates (filters) definitions

$\text{holds}(\text{decrease}(Attribute), S_{id}, T) \leftarrow$ [conditions]
$\text{holds}(\text{increase}(Attribute), S_{id}, T) \leftarrow$ [conditions]
...

SFA structural specification

$\text{transition}(0, any, 0).$ $\text{transition}(0, p_1, 1).$ $\text{transition}(1, p_1, 1).$
$\text{transition}(1, p_2, 2).$ $\text{transition}(2, p_2, 2).$ $\text{transition}(2, p_3, 3).$

Transition guards definitions

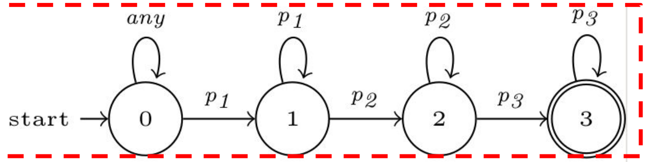
$\text{holds}(p_1, S_{id}, T) \leftarrow \text{holds}(\text{decrease}(alive), S_{id}, T), \text{holds}(\text{increase}(apoptotic), S_{id}, T).$
$\text{holds}(p_2, S_{id}, T) \leftarrow \text{holds}(\text{less\_than\_val}(alive, 800), S_{id}, T).$
...

Answer Set Automata

EVENFLOW

- Pattern specifications in Answer Set Programming.

Symbolic Automata



$p_1(T) \leftarrow \text{decrease}(alive(T)), \text{increase}(apoptotic(T)).$
$p_2(T) \leftarrow \text{less\_than\_val}(alive(T), 800).$
$p_3(T) \leftarrow \text{less\_than\_att}(alive(T), necrotic(T)).$

Event Specification Languages

PATTERN SEQ(ITER(X_t), ITER(Y_t), ITER(Z_t))
FILTER X_t.alive $<$ X_{t-1}.alive
AND X_t.apoptotic $>$ X_{t-1}.apoptotic
AND Y_t.alive $<$ 800
AND Z_t.alive $<$ Z_t.necrotic

SFA structure

Filters definitions

Logic Programs

SFA interpreter
$\text{inState}(S_{id}, 0, T) \leftarrow \text{sequence}(S_{id}), \text{start}(T).$
$\text{inState}(S_{id}, S2, T+1) \leftarrow \text{inState}(SeqId, S1, T), \text{transition}(S1, F, S2), \text{holds}(F, SeqId, T).$
$\text{accepted}(S_{id}) \leftarrow \text{inState}(S_{id}, X, T), \text{accepting}(X), \text{seqEnd}(S_{id}, T).$

BK predicates (filters) definitions
$\text{holds}(\text{decrease}(Attribute), S_{id}, T) \leftarrow$ [conditions]
$\text{holds}(\text{increase}(Attribute), S_{id}, T) \leftarrow$ [conditions]
...

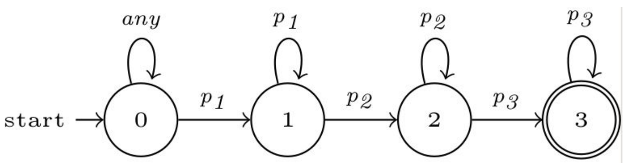
SFA structural specification
$\text{transition}(0, any, 0).$ $\text{transition}(0, p_1, 1).$ $\text{transition}(1, p_1, 1).$
$\text{transition}(1, p_2, 2).$ $\text{transition}(2, p_2, 2).$ $\text{transition}(2, p_3, 3).$

Transition guards definitions
$\text{holds}(p_1, S_{id}, T) \leftarrow \text{holds}(\text{decrease}(alive), S_{id}, T), \text{holds}(\text{increase}(apoptotic), S_{id}, T).$
$\text{holds}(p_2, S_{id}, T) \leftarrow \text{holds}(\text{less\_than\_val}(alive, 800), S_{id}, T).$
...

Answer Set Automata

- Pattern specifications in Answer Set Programming (ASP)
- Pattern matching with a CER engine equivalent to reasoning with an ASP solver.

Symbolic Automata



$p_1(T) \leftarrow$ decrease$(alive(T))$, increase$(apoptotic(T))$.
$p_2(T) \leftarrow$ less_than_val$(alive(T), 800)$.
$p_3(T) \leftarrow$ less_than_att$(alive(T), necrotic(T))$.

Event Specification Languages

PATTERN	SEQ(ITER(X_t), ITER(Y_t), ITER(Z_t))
FILTER	X_t.alive $<$ X_{t-1}.alive
AND	X_t.apoptotic $>$ X_{t-1}.apoptotic
AND	Y_t.alive $<$ 800
AND	Z_t.alive $<$ Z_t.necrotic

Logic Programs

SFA interpreter

inState$(S_{id}, 0, T) \leftarrow$ sequence(S_{id}), start(T).
inState$(S_{id}, S2, T+1) \leftarrow$ inState$(SeqId, S1, T)$, transition$(S1, F, S2)$, holds$(F, SeqId, T)$.
accepted$(S_{id}) \leftarrow$ inState(S_{id}, X, T), accepting(X), seqEnd(S_{id}, T).

BK predicates (filters) definitions

holds(decrease$(Attribute), S_{id}, T) \leftarrow$ [conditions]
holds(increase$(Attribute), S_{id}, T) \leftarrow$ [conditions]
...

SFA structural specification

transition$(0, any, 0)$. transition$(0, p_1, 1)$. transition$(1, p_1, 1)$.
transition$(1, p_2, 2)$. transition$(2, p_2, 2)$. transition$(2, p_3, 3)$.

Transition guards definitions

holds$(p_1, S_{id}, T) \leftarrow$ holds(decrease$(alive), S_{id}, T)$, holds(increase$(apoptotic), S_{id}, T)$.
holds$(p_2, S_{id}, T) \leftarrow$ holds(less_than_val$(alive, 800), S_{id}, T)$.
...

Answer Set Automata

Correctness property

Proposition 1 *Let L be any event specification language generated by the grammar:* FILTER | SEQ | ITER | OR *and p be any L-pattern. There exists a program Π_p such that for any finite event tuple sequence s:*

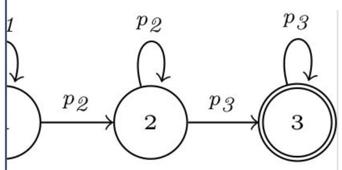
$$\text{matches}(p, s) \textit{ iff } \text{accepted}(s) \in SM(\Pi_p \cup HI(s))$$

Defined inductively on the structure of L

Unique stable model

Logical representation of the input

ic Automata



T)), increase($apoptotic(T)$).
$ve(T), 800$).
$ve(T), necrotic(T)$).

- Pattern specifications in Answer Set Programming (ASP)
- Pattern matching with a CER engine equivalent to reasoning with an ASP solver

Event Specification Languages

PATTERN	SEQ(ITER(X_t), ITER(Y_t), ITER(Z_t))
FILTER	X_t.alive $<$ X_{t-1}.alive
AND	X_t.apoptotic $>$ X_{t-1}.apoptotic
AND	Y_t.alive $<$ 800
AND	Z_t.alive $<$ Z_t.necrotic

Logic Programs

SFA interpreter

inState$(S_{id}, 0, T) \leftarrow$ sequence(S_{id}), start(T).
inState$(S_{id}, S2, T+1) \leftarrow$ inState$(SeqId, S1, T)$, transition$(S1, F, S2)$, holds$(F, SeqId, T)$.
accepted$(S_{id}) \leftarrow$ inState(S_{id}, X, T), accepting(X), seqEnd(S_{id}, T).

BK predicates (filters) definitions

holds(decrease$(Attribute)$, $S_{id}, T) \leftarrow$ [conditions]
holds(increase$(Attribute)$, $S_{id}, T) \leftarrow$ [conditions]
...

SFA structural specification

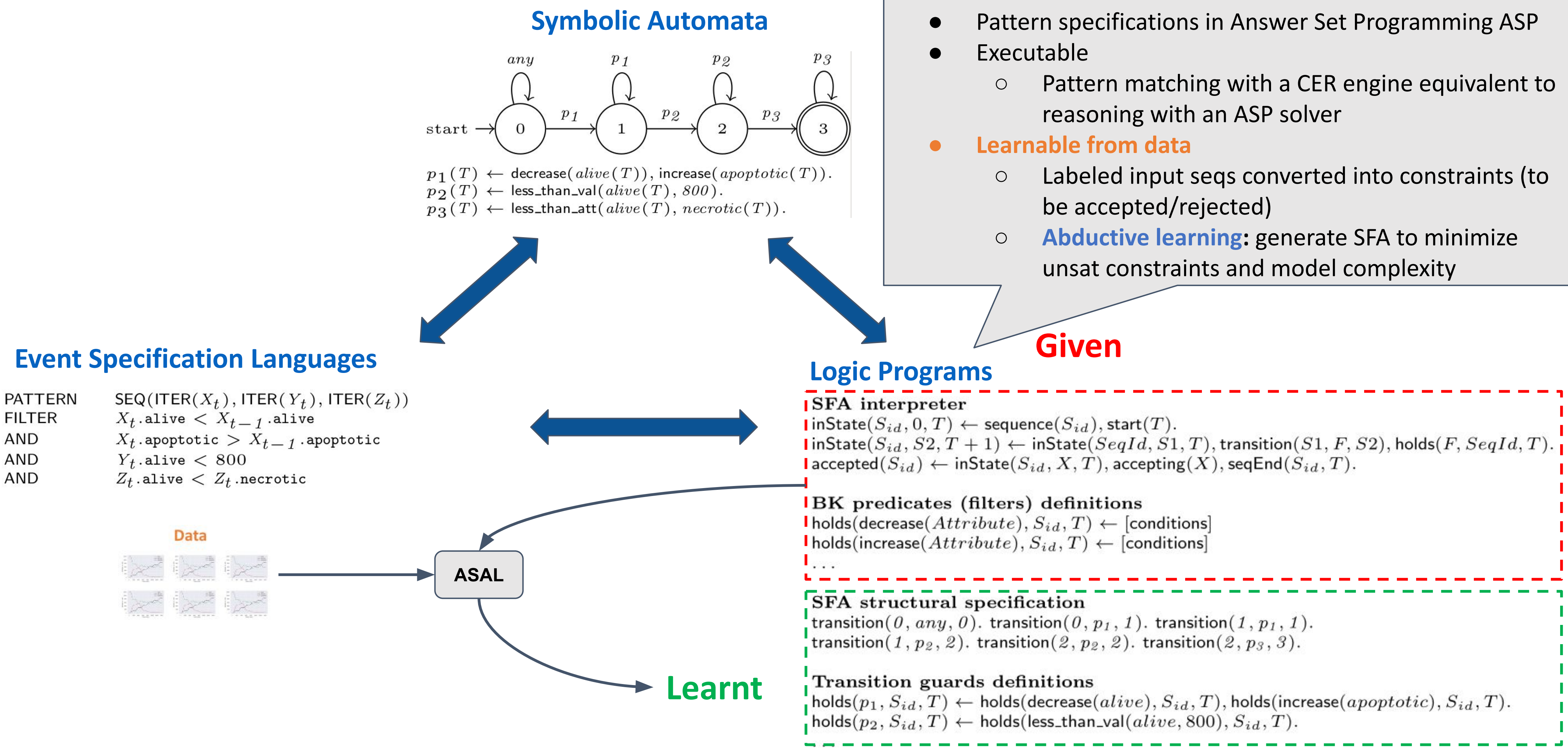
transition$(0, any, 0)$. transition$(0, p_1, 1)$. transition$(1, p_1, 1)$.
transition$(1, p_2, 2)$. transition$(2, p_2, 2)$. transition$(2, p_3, 3)$.

Transition guards definitions

holds$(p_1, S_{id}, T) \leftarrow$ holds(decrease$(alive)$, S_{id}, T), holds(increase$(apoptotic)$, S_{id}, T).
holds$(p_2, S_{id}, T) \leftarrow$ holds(less_than_val$(alive, 800)$, S_{id}, T).
...

Answer Set Automata Learning (ASAL)

- Pattern specifications in Answer Set Programming ASP
- Executable
  - Pattern matching with a CER engine equivalent to reasoning with an ASP solver
- Learnable from data
  - Labeled input seqs converted into constraints (to be accepted/rejected)
  - **Abductive learning**: generate SFA to minimize unsat constraints and model complexity



Answer Set Automata Learning (ASAL)

Algorithm 1 ASAL$(n, m, t, DSFA, ESS, \mathcal{I}, \mathcal{B}, \mathcal{S})$

Input: n: max number of states ; m: max number of alternative (disjunctive) definitions for a guard; t: solving time limit; $DSFA$: boolean flag for (n-)deterministic SFA; ESS: event selection strategy; $\mathcal{I}$: SFA interpreter; $\mathcal{B}$: BK predicate definitions; $\mathcal{S}$: labeled training set.

Output: $\mathcal{T}$: structural SFA specification of up to n states; $\mathcal{G}$: transition guard definitions

$\mathcal{E} \leftarrow$ guard_template$(n, DSFA, ESS)$.
$\mathcal{P}_1 \leftarrow$ generate_part$(n, m, \mathcal{B})$.
$\mathcal{P}_2 \leftarrow$ test_part$(\mathcal{B})$.
$\mathcal{M} \leftarrow$ solve$(t, \mathcal{E}, \mathcal{P}_1, \mathcal{P}_2, \mathcal{I}, \mathcal{B}, \mathcal{S})$.
$(\mathcal{T}, \mathcal{G}) \leftarrow$ assemble$(\mathcal{M}, \mathcal{E})$.
return $(\mathcal{T}, \mathcal{G})$.

function assemble$(\mathcal{M}, \mathcal{E})$:
  $\mathcal{T} \leftarrow$ all transition/3 facts in $\mathcal{M}$
  $\mathcal{G} \leftarrow \emptyset$
  for each atom $\alpha \in \mathcal{M}$ of the form $\alpha :=$ atom(i, j, δ):
    $g_{ij} \leftarrow$ the j-th disjunct of guard i's definition
    if no such g_{ij} exists in $\mathcal{G}$:
      $\mathcal{G} \leftarrow \mathcal{G} \cup$ holds$(g_{ij}, S, T) \leftarrow$ *# adds empty-bodied rule*
    else add δ to the body of g_{ij}
  for each rule $g_{ij} \in \mathcal{G}$
    add to g_{ij}'s body its corresponding mutual exclusivity conditions specified in $\mathcal{E}$.
  return $(\mathcal{T}, \mathcal{G})$

Answer Set Automata Learning (ASAL)

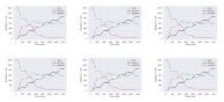
Algorithm 1 ASAL$(n, m, t, DSFA, ESS, \mathcal{I}, \mathcal{B}, \mathcal{S})$
Input: n: max number of states ; m: max number of alternative (disjunctive) definitions for a guard; t: solving time limit; $DSFA$: boolean flag for (n-)deterministic SFA; ESS: event selection strategy; $\mathcal{I}$: SFA interpreter; $\mathcal{B}$: BK predicate definitions; $\mathcal{S}$: labeled training set.
Output: $\mathcal{T}$: structural SFA specification of up to n states; $\mathcal{G}$: transition guard definitions

$\mathcal{E} \leftarrow$ guard_template$(n, DSFA, ESS)$.
$\mathcal{P}_1 \leftarrow$ generate_part$(n, m, \mathcal{B})$.
$\mathcal{P}_2 \leftarrow$ test_part$(\mathcal{B})$.
$\mathcal{M} \leftarrow$ solve$(t, \mathcal{E}, \mathcal{P}_1, \mathcal{P}_2, \mathcal{I}, \mathcal{B}, \mathcal{S})$.
$(\mathcal{T}, \mathcal{G}) \leftarrow$ assemble$(\mathcal{M}, \mathcal{E})$.
return $(\mathcal{T}, \mathcal{G})$.

function assemble$(\mathcal{M}, \mathcal{E})$:
  $\mathcal{T} \leftarrow$ all transition/3 facts in $\mathcal{M}$
  $\mathcal{G} \leftarrow \emptyset$
  for each atom $\alpha \in \mathcal{M}$ of the form $\alpha :=$ atom(i, j, δ):
    $g_{ij} \leftarrow$ the j-th disjunct of guard i's definition
    if no such g_{ij} exists in $\mathcal{G}$:
      $\mathcal{G} \leftarrow \mathcal{G} \cup$ holds$(g_{ij}, S, T) \leftarrow$ *# adds empty-bodied rule*
    else add δ to the body of g_{ij}
  for each rule $g_{ij} \in \mathcal{G}$
    add to g_{ij}'s body its corresponding mutual exclusivity conditions specified in $\mathcal{E}$.
  return $(\mathcal{T}, \mathcal{G})$

Input data as labeled Herband Interpretations

Data



obs$(s_1$, av$(al, 200), 0), \ldots,$ obs$(s_1$, av$(al, 83), 50)$
obs$(s_1$, av$(ap, 40), 0), \ldots,$ obs$(s_1$, av$(ap, 5), 50)$
obs$(s_1$, av$(n, 0), 0), \ldots,$ obs$(s_1$, av$(n, 800), 50)$
class$(s_1$, positive$)$
...
class$(s_{10}$, negative$)$

Answer Set Automata Learning (ASAL)

Algorithm 1 ASAL$(n, m, t, DSFA, ESS, \mathcal{I}, \mathcal{B}, \mathcal{S})$

Input: n: max number of states ; m: max number of alternative (disjunctive) definitions for a guard; t: solving time limit; $DSFA$: boolean flag for (n-)deterministic SFA; ESS: event selection strategy; $\mathcal{I}$: SFA interpreter; $\mathcal{B}$: BK predicate definitions; $\mathcal{S}$: labeled training set.

Output: $\mathcal{T}$: structural SFA specification of up to n states; $\mathcal{G}$: transition guard definitions

```
E ← guard_template(n, DSFA, ESS).
P1 ← generate_part(n, m, B).
P2 ← test_part(B).
M ← solve(t, E, P1, P2, I, B, S).
(T, G) ← assemble(M, E).
return (T, G).

function assemble(M, E):
  T ← all transition/3 facts in M
  G ← ∅
  for each atom α ∈ M of the form α := atom(i, j, δ):
    g_ij ← the j-th disjunct of guard i's definition
    if no such g_ij exists in G:
      G ← G ∪ holds(g_ij, S, T) ←        # adds empty-bodied rule
    else add δ to the body of g_ij
  for each rule g_ij ∈ G
    add to g_ij's body its corresponding mutual exclusivity conditions
    specified in E.
  return (T, G)
```

(A) Example result of guard_template$(n = 3, DSFA = \text{true}, ESS = \text{skip-till-any-match})$:

(1) $\text{holds}(g(0,0), S, T) \leftarrow \text{seq}(S), \text{time}(T), \text{not holds}(g(0,1), S, T), \text{not holds}(g(0,2), S, T).$
(2) $\text{holds}(g(0,1), S, T) \leftarrow \text{holds}(\text{body}(g(0,1), J), S, T), \text{not holds}(g(0,2), S, T).$
(3) $\text{holds}(g(0,2), S, T) \leftarrow \text{holds}(\text{body}(g(0,2), J), S, T).$
(4) $\text{holds}(g(1,0), S, T) \leftarrow \text{holds}(\text{body}(g(1,0), J), S, T), \text{not holds}(g(1,2), S, T).$
(5) $\text{holds}(g(1,1), S, T) \leftarrow \text{seq}(S), \text{time}(T), \text{not holds}(g(1,0), S, T), \text{not holds}(g(1,2), S, T).$
(6) $\text{holds}(g(1,2), S, T) \leftarrow \text{holds}(\text{body}(g(1,2), J), S, T).$
(7) $\text{holds}(g(2,2), S, T) \leftarrow \text{seq}(S), \text{time}(T).$
(8) $\leftarrow \text{state}(S), \text{not transition}(S, \_, S).$
(9) $\text{holds}(\text{body}(I, J), S, T) \leftarrow$
$\text{guard}(I), \text{disjunct}(J), \text{seq}(S), \text{time}(T), \text{holds}(F, S, T) : \text{atom}(I, J, F).$

- Provides "placeholder" definitions for the guards of a fully connected graph of up to `max_number` of states.
- **Defeasible:** the goal is to simplify as much as possible, keep only what's necessary to explain the input (discard entire rules or rule conditions).
- Specifies mutual exclusivity conditions for the guards, in case the target is a deterministic SFA.
- **Rule (9)** allows to "unfold" the placeholder definition of the *I*-th guard into *J* disjunctions of conjunctions of BK predicate instances.

Answer Set Automata Learning (ASAL)

Algorithm 1 ASAL$(n, m, t, DSFA, ESS, \mathcal{I}, \mathcal{B}, \mathcal{S})$

Input: n: max number of states ; m: max number of alternative (disjunctive) definitions for a guard; t: solving time limit; $DSFA$: boolean flag for (n-)deterministic SFA; ESS: event selection strategy; $\mathcal{I}$: SFA interpreter; $\mathcal{B}$: BK predicate definitions; $\mathcal{S}$: labeled training set.

Output: $\mathcal{T}$: structural SFA specification of up to n states; $\mathcal{G}$: transition guard definitions

$\mathcal{E} \leftarrow \mathsf{guard\_template}(n, DSFA, ESS)$.
$\mathcal{P}_1 \leftarrow \mathsf{generate\_part}(n, m, \mathcal{B})$.
$\mathcal{P}_2 \leftarrow \mathsf{test\_part}(\mathcal{B})$.
$\mathcal{M} \leftarrow \mathsf{solve}(t, \mathcal{E}, \mathcal{P}_1, \mathcal{P}_2, \mathcal{I}, \mathcal{B}, \mathcal{S})$.
$(\mathcal{T}, \mathcal{G}) \leftarrow \mathsf{assemble}(\mathcal{M}, \mathcal{E})$.
return $(\mathcal{T}, \mathcal{G})$.

function $\mathsf{assemble}(\mathcal{M}, \mathcal{E})$:
  $\mathcal{T} \leftarrow$ all $\mathsf{transition}/3$ facts in $\mathcal{M}$
  $\mathcal{G} \leftarrow \emptyset$
  for each atom $\alpha \in \mathcal{M}$ of the form $\alpha := \mathsf{atom}(i, j, \delta)$:
    $g_{ij} \leftarrow$ the j-th disjunct of guard i's definition
    if no such g_{ij} exists in $\mathcal{G}$:
      $\mathcal{G} \leftarrow \mathcal{G} \cup \mathsf{holds}(g_{ij}, S, T) \leftarrow$ *# adds empty-bodied rule*
    else add δ to the body of g_{ij}
  for each rule $g_{ij} \in \mathcal{G}$
    add to g_{ij}'s body its corresponding mutual exclusivity conditions specified in $\mathcal{E}$.
  return $(\mathcal{T}, \mathcal{G})$

(A) Example result of $\mathsf{guard\_template}(n = 3, DSFA = \mathsf{true}, ESS = \mathsf{skip\text{-}till\text{-}any\text{-}match})$:

(1) $\mathsf{holds}(g(0,0), S, T) \leftarrow \mathsf{seq}(S), \mathsf{time}(T), \mathsf{not\ holds}(g(0,1), S, T), \mathsf{not\ holds}(g(0,2), S, T).$
(2) $\mathsf{holds}(g(0,1), S, T) \leftarrow \mathsf{holds}(\mathsf{body}(g(0,1), J), S, T), \mathsf{not\ holds}(g(0,2), S, T).$
(3) $\mathsf{holds}(g(0,2), S, T) \leftarrow \mathsf{holds}(\mathsf{body}(g(0,2), J), S, T).$
(4) $\mathsf{holds}(g(1,0), S, T) \leftarrow \mathsf{holds}(\mathsf{body}(g(1,0), J), S, T), \mathsf{not\ holds}(g(1,2), S, T).$
(5) $\mathsf{holds}(g(1,1), S, T) \leftarrow \mathsf{seq}(S), \mathsf{time}(T), \mathsf{not\ holds}(g(1,0), S, T), \mathsf{not\ holds}(g(1,2), S, T).$
(6) $\mathsf{holds}(g(1,2), S, T) \leftarrow \mathsf{holds}(\mathsf{body}(g(1,2), J), S, T).$
(7) $\mathsf{holds}(g(2,2), S, T) \leftarrow \mathsf{seq}(S), \mathsf{time}(T).$
(8) $\leftarrow \mathsf{state}(S), \mathsf{not\ transition}(S, \_, S).$
(9) $\mathsf{holds}(\mathsf{body}(I, J), S, T) \leftarrow$
$\mathsf{guard}(I), \mathsf{disjunct}(J), \mathsf{seq}(S), \mathsf{time}(T), \mathsf{holds}(F, S, T) : \mathsf{atom}(I, J, F).$

(B) Example result of $\mathsf{generate\_part}(n, m, \mathcal{B})$ for $\mathcal{B}$ from Table 2(iv):

(10) $\mathsf{state}(0..2).\ \mathsf{start}(0).\ \mathsf{accepting}(2).\ \mathsf{guard}(g(S_1, S_2)) \leftarrow \mathsf{transition}(S_1, g(S_1, S_2), S_2).$
(11) $\{\mathsf{transition}(S_1, g(S_1, S_2), S_2)\} \leftarrow \mathsf{state}(S_1), \mathsf{state}(S_2).$
(12) $\{\mathsf{disjunct}(1..m)\}.$
(13) $\{\mathsf{atom}(I, J, \mathsf{increase}(A))\} \leftarrow \mathsf{guard}(I), \mathsf{disjunct}(J), \mathsf{attr}(A).$
(14) $\{\mathsf{atom}(I, J, \mathsf{less\_than\_val}(A, V))\} \leftarrow \mathsf{guard}(I), \mathsf{disjunct}(J), \mathsf{av}(A, V).$
(15) $\{\mathsf{atom}(I, J, \mathsf{less\_than\_att}(A_1, A_2))\} \leftarrow \mathsf{guard}(I), \mathsf{disjunct}(J), \mathsf{attr}(A_1), \mathsf{attr}(A_2).$

Abduces `atom/3` instances

Answer Set Automata Learning (ASAL)

Algorithm 1 ASAL$(n, m, t, DSFA, ESS, \mathcal{I}, \mathcal{B}, \mathcal{S})$

Input: n: max number of states ; m: max number of alternative (disjunctive) definitions for a guard; t: solving time limit; $DSFA$: boolean flag for (n-)deterministic SFA; ESS: event selection strategy; $\mathcal{I}$: SFA interpreter; $\mathcal{B}$: BK predicate definitions; $\mathcal{S}$: labeled training set.

Output: $\mathcal{T}$: structural SFA specification of up to n states; $\mathcal{G}$: transition guard definitions

```
𝓔 ← guard_template(n, DSFA, ESS).
𝓟1 ← generate_part(n, m, 𝓑).
𝓟2 ← test_part(𝓑).
𝓜 ← solve(t, 𝓔, 𝓟1, 𝓟2, 𝓘, 𝓑, 𝓢).
(𝓣, 𝓖) ← assemble(𝓜, 𝓔).
return (𝓣, 𝓖).

function assemble(𝓜, 𝓔):
  𝓣 ← all transition/3 facts in 𝓜
  𝓖 ← ∅
  for each atom α ∈ 𝓜 of the form α := atom(i, j, δ):
    g_ij ← the j-th disjunct of guard i's definition
    if no such g_ij exists in 𝓖:
      𝓖 ← 𝓖 ∪ holds(g_ij, S, T) ←        # adds empty-bodied rule
    else add δ to the body of g_ij
  for each rule g_ij ∈ 𝓖
    add to g_ij's body its corresponding mutual exclusivity conditions
    specified in 𝓔.
  return (𝓣, 𝓖)
```

(A) Example result of guard_template$(n = 3, DSFA = \text{true}, ESS = \text{skip-till-any-match})$:

(1) holds$(g(0,0), S, T) \leftarrow$ seq(S), time(T), not holds$(g(0,1), S, T)$, not holds$(g(0,2), S, T)$.
(2) holds$(g(0,1), S, T) \leftarrow$ holds(body$(g(0,1), J)$, $S, T)$, not holds$(g(0,2), S, T)$.
(3) holds$(g(0,2), S, T) \leftarrow$ holds(body$(g(0,2), J)$, $S, T)$.
(4) holds$(g(1,0), S, T) \leftarrow$ holds(body$(g(1,0), J)$, $S, T)$, not holds$(g(1,2), S, T)$.
(5) holds$(g(1,1), S, T) \leftarrow$ seq(S), time(T), not holds$(g(1,0), S, T)$, not holds$(g(1,2), S, T)$.
(6) holds$(g(1,2), S, T) \leftarrow$ holds(body$(g(1,2), J)$, $S, T)$.
(7) holds$(g(2,2), S, T) \leftarrow$ seq(S), time(T).
(8) $\leftarrow$ state(S), not transition$(S, _, S)$.
(9) holds(body(I, J), $S, T) \leftarrow$
guard(I), disjunct(J), seq(S), time(T), holds(F, S, T) : atom(I, J, F).

(B) Example result of generate_part$(n, m, \mathcal{B})$ for $\mathcal{B}$ from Table 2(iv):

(10) state$(0..2)$. start(0). accepting(2). guard$(g(S_1, S_2)) \leftarrow$ transition$(S_1, g(S_1, S_2), S_2)$.
(11) {transition$(S_1, g(S_1, S_2), S_2)$} $\leftarrow$ state(S_1), state(S_2).
(12) {disjunct$(1..m)$}.
(13) {atom$(I, J$, increase$(A))$} $\leftarrow$ guard(I), disjunct(J), attr(A).
(14) {atom$(I, J$, less_than_val$(A, V))$} $\leftarrow$ guard(I), disjunct(J), av(A, V).
(15) {atom$(I, J$, less_than_att$(A_1, A_2))$} $\leftarrow$ guard(I), disjunct(J), attr(A_1), attr(A_2).

(C) Example result of test_part$(\mathcal{B})$:

(16) :~ false_negative(S). $[1@0, S]$
(17) :~ false_positive(S). $[1@0, S]$
(18) :~ atom(I, J, F). $[1@0, I, J, F]$
(19) :~ used_attribute(A). $[1@0, A]$
(20) used_attribute$(A) \leftarrow$ atom$(_, _,$ increase$(A))$.
(21) used_attribute$(A) \leftarrow$ atom$(_, _,$ decrease$(A))$.
. . . rest of used_attribute/1 definitions...
(22) false_negative$(S) \leftarrow$ pos(S), not accepted(S).
(23) false_positive$(S) \leftarrow$ neg(S), accepted(S).

Guides the abduction process through (weak) constraints that are to be satisfied “as much a possible”.

Answer Set Automata Learning (ASAL)

EVENFLOW

Algorithm 1 ASAL$(n, m, t, DSFA, ESS, \mathcal{I}, \mathcal{B}, \mathcal{S})$

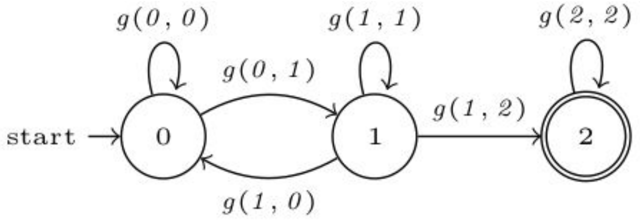
Input: n: max number of states ; m: max number of alternative (disjunctive) definitions for a guard; t: solving time limit; $DSFA$: boolean flag for (n-)deterministic SFA; ESS: event selection strategy; $\mathcal{I}$: SFA interpreter; $\mathcal{B}$: BK predicate definitions; $\mathcal{S}$: labeled training set.

Output: $\mathcal{T}$: structural SFA specification of up to n states; $\mathcal{G}$: transition guard definitions

```
E ← guard_template(n, DSFA, ESS).
P1 ← generate_part(n, m, B).
P2 ← test_part(B).
M ← solve(t, E, P1, P2, I, B, S).
(T, G) ← assemble(M, E).
return (T, G).

function assemble(M, E):
    T ← all transition/3 facts in M
    G ← ∅
    for each atom α ∈ M of the form α := atom(i, j, δ):
        g_ij ← the j-th disjunct of guard i's definition
        if no such g_ij exists in G:
            G ← G ∪ holds(g_ij, S, T) ←        # adds empty-bodied rule
        else add δ to the body of g_ij
    for each rule g_ij ∈ G
        add to g_ij's body its corresponding mutual exclusivity conditions
        specified in E.
    return (T, G)
```

Extracts solutions from the generated and compiles the guards using the template if necessary (for deterministic SFA).

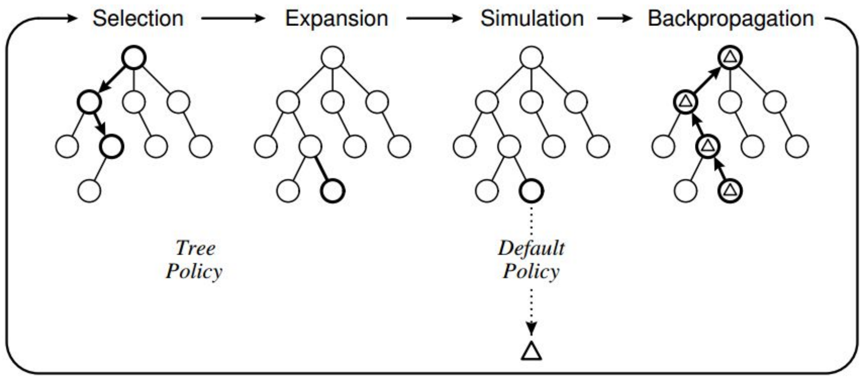


$g(0,0) \leftarrow$ not $g(0,1)$.
$g(1,1) \leftarrow$ not $g(1,0)$, not $g(1,2)$.
$g(2,2) \leftarrow$ #true.
$g(0,1) \leftarrow$ increase($apopt$).
$g(1,0) \leftarrow$ less_than_val($apopt$, 700), decrease($alive$), not $g(1,2)$.
$g(1,2) \leftarrow$ less_than_att($necr$, $alive$).
$g(1,2) \leftarrow$ less_than_val($alive$, 100), increase($apopt$).

Incremental ASAL (Scaling-Up)

MCTS approach

```
for max_iters do:
    Descent to best leaf SFA A
    Sample mini-batch D
    Add up to k D-optimal revisions of A as children
    Pick a child and "play" a sequence of revisions
    Evaluate on training set and propagate rewards
Return best SFA found
```



- **SFA revision:**
  - Same technique used for when learning from scratch.
  - Guards definitions in defeasible form.
  - Guards may be generalized (remove conditions), or specialized (add conditions).
  - New guards maybe added (possibly with addition of new states to the SFA)
  - Guards may be entirely removed (removing also "stranded" states)

Image from Browne, Cameron B., et al. A survey of monte carlo tree search methods. IEEE Transactions on Computational Intelligence and AI in games, 2012.

Proof of Concept Results

	Method	Batch F_1-score	MCTS F_1 / *iterations* 5	10	$\|States\|$	$\|Guards\|$	Grounding (min)	Solving (min)	Total (min)
(A)									
Bio	ASAL	**0.968**			**4**	**5**	1.8	7.2	7.2
	MCTS		0.910	0.962	**4**	7	**0.3**	**0.2**	**3.8**
Maritime	ASAL	**0.982**			**4**	**4**	2.7	12.6	12.6
	MCTS		0.740	0.980	**4**	**4**	**0.3**	**0.1**	**2.8**
Activities	ASAL	**0.788**			**6**	**8**	1.2	18	18
	MCTS		0.740	0.773	7	11	**0.1**	**0.8**	**4.6**
(B)									
Bio	MCTS		0.858	0.968	4	6	0.4	0.9	5.7
Maritime	MCTS		0.915	0.985	5	6	0.6	1.2	7.2
Activities	MCTS		0.740	0.778	7	12	0.2	1.4	7.8
(C)									
Bio	MCTS		0.85	**0.963**	4	6	0.34	0.9	5.3
	RPNI	0.702			13				**0.05**
	EDSM	0.722			12				**0.05**
BioLarge	MCTS		0.852	**0.97**	4	6	0.34	1.02	14.3
	RPNI	–	–	–	–	–	–	–	–
	EDSM	–	–	–	–	–	–	–	–

Table 4: Experimental results.

- Bio: 3-variate, seq length: 50, examples: ~ 650
- Maritime: 6-variate, seq length: 30, examples: ~ 5000
- Activities: 4-variate, seq length: 100, examples: ~ 250
- BioLarge: uni-variate, seq length: 50, examples: ~ 50K

- Comparable predictive performance for batch (ASAL) & incremental (MCTS) versions.
- MCTS scales to large datasets and outperforms classical automata learning algs.

Future Work

- **Paper:**
    - Katzouris N. & Paliouras G., *Answer Set Automata: A Learnable Pattern Specification Framework for Complex Event Recognition*, ECAI 2023
- **Code:**
    - https://github.com/nkatzz/asal

Current/future work:

- **Scalability:**
    - What happens if the task is hard at a mini-batch level?
    - Long sequences, n-variate input for large n…
- **Expressive power:**
    - Learning Register Automata for long-range, temporal relations. (Finished, not properly evaluated).
- **Neuro-symbolic (NeSy) approaches:**
    - NeSy training with given event patterns.
    - NeSy event pattern learning.