Logic Learning for Multi Agent Systems
ACAI-2019

Nikos Katzouris, Alexander Artikis and
Georgios Paliouras

Institute of Informatics & Telecommunications,
NCSR Demokritos

`http://cer.iit.demokritos.gr`

Presenter: Nikos Katzouris

Overview

- Part I: Dealing with Interacting Entities in Dynamic Domains.
  - Norm-governed systems.
  - Complex Event Recognition systems.
  - Logical specifications of domain dynamics.
  - The Event Calculus.
- Part II: Learning logical specifications of domain dynamics.
  - Basics of Logical & Relational Learning.
  - Learning with the Event Calculus.
  - Abductive-Inductive learning.
  - Learning from relational data streams.
- Part IV: Statistical Relational Learning.
  - Markov Logic Networks.
  - Online structure & weight learning.

Overview

- Part I: Dealing with Interacting Entities in Dynamic Domains.
  - Norm-governed systems.
  - Complex Event Recognition systems.
  - Logical specifications of domain dynamics.
  - The Event Calculus.
- Part II: Learning logical specifications of domain dynamics.
  - Basics of Logical & Relational Learning.
  - Learning with the Event Calculus.
  - Abductive-Inductive learning.
  - Learning from relational data streams.
- Part IV: Statistical Relational Learning.
  - Markov Logic Networks.
  - Online structure & weight learning.

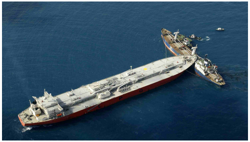
Dealing with Interacting Entities in Dynamic Domains

kdnuggets.com

Electronic markets

www.bankofengland.co.uk

Agent-based modeling



cer.iit.demokritos.gr

Intelligent monitoring & complex event detection

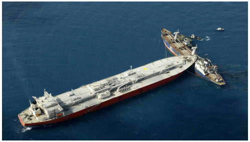
Dealing with Interacting Entities in Dynamic Domains

kdnuggets.com

Electronic markets

www.bankofengland.co.uk

Agent-based modeling



cer.iit.demokritos.gr

Intelligent monitoring & complex event detection

- Requirements:
  - Specifying & enforcing norms on entities' interactions.
  - Handling time & change.
    - Dealing with the effects of agent actions/event occurrences.
  - Verifying, tracing and explaining behavior.
  - Preventing failure & undesired behavior.

Example: Norm-governed Systems

- Pick a network:
  - individual people, forming online communities or social networks via computer-mediated communication
  - computing devices, forming ad hoc networks, Sensor Networks, etc.
  - business processes, forming virtual enterprises/organizations, computational economies, etc.
- Open systems:
  - autonomous components of heterogeneous provenance
  - can assume that components can communicate (i.e. a common language)
  - can *not* assume a common objective or central controller

Example: Norm-governed Systems

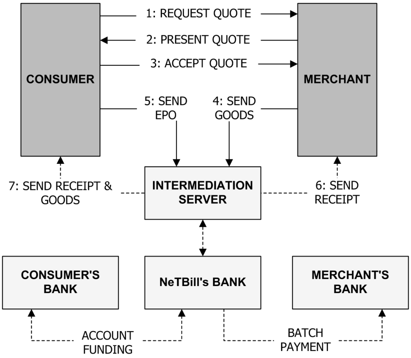
- Common features of open systems:
  - Dynamic and ‘volatile’: the environment, network topology and constituent nodes can vary rapidly and unpredictably
  - ‘Evolutionary’: known nodes can come/go, but can also have new nodes and node ‘death’
  - Co-dependence and internal competition: nodes need others to satisfy their own requirements, but may also behave to maximise individual (rather than collective) utility
  - Partial knowledge: no single source knowledge, union of knowledge may be inconsistent
  - Sub-ideal operation: the nodes may fail to comply according to the system specification, by accident, necessity, or design.

Example: Norm-governed Systems

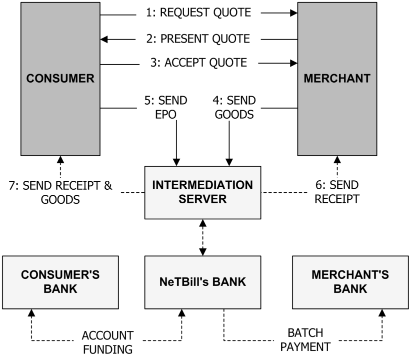
Addressing the features:

- Specify obligations and permissions, and describe agent behaviour as governed by norms, which may be violated, accidentally or on purpose.
- Predict, test, and verify the properties that hold if these norms are violated, and test the effectiveness of introducing proposed control, enforcement, and recovery mechanisms.
- Specify rules and procedures for adapting to changing environmental, financial and social conditions.

Example: Norm-governed Systems

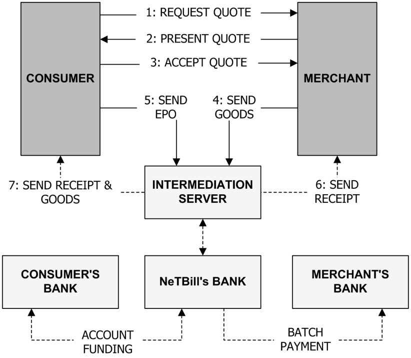


Example: Norm-governed Systems



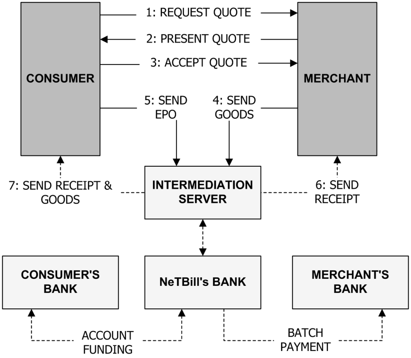
- Effects of actions.
  - A request is pending for as long as it has been issued by the consumer and has not been presented by the merchant.

Example: Norm-governed Systems



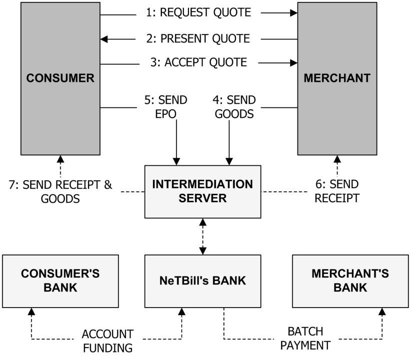
- Effects of actions.
  - A request is pending for as long as it has been issued by a consumer and has not been presented by the merchant.
  - $action_1 \rightarrow effect \xrightarrow{\text{holds until}} action_2$

Example: Norm-governed Systems



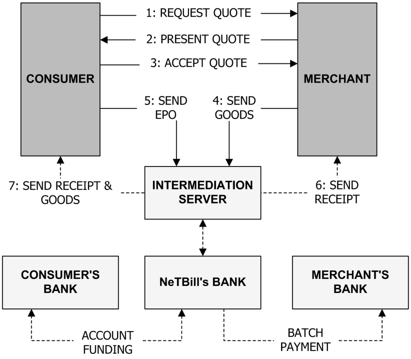
- Norms: Institutional power
  - A consumer is empowered to accept a quote if that quote was issued by a merchant, and the quote has not expired.

Example: Norm-governed Systems



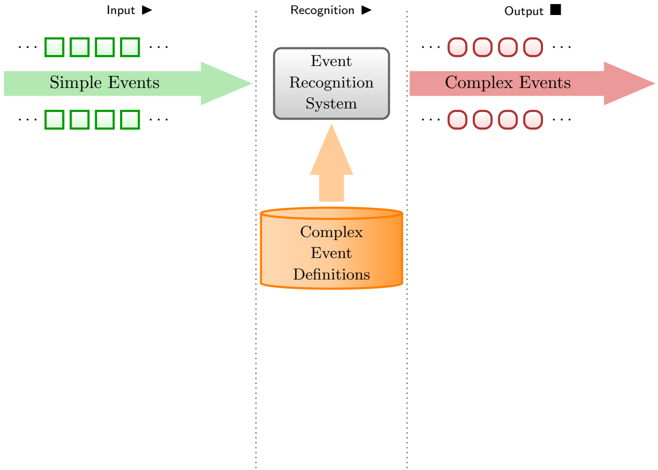
- Norms: Institutional power
  - A consumer is empowered to accept a quote for as long as that quote has been issued by a merchant, and the quote has not expired .
  - $\mathit{action}_1 \rightarrow \mathit{effect} \xrightarrow{\text{holds until}} \mathit{event\ occurrence}$

Example: Norm-governed Systems



- Norms: Obligation
  - A consumer is obliged to pay the agreed price to the contracting merchant by a specified deadline for as long as the consumer has accepted the quote, while being empowered to do so, and has not yet payed.
  - Similar durative cause-effect structure as before.

Example: Complex Event Recognition Systems



Example: Complex Event Recognition Systems

Maritime Surveillance

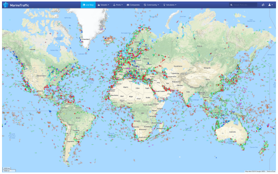
Input ▶ | Recognition ▶ | Output ■

Simple Events

Event Recognition System

Complex Event Definitions

Complex Events



www.marinetraffic.com

AIS signals

cer.iit.demokritos.gr

Suspicious vessel rendezvous

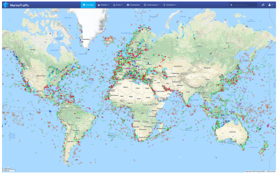
Example: Complex Event Recognition Systems

Maritime Surveillance

Input ▶ | Recognition ▶ | Output ■

Simple Events → Event Recognition System → Complex Events

Complex Event Definitions



www.marinetraffic.com

AIS signals

cer.iit.demokritos.gr

Suspicious vessel rendezvous

Two vessels are in suspicious rendezvous for as long as one sails close to another in the open sea and none sails away.

Example: Complex Event Recognition Systems

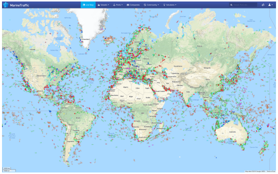
Maritime Surveillance

Input ▶ Recognition ▶ Output ■

Simple Events

Event Recognition System

Complex Events



www.marinetraffic.com

AIS signals

Complex Event Definitions

cer.iit.demokritos.gr

Suspicious vessel rendezvous

Two vessels are in suspicious rendezvous for as long as one sails close to another in the open sea and none sails away .

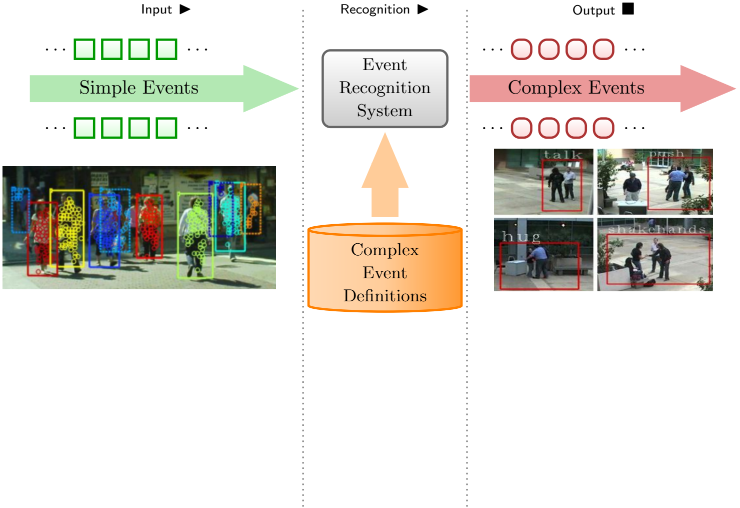
$$\textit{simple\_event\_occurrence}_1 \rightarrow \textit{effect}/\textit{complex\_event} \xrightarrow{\text{holds until}} \textit{simple\_event\_occurrence}_2$$

Example: Complex Event Recognition Systems

Maritime video

Example: Complex Event Recognition Systems

Actitivy recognition



Example: Complex Event Recognition Systems

Activity Recognition video.

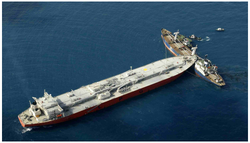
Logical Specifications of Domain Dynamics

kdnuggets.com

Electronic markets

www.bankofengland.co.uk

Agent-based modeling



cer.iit.demokritos.gr

Intelligent monitoring & complex event detection

- First-Order Logic (FOL):
  - Connections to action languages
    - Handling time & change.
  - Relational representations
    - Useful for dealing with multiple interacting entities.
  - Formal semantics
    - Useful for verifying, tracing & explaining behavior.
  - Allows arbitrarily complex background knowledge
  - Declarative, focus on what should a system do, not on how to do it.

Handling Time & Change

- Action languages:
  - representation of the agents' actions and their effects
  - exhibit a formal semantics
  - exhibit a declarative semantics
  - have direct routes to (efficient) implementation
- Examples:
  - Situation Calculus
  - Event Calculus
  - $C+$
- We will use the Event Calculus

The Event Calculus

- General purpose language for representing events, and for reasoning about effects of events.
- An action language with a logical semantics. Therefore, there are links to:
  - Implementation directly in Prolog.
  - Implementation in other programming languages.
- Prolog:
  - specification is its own implementation;
  - hence executable specification.
- We will use the Event Calculus for Run-Time reasoning (RTEC):
  - Highly efficient;
  - Sufficiently expressive.

Fluents and Events

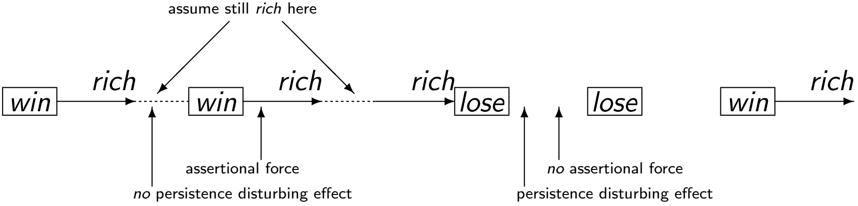
- Focus on events rather than situations; local states rather than global states
- Fluents
  - A fluent is a proposition whose value changes over time
  - A local state is a period of time during which a fluent holds continuously
- Events
  - *initiate* and *terminate* ...
  - ... a period of time during which a fluent holds continuously
- Example
  - $give(X, obj, Y)$ initiates $has(Y, obj)$
  - $give(X, obj, Y)$ terminates $has(X, obj)$

Event Calculus

- Fluents:
  - Values are assigned initially
  - Values are given when asserted (initiated)
  - Values persist until disturbed (terminated)
  - Otherwise we have ‘missing information’
- A formula of the form
  - *Event* terminates *fluent*
  - Has persistence disturbing effect, but no assertional force
- A formula of the form
  - *Event* initiates *fluent*
  - Has assertional force, but no persistence disturbing effect

Example

- $win\_lottery(X)$ initiates $rich(X)$
  - Winning the lottery initiates rich (but you might be rich already)
- $lose\_wallet(X)$ terminates $rich(X)$
  - Losing your wallet terminates rich (but you might not be rich when you lose it)



Event Calculus

- Actual syntax:

$$\textbf{initiatedAt}(\mathit{rich}(X) = \text{true},\ T) \leftarrow \\ \quad \textbf{happensAt}(\mathit{win\_lottery}(X),\ T)$$

$$\textbf{terminatedAt}(\mathit{rich}(X) = \text{true},\ T) \leftarrow \\ \quad \textbf{happensAt}(\mathit{lose\_wallet}(X),\ T)$$

- Given rules of the above form, the Event Calculus computes the maximal intervals for which a fluent has some value continuously. Eg:

  $\textbf{holdsFor}(\mathit{rich}(X) = \text{true},\ I)$

  where $I = [(S_1, E_1), \ldots, (S_n, E_n)]$

- Also: $\textbf{holdsAt}(\mathit{rich}(X) = \text{true},\ T)$ iff T in the list I of maximal intervals.

Event Calculus

- Sometimes it is easier to write the conditions in which a fluent has some value:

$$\begin{aligned}&\mathbf{holdsFor}(\mathit{happy}(X) = \mathsf{true},\ I) \leftarrow \\ &\qquad \mathbf{holdsFor}(\mathit{rich}(X) = \mathsf{true},\ I_1), \\ &\qquad \mathbf{holdsFor}(\mathit{loc}(X) = \mathit{pub},\ I_2), \\ &\qquad \mathbf{union\_all}([I_1, I_2],\ I)\end{aligned}$$

- How would you write the above rule in terms of **initiatedAt** and **terminatedAt**?

Event Calculus

- Short version:

$$\begin{aligned}&\textbf{holdsFor}(\mathit{happy}(X) = \mathsf{true},\ I) \leftarrow\\&\qquad \textbf{holdsFor}(\mathit{rich}(X) = \mathsf{true},\ I_1),\\&\qquad \textbf{holdsFor}(\mathit{loc}(X) = \mathit{pub},\ I_2),\\&\qquad \textbf{union\_all}([I_1, I_2],\ I)\end{aligned}$$

- Long version:

$$\begin{aligned}&\textbf{initiatedAt}(\mathit{happy}(X) = \mathsf{true},\ T) \leftarrow\\&\qquad \textbf{initiatedAt}(\mathit{rich}(X) = \mathsf{true},\ T)\end{aligned}$$

$$\begin{aligned}&\textbf{initiatedAt}(\mathit{happy}(X) = \mathsf{true},\ T) \leftarrow\\&\qquad \textbf{initiatedAt}(\mathit{loc}(X) = \mathit{pub},\ T)\end{aligned}$$

$$\begin{aligned}&\textbf{terminatedAt}(\mathit{happy}(X) = \mathsf{true},\ T) \leftarrow\\&\qquad \textbf{terminatedAt}(\mathit{rich}(X) = \mathsf{true},\ T),\\&\qquad \text{not } \textbf{holdsAt}(\mathit{loc}(X) = \mathit{pub},\ T)\end{aligned}$$

$$\begin{aligned}&\textbf{terminatedAt}(\mathit{happy}(X) = \mathsf{true},\ T) \leftarrow\\&\qquad \textbf{terminatedAt}(\mathit{loc}(X) = \mathit{pub},\ T),\\&\qquad \text{not } \textbf{holdsAt}(\mathit{rich}(X) = \mathsf{true},\ T)\end{aligned}$$

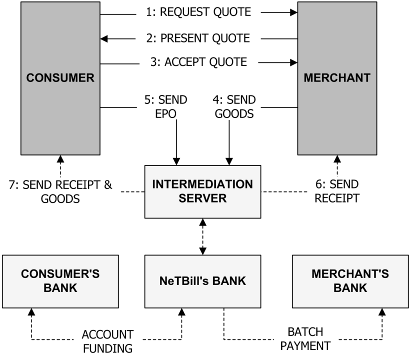
General Formulation

Predicate	Meaning
happensAt$(E,\ T)$	Event E is occurring at time T
initially$(F = V)$	The value of fluent F is V at time 0
initiatedAt$(F = V,\ T)$	At time T a period of time for which $F = V$ is initiated
terminatedAt$(F = V, T)$	At time T a period of time for which $F = V$ is terminated
holdsFor$(F = V,\ I)$	I is the list of the maximal intervals for which $F = V$ holds continuously
holdsAt$(F = V,\ T)$	The value of fluent F is V at time T

General Formulation (Interval Manipulation)

Predicate	Meaning
union_all$(L,\ I)$	I is the list of maximal intervals produced by the union of the lists of maximal intervals of list L
intersect_all$(L,\ I)$	I is the list of maximal intervals produced by the intersection of the lists of maximal intervals of list L
relative_complement_all$(I',\ L,\ I)$	I is the list of maximal intervals produced by the relative complement of the list of maximal intervals I' with respect to every list of maximal intervals of list L

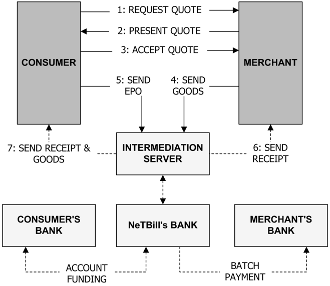
Examples Revisited



- A consumer is empowered to accept a quote for as long as that quote has been issued by a merchant, and the quote has not expired .

$$
\begin{aligned}
&\textbf{holdsFor}(\mathit{pow}(C, \mathit{accept\_quote}(C, M)) = \mathrm{true},\ I) \leftarrow \\
&\quad \textbf{holdsFor}(\mathit{role\_of}(C, \mathit{consumer}) = \mathrm{true},\ I_1), \\
&\quad \textbf{holdsFor}(\mathit{role\_of}(M, \mathit{merchant}) = \mathrm{true},\ I_2), \\
&\quad \textbf{holdsFor}(\mathit{quote}(M, C) = \mathrm{true},\ I_3), \\
&\quad \textbf{intersect\_all}([I_1, I_2, I_3],\ I)
\end{aligned}
$$

Examples Revisited



- A consumer is obliged to pay the agreed price to the contracting merchant by a specified deadline for as long as the consumer has accepted the quote, while being empowered to do so, and has not yet payed.

$$
\begin{aligned}
&\mathbf{initiatedAt}(obl(C, send\_EPO(C, IS)) = \text{true},\ \ T\ ) \leftarrow \\
&\quad \mathbf{happensAt}(accept\_quote(C, M),\ \ T), \\
&\quad \mathbf{holdsAt}(pow(C, accept\_quote(C, M)) = \text{true},\ \ T), \\
&\quad \mathbf{holdsAt}(role\_of(IS, iServer) = \text{true},\ \ T)
\end{aligned}
$$

$$
\begin{aligned}
&\mathbf{terminatedAt}(obl(C, send\_EPO(C, IS)) = \text{true},\ \ T) \leftarrow \\
&\quad \mathbf{happensAt}(send\_EPO(C, IS),\ \ T)
\end{aligned}
$$

Examples Revisited

Event Calculus as a Reasoning Engine

holdsAt$(F, T+1) \leftarrow$
  initiatedAt$F, T)$

holdsAtAt$(F, T+1) \leftarrow$
  holdsAt(F, T),
  not terminatedAt(F, T).

Very efficient inference: *Artikis et al. An Event Calculus for Event Recognition, TKDE, 2015.*

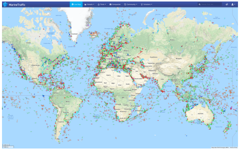
Input ▶ Recognition ▶ Output ■

Simple Events

Event Recognition System

Complex Events

Complex Event Definitions



AIS signals,
contextual info (ports, weather, environment...)

holdsAt$(rendezVous(vessel_1, vessel_2), 10)$
holdsAt$(rendezVous(vessel_1, vessel_2), 11)$
holdsAt$(rendezVous(vessel_1, vessel_2), 12)$
...

initiatedAt$(rendezVous(X, Y), T) \leftarrow$
  happensAt$(stopped(X), T)$,
  happensAt$(lowSpeed(Y), T)$,
  $farFromPorts(X, T)$,
  $farFromPorts(Y, T)$,
  $closeProximity(X, Y, T)$.

terminatedAt$(rendezVous(X, Y), T) \leftarrow$
  happensAt$(travelSpeed(X), T)$,
  $not\ closeProximity(X, Y, T)$.

Examples Revisited

Event Calculus as a Reasoning Engine

holdsAt(F, $T+1$) ←
initiatedAtF, T)

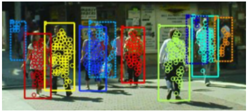
holdsAtAt(F, $T+1$) ←
holdsAt(F, T),
not terminatedAt(F, T).

Very efficient inference: *Artikis et al. An Event Calculus for Event Recognition, TKDE, 2015.*

Input ▶ | Recognition ▶ | Output ■

Simple Events → Event Recognition System → Complex Events

Complex Event Definitions



happensAt($active(id_0)$, 10)
holdsAt($coord(id_0, 20.88, 11.90)$, 10)
happensAt($active(id_1)$, 10)
holdsAt($coord(id_1, 22.34, 15.23)$, 10)
...

holdsAt($meet(id_0, id_1)$, 11)
holdsAt($meet(id_0, id_1)$, 12)
holdsAt($meet(id_0, id_1)$, 13)
...

initiatedAt($meet(X, Y)$, T) ←
happensAt($active(X)$, T),
happensAt($active(Y)$, T),
holdsAt($close(X, Y, 25)$, T).

terminatedAt($meet(X, Y)$, T) ←
happensAt($walking(X)$, T),
not holdsAt($close(X, Y, 25)$, T).

Summary: So far

- Introduced the Event Calculus
  - Events and Fluents.
  - Fluent formulations.
- Formulating domain dynamics specifications in the Event Calculus.
- Next: Learning such specifications from data.

Overview

- Part I: Dealing with Interacting Entities in Dynamic Domains.
  - Norm-governed systems.
  - Complex Event Recognition systems.
  - Logical specifications of domain dynamics.
  - The Event Calculus.
- Part II: Learning logical specifications of domain dynamics.
  - Basics of Logical & Relational Learning.
  - Learning with the Event Calculus.
  - Abductive-Inductive learning.
  - Learning from relational data streams.
- Part IV: Statistical Relational Learning.
  - Markov Logic Networks.
  - Online structure & weight learning.

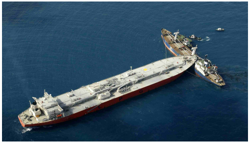
Learning Logical Specifications of Domain Dynamics

kdnuggets.com

Electronic markets

www.bankofengland.co.uk

Agent-based modeling

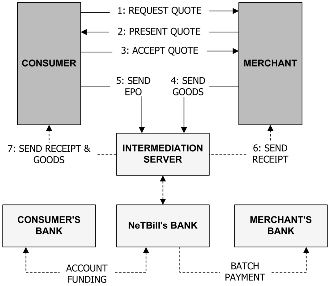


cer.iit.demokritos.gr

Intelligent monitoring & complex event detection

- Goal:
  - Given traces of a system's/domain's execution/evolution in time.
  - Learn the rules that govern the dynamics of the system/domain.

Learning Logical Specifications of Domain Dynamics



- Given past examples of transactions.
- Learn the underlying normative temporal rules.

$$\begin{aligned}
&\mathbf{initiatedAt}(\mathit{obl}(C, \mathit{send_EPO}(C, \mathit{IS})) = \text{true},\ \ T\ ) \leftarrow\\
&\quad \mathbf{happensAt}(\mathit{accept_quote}(C, M),\ \ T),\\
&\quad \mathbf{holdsAt}(\mathit{pow}(C, \mathit{accept_quote}(C, M)) = \text{true},\ \ T),\\
&\quad \mathbf{holdsAt}(\mathit{role_of}(\mathit{IS}, \mathit{iServer}) = \text{true},\ \ T)
\end{aligned}$$

$$\begin{aligned}
&\mathbf{terminatedAt}(\mathit{obl}(C, \mathit{send_EPO}(C, \mathit{IS})) = \text{true},\ \ T) \leftarrow\\
&\quad \mathbf{happensAt}(\mathit{send_EPO}(C, \mathit{IS}),\ \ T)
\end{aligned}$$

Learning Logical Specifications of Domain Dynamics

Event Calculus as a Reasoning Engine

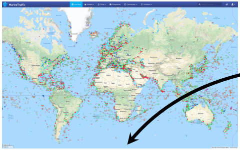
holdsAt(F, $T+1$) ←
    initiatedAtF, T)

holdsAtAt(F, $T+1$) ←
    holdsAt(F, T),
    not terminatedAt(F, T).

Very efficient inference: *Artikis et al. An Event Calculus for Event Recognition, TKDE, 2015.*

Input ▶ Recognition ▶ Output ■

Simple Events → Event Recognition System → Complex Events



AIS signals,
contextual info (ports, weather, environment...)

Complex Event Definitions

holdsAt($rendezVous(vessel_1, vessel_2)$, 10)
holdsAt($rendezVous(vessel_1, vessel_2)$, 11)
holdsAt($rendezVous(vessel_1, vessel_2)$, 12)
...

Learn this

initiatedAt($rendezVous(X, Y)$, T) ←
    happensAt($stopped(X)$, T),
    happensAt($lowSpeed(Y)$, T),
    $farFromPorts(X, T)$,
    $farFromPorts(Y, T)$,
    $closeProximity(X, Y, T)$.

terminatedAt($rendezVous(X, Y)$, T) ←
    happensAt($travelSpeed(X)$, T),
    $not\ closeProximity(X, Y, T)$.

From These

Learning Logical Specifications of Domain Dynamics

Event Calculus as a Reasoning Engine

holdsAt(F, $T+1$) $\leftarrow$
    initiatedAtF, T)

holdsAtAt(F, $T+1$) $\leftarrow$
    holdsAt(F, T),
    not terminatedAt(F, T).

Very efficient inference: *Artikis et al. An Event Calculus for Event Recognition, TKDE, 2015.*

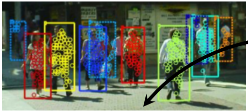
Input ▶ | Recognition ▶ | Output ■

Simple Events

Event Recognition System

Complex Events

Complex Event Definitions



happensAt($active(id_0)$, 10)
holdsAt($coord(id_0, 20.88, 11.90)$, 10)
happensAt($active(id_1)$, 10)
holdsAt($coord(id_1, 22.34, 15.23)$, 10)
...

holdsAt($meet(id_0, id_1)$, 11)
holdsAt($meet(id_0, id_1)$, 12)
holdsAt($meet(id_0, id_1)$, 13)
...

Learn this

initiatedAt($meet(X, Y)$, T) $\leftarrow$
    happensAt($active(X)$, T),
    happensAt($active(Y)$, T),
    holdsAt($close(X, Y, 25)$, T).

terminatedAt($meet(X, Y)$, T) $\leftarrow$
    happensAt($walking(X)$, T),
    not holdsAt($close(X, Y, 25)$, T).

From These

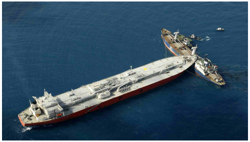
Learning Logical Specifications of Domain Dynamics

kdnuggets.com

Electronic markets

www.bankofengland.co.uk

Agent-based modeling



cer.iit.demokritos.gr

Intelligent monitoring & complex event detection

- Why?
  - The specifications may not always be known in advance.
  - Even if they are, manual authoring is time-consuming & error-prone.
  - The specifications are not static in principle.
    - They often change over time reflecting change in the domain.
    - Manual "tweaking" towards retaining performance is hard.
  - A learnt model can fit data properties that a human cannot foresee.
    - It is therefore likely to be more robust.
    - An initial, hand-crafted set of specifications may be improved by learning.

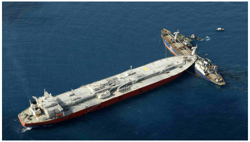
Learning Logical Specifications of Domain Dynamics

kdnuggets.com

Electronic markets

www.bankofengland.co.uk

Agent-based modeling



cer.iit.demokritos.gr

Intelligent monitoring & complex event detection

- How?
  - Logic-based approaches offer direct connections to machine learning
    - Inductive Logic Programming (ILP).
    - Statistical Relational Learning.
  - Arbitrarily complex background knowledge easy to incorporate into the learning task.
  - Interpretable models.

Overview

- Part I: Dealing with Interacting Entities in Dynamic Domains.
  - Norm-governed systems.
  - Complex Event Recognition systems.
  - Logical specifications of domain dynamics.
  - The Event Calculus.
- Part II: Learning logical specifications of domain dynamics.
  - Basics of Logical & Relational Learning (ILP).
  - Then: Learning with the Event Calculus.
  - Abductive-Inductive learning.
  - Learning from relational data streams.
- Part IV: Statistical Relational Learning.
  - Markov Logic Networks.
  - Online structure & weight learning.

Inductive Logic Programming (ILP)

Deduction

$\forall x(man(x) \rightarrow mortal(x))$ $\qquad$ $man(socrates).$

$$mortal(socrates).$$

Slide adapted from Zelezny et al., *Taming the Complexity of Inductive Logic Programming*, SOFSEM 2010.

Inductive Logic Programming (ILP)

Induction

General $\qquad \forall x(man(x) \rightarrow mortal(x)) \qquad man(socrates).$

Specific $\qquad mortal(socrates).$

Slide adapted from Zelezny et al., *Taming the Complexity of Inductive Logic Programming*, SOFSEM 2010.

Inductive Logic Programming (ILP)

Induction = Generalization (logic) + Justification (statistics)

General

$\forall x(man(x) \rightarrow mortal(x))$ ⟵ $man(socrates)$.

Specific

$mortal(socrates)$.

Slide adapted from Zelezny et al., *Taming the Complexity of Inductive Logic Programming*, SOFSEM 2010.

Inductive Logic Programming (ILP)

Induction = Generalization (logic) + Justification (statistics)

General

Specific

$\forall x(man(x) \rightarrow mortal(x))$

$man(socrates).$
$man(plato).$

$mortal(socrates).$
$mortal(plato).$

Slide adapted from Zelezny et al., *Taming the Complexity of Inductive Logic Programming*, SOFSEM 2010.

Inductive Logic Programming (ILP)

Induction = Generalization (logic) + Justification (statistics)

General

Specific

$\forall x(man(x) \rightarrow mortal(x))$

$man(socrates).$
$man(plato).$
$man(aristotle).$

$mortal(socrates).$
$mortal(plato).$
$mortal(aristotle).$

Slide adapted from Zelezny et al., *Taming the Complexity of Inductive Logic Programming*, SOFSEM 2010.

Inductive Logic Programming (ILP)

Induction = Generalization (logic) + Justification (statistics)

General

$\forall \mathbf{x}(\mathbf{man}(\mathbf{x}) \rightarrow \mathbf{mortal}(\mathbf{x}))$

man(*socrates*).
man(*plato*).
man(*aristotle*).
...

Specific

mortal(*socrates*).
mortal(*plato*).
mortal(*aristotle*).
...

Slide adapted from Zelezny et al., *Taming the Complexity of Inductive Logic Programming*, SOFSEM 2010.

ILP: A Very Simple Example

```
bird(sparrow). bird(eagle).
plane(boeing). plane(airbus).
penguin(joe).
flies(sparrow). flies(airbus). flies(boeing). flies(eagle). not flies(joe).
entity(X) ← bird(X).
entity(X) ← plane(X).
bird(X) ← penguin(X).
```

ILP: A Very Simple Example

```
bird(sparrow). bird(eagle).
plane(boeing). plane(airbus).
penguin(joe).
flies(sparrow). flies(airbus). flies(boeing). flies(eagle). not flies(joe).
entity(X) ← bird(X).
entity(X) ← plane(X).
bird(X) ← penguin(X).
```

Find the simplest hypothesis consistent with the data, specifying when things fly.

ILP: A Very Simple Example

```
bird(sparrow). bird(eagle).
plane(boeing). plane(airbus).
penguin(joe).
flies(sparrow). flies(airbus). flies(boeing). flies(eagle). not flies(joe).
entity(X) ← bird(X).
entity(X) ← plane(X).
bird(X) ← penguin(X).
```

Find the simplest hypothesis consistent with the data, specifying when things fly.

```
flies(X) ← entity(X).
```

ILP: A Very Simple Example

```
bird(sparrow). bird(eagle).
plane(boeing). plane(airbus).
penguin(joe).
flies(sparrow). flies(airbus). flies(boeing). flies(eagle). not flies(joe).
entity(X) ← bird(X).
entity(X) ← plane(X).
bird(X) ← penguin(X).
```

Find the simplest hypothesis consistent with the data, specifying when things fly.

`flies(X) ← entity(X).`

- No good, entails all of the given facts for flying, but it also entails `flies(joe)` which is false.

ILP: A Very Simple Example

```
bird(sparrow). bird(eagle).
plane(boeing). plane(airbus).
penguin(joe).
flies(sparrow). flies(airbus). flies(boeing). flies(eagle). not flies(joe).
entity(X) ← bird(X).
entity(X) ← plane(X).
bird(X) ← penguin(X).
```

Find the simplest hypothesis consistent with the data, specifying when things fly.

```
flies(X) ← bird(X).
```

ILP: A Very Simple Example

```
bird(sparrow). bird(eagle).
plane(boeing). plane(airbus).
penguin(joe).
flies(sparrow). flies(airbus). flies(boeing). flies(eagle). not flies(joe).
entity(X) ← bird(X).
entity(X) ← plane(X).
bird(X) ← penguin(X).
```

Find the simplest hypothesis consistent with the data, specifying when things fly.

`flies(X) ← bird(X).`

- No good, entails some, but not all of the given facts for flying, and it does entail `flies(joe)` which is false.

ILP: A Very Simple Example

```
bird(sparrow). bird(eagle).
plane(boeing). plane(airbus).
penguin(joe).
flies(sparrow). flies(airbus). flies(boeing). flies(eagle). not flies(joe).
entity(X) ← bird(X).
entity(X) ← plane(X).
bird(X) ← penguin(X).
```

Find the simplest hypothesis consistent with the data, specifying when things fly.

```
flies(X) ← plane(X).
flies(X) ← bird(X) ∧ not penguin(X).
```

ILP: A Very Simple Example

```
bird(sparrow). bird(eagle).
plane(boeing). plane(airbus).
penguin(joe).
flies(sparrow). flies(airbus). flies(boeing). flies(eagle). not flies(joe).
entity(X) ← bird(X).
entity(X) ← plane(X).
bird(X) ← penguin(X).
```

Find the simplest hypothesis consistent with the data, specifying when things fly.

```
flies(X) ← plane(X).
flies(X) ← bird(X) ∧ not penguin(X).
```

- Consistent with the data. Can it be simplified?

ILP: A Very Simple Example

```
bird(sparrow). bird(eagle).
plane(boeing). plane(airbus).
penguin(joe).
flies(sparrow). flies(airbus). flies(boeing). flies(eagle). not flies(joe).
entity(X) ← bird(X).
entity(X) ← plane(X).
bird(X) ← penguin(X).
```

Find the simplest hypothesis consistent with the data, specifying when things fly.

```
flies(X) ← plane(X).
flies(X) ← bird(X) ∧ not penguin(X).
```

- Consistent with the data.Can it be simplified?

```
flies(X) ← entity(X) ∧ not penguin(X).
```

- OK!

ILP: A Very Simple Example

```
bird(sparrow). bird(eagle).
plane(boeing). plane(airbus).
penguin(joe).
flies(sparrow). flies(airbus). flies(boeing). flies(eagle). not flies(joe).
entity(X) ← bird(X).
entity(X) ← plane(X).
bird(X) ← penguin(X).
```

Find the simplest hypothesis consistent with the data, specifying when things fly.

```
flies(X) ← plane(X).
flies(X) ← bird(X) ∧ not penguin(X).
```

- Consistent with the data.Can it be simplified?

```
flies(X) ← entity(X) ∧ not penguin(X).
```

- OK!

- Simplicity $\Leftrightarrow$ over-fitting avoidance ("Occam's razor")

ILP

Given:

- Training data, positive & negative examples E^+, E^-.
  - E.g. traces of system execution/domain evolution in time, in the form of sets of logical atoms.

ILP

Given:

- Training data, positive & negative examples E^+, E^-.
  - E.g. traces of system execution/domain evolution in time, in the form of sets of logical atoms.
- Some background knowledge B.
  - Existing domain knowledge, Easily codified in logic.
  - Simplify learning, avoid learning known stuff from scratch.
  - May itself be revised/improved via learning.

ILP

Given:

- Training data, positive & negative examples E^+, E^-.
  - E.g. traces of system execution/domain evolution in time, in the form of sets of logical atoms.
- Some background knowledge B.
  - Existing domain knowledge, Easily codified in logic.
  - Simplify learning, avoid learning known stuff from scratch.
  - May itself be revised/improved via learning.
- A "covers" relation.
  - Logical entailment, $covers(hypothesis, example) \Leftrightarrow hypothesis \vDash example$

ILP

Given:

- Training data, positive & negative examples E^+, E^-.
  - E.g. traces of system execution/domain evolution in time, in the form of sets of logical atoms.
- Some background knowledge B.
  - Existing domain knowledge, Easily codified in logic.
  - Simplify learning, avoid learning known stuff from scratch.
  - May itself be revised/improved via learning.
- A "covers" relation.
  - Logical entailment, $covers(hypothesis, example) \Leftrightarrow hypothesis \vDash example$
- A hypothesis quality criterion Q.
  - E.g. a hyp. should cover all positives, no negatives.
  - ...or, as many pos. & as few negs as possible (more realistic).
  - Penalize complexity, prefer simpler hypotheses.

ILP

Given:

- Training data, positive & negative examples E^+, E^-.
  - E.g. traces of system execution/domain evolution in time, in the form of sets of logical atoms.
- Some background knowledge B.
  - Existing domain knowledge, Easily codified in logic.
  - Simplify learning, avoid learning known stuff from scratch.
  - May itself be revised/improved via learning.
- A "covers" relation.
  - Logical entailment, $covers(hypothesis, example) \Leftrightarrow hypothesis \models example$
- A hypothesis quality criterion Q.
  - E.g. a hyp. should cover all positives, no negatives.
  - ...or, as many pos. & as few negs as possible (more realistic).
  - Penalize complexity, prefer simpler hypotheses.
- A hypothesis language $\mathcal{L}$.
  - Simplify learning, avoid generating redundant hypotheses.

ILP

Given:

- Training data, positive & negative examples E^+, E^-.
  - E.g. traces of system execution/domain evolution in time, in the form of sets of logical atoms.
- Some background knowledge B.
  - Existing domain knowledge, Easily codified in logic.
  - Simplify learning, avoid learning known stuff from scratch.
  - May itself be revised/improved via learning.
- A "covers" relation.
  - Logical entailment, $covers(hypothesis, example) \Leftrightarrow hypothesis \vDash example$
- A hypothesis quality criterion Q.
  - E.g. a hyp. should cover all positives, no negatives.
  - ...or, as many pos. & as few negs as possible (more realistic).
  - Penalize complexity, prefer simpler hypotheses.
- A hypothesis language $\mathcal{L}$.
  - Simplify learning, avoid generating redundant hypotheses.

Find:

- A hypothesis $H \in \mathcal{L}$ which H is optimal according to Q.

ILP

Given:

- Training data, positive & negative examples E^+, E^-.
  - E.g. traces of system execution/domain evolution in time, in the form of sets of logical atoms.
- Some background knowledge B.
  - Existing domain knowledge, Easily codified in logic.
  - Simplify learning, avoid learning known stuff from scratch.
  - May itself be revised/improved via learning.
- A "covers" relation.
  - Logical entailment, $covers(hypothesis, example) \Leftrightarrow hypothesis \models example$
- A hypothesis quality criterion Q.
  - E.g. a hyp. should cover all positives, no negatives.
  - ...or, as many pos. & as few negs as possible (more realistic).
  - Penalize complexity, prefer simpler hypotheses.
- A hypothesis language $\mathcal{L}$.
  - Simplify learning, avoid generating redundant hypotheses.

Find:

- A hypothesis $H \in \mathcal{L}$ which H is optimal according to Q.
- ...or, at least a "good-enough" $H \in \mathcal{L}$.
  - (finding an optimal H is intractable in principle).

Examples & Background Knowledge
1st Set-up: Learning from Entailment

```
bird(sparrow). bird(eagle).
plane(boeing). plane(airbus).
penguin(joe).
flies(sparrow). flies(airbus). flies(boeing). flies(eagle). not flies(joe).
entity(X) ← bird(X).
entity(X) ← plane(X).
bird(X) ← penguin(X).
```

- Positive and negative examples are target concept instances.
- Everything else is background knowledge.
- This is the classical ILP setting.

Examples & Background Knowledge
2nd Set-up: Learning from Interpretations

$I_1 = \{\texttt{bird(sparrow)}, \texttt{flies(sparrow)}\}$, $I_2 = \{\texttt{bird(eagle)}, \texttt{flies(eagle)}\}$
$I_3 = \{\texttt{plane(boeing)}, \texttt{flies(boeing)}\}$, $I_4 = \{\texttt{plane(airbus)}, \texttt{flies(airbus)}\}$,
$I_5 = \{\texttt{penguin(joe)}, \texttt{not flies(joe)}\}$
$\texttt{entity(X)} \leftarrow \texttt{bird(X)}.$
$\texttt{entity(X)} \leftarrow \texttt{plane(X)}.$
$\texttt{bird(X)} \leftarrow \texttt{penguin(X)}.$

- Examples = interpretations = sets of true facts.
- Each interpretation contains a **full description** of the example.
  - **CWA assumed within the interpretation**.
    - So, e.g. I_5 is actually simply $\{\texttt{penguin(joe)}\}$.
- All information that intuitively belongs to the example, is represented in the example, not in the background knowledge.
- Background knowledge = domain knowledge.
  - General info concerning the domain, not specific examples.

Examples & Background Knowledge
2nd Set-up: Learning from Interpretations

$I_1 = \{\texttt{bird(sparrow)}, \texttt{flies(sparrow)}\}$, $I_2 = \{\texttt{bird(eagle)}, \texttt{flies(eagle)}\}$
$I_3 = \{\texttt{plane(boeing)}, \texttt{flies(boeing)}\}$, $I_4 = \{\texttt{plane(airbus)}, \texttt{flies(airbus)}\}$,
$I_5 = \{\texttt{penguin(joe)}, \texttt{not flies(joe)}\}$
$\texttt{entity(X)} \leftarrow \texttt{bird(X)}.$
$\texttt{entity(X)} \leftarrow \texttt{plane(X)}.$
$\texttt{bird(X)} \leftarrow \texttt{penguin(X)}.$

- Examples = interpretations = sets of true facts.
- Each interpretation contains a **full description** of the example.
  - **CWA assumed within the interpretation**.
    - So, e.g. I_5 is actually simply $\{\texttt{penguin(joe)}\}$.
- All information that intuitively belongs to the example, is represented in the example, not in the background knowledge.
- Background knowledge = domain knowledge.
  - General info concerning the domain, not specific examples.
- More efficient than Learning from Entailment.
- But also weaker, cannot learn recursive concepts, nor relations between examples.

Hypothesis Language

```
bird(sparrow). bird(eagle).
plane(boeing). plane(airbus).
penguin(joe).
flies(sparrow). flies(airbus). flies(boeing). flies(eagle). not flies(joe).
entity(X) ← bird(X).
entity(X) ← plane(X).
bird(X) ← penguin(X).
```

- Syntactic constraints on the acceptable hypotheses.
- Avoid constructing hypotheses that *we know* are useless for the current task.
  - `flies(X) ← bird(X).` potentially useful.
  - `plane(X) ← bird(X).` useless.

Hypothesis Language

```
bird(sparrow). bird(eagle).
plane(boeing). plane(airbus).
penguin(joe).
flies(sparrow). flies(airbus). flies(boeing). flies(eagle). not flies(joe).
entity(X) ← bird(X).
entity(X) ← plane(X).
bird(X) ← penguin(X).
```

- Syntactic constraints on the acceptable hypotheses.
- Avoid constructing hypotheses that *we know* are useless for the current task.
  - `flies(X) ← bird(X).` potentially useful.
  - `plane(X) ← bird(X).` useless.
- Mode declarations: Declarative directives for rule generation.
  - E.g. `head(flies(entity)). body(not penguin(entity)).`
  - Can be used to generate e.g. `flies(X) ← not penguin(X)` $\wedge$ `entity(X).`
  - Also, directives on how to variabilize rules, variable chaining, types of variables etc.

Learning & Search

- **Ideal:** Find the simplest theory in the given language $\mathcal{L}$ that along with the background knowledge B covers as many positives and as few negatives as possible.

Learning & Search

- **Ideal:** Find the simplest theory in the given language $\mathcal{L}$ that along with the background knowledge B covers as many positives and as few negatives as possible.
- Intractable: full search in space of theories generated by $\mathcal{L}$.
  - Doubly exponential in the size of $\mathcal{L}$.

Learning & Search

- **Ideal:** Find the simplest theory in the given language $\mathcal{L}$ that along with the background knowledge B covers as many positives and as few negatives as possible.
- Intractable: full search in space of theories generated by $\mathcal{L}$.
  - Doubly exponential in the size of $\mathcal{L}$.
- Typical ILP approaches:
  - Incremental rule learning strategies.
    - iteratively learn one "good" rule at a time until stopping criterion met.
    - Collection of such good rules approximates the learning objective.
  - Heuristic search strategies for learning a single rule.
    - Learning "the best possible" rule is also intractable.
    - Exponential search space (subsets of possible attributes).

The Set-Cover Loop

$\{\texttt{bird(sparrow)}, \texttt{flies(sparrow)}\}$
$\{\texttt{bird(eagle)}, \texttt{flies(eagle)}\}$
$\{\texttt{plane(boeing)}, \texttt{flies(boeing)}\}$
$\{\texttt{plane(airbus)}, \texttt{flies(airbus)}\}$
$\{\texttt{penguin(joe)}, \texttt{not flies(joe)}\}$

Learnt Theory:

The Set-Cover Loop

1. Select positive example.

{bird(sparrow), flies(sparrow)}
{bird(eagle), flies(eagle)}
{plane(boeing), flies(boeing)}
{plane(airbus), flies(airbus)}
{penguin(joe), not flies(joe)}

Learnt Theory:

The Set-Cover Loop

1. Select positive example.
2. Generate rule that covers the example.

$\{\texttt{bird(sparrow)}, \texttt{flies(sparrow)}\}$
$\{\texttt{bird(eagle)}, \texttt{flies(eagle)}\}$
$\{\texttt{plane(boeing)}, \texttt{flies(boeing)}\}$
$\{\texttt{plane(airbus)}, \texttt{flies(airbus)}\}$
$\{\texttt{penguin(joe)}, \texttt{not flies(joe)}\}$

Learnt Theory:
$\texttt{flies(boeing)} \leftarrow \texttt{plane(boeing)}.$

The Set-Cover Loop

1. Select positive example.
2. Generate rule that covers the example.
3. Generalize.

$\{\texttt{bird(sparrow)}, \texttt{flies(sparrow)}\}$
$\{\texttt{bird(eagle)}, \texttt{flies(eagle)}\}$
$\{\texttt{plane(boeing)}, \texttt{flies(boeing)}\}$
$\{\texttt{plane(airbus)}, \texttt{flies(airbus)}\}$
$\{\texttt{penguin(joe)}, \texttt{not flies(joe)}\}$

Learnt Theory:
$\texttt{flies(X)} \leftarrow \texttt{plane(X)}.$

The Set-Cover Loop

1. Select positive example.
2. Generate rule that covers the example.
3. Generalize.
4. Remove covered positives.

{bird(sparrow), flies(sparrow)}
{bird(eagle), flies(eagle)}
{plane(boeing), flies(boeing)}
{plane(airbus), flies(airbus)}
{penguin(joe), not flies(joe)}

Learnt Theory:
flies(X) $\leftarrow$ plane(X).

The Set-Cover Loop

1. Select positive example.
2. Generate rule that covers the example.
3. Generalize.
4. Remove covered positives.

$\{\texttt{bird(sparrow)}, \texttt{flies(sparrow)}\}$
$\{\texttt{bird(eagle)}, \texttt{flies(eagle)}\}$

$\{\texttt{penguin(joe)}, \texttt{not flies(joe)}\}$

Learnt Theory:
$\texttt{flies(X)} \leftarrow \texttt{plane(X)}.$

The Set-Cover Loop

1. Select positive example.
2. Generate rule that covers the example.
3. Generalize.
4. Remove covered positives.
5. Repeat until no positives left (or other stopping criterion).

$\{\texttt{bird(sparrow)}, \texttt{flies(sparrow)}\}$
$\{\texttt{bird(eagle)}, \texttt{flies(eagle)}\}$

$\{\texttt{penguin(joe)}, \texttt{not flies(joe)}\}$

Learnt Theory:
$\texttt{flies(X)} \leftarrow \texttt{plane(X)}.$

The Set-Cover Loop

1. Select positive example.
2. Generate rule that covers the example.
3. Generalize.
4. Remove covered positives.
5. Repeat until no positives left (or other stopping criterion).

$\{\texttt{bird(sparrow)}, \texttt{flies(sparrow)}\}$
$\{\texttt{bird(eagle)}, \texttt{flies(eagle)}\}$

$\{\texttt{penguin(joe)}, \texttt{not flies(joe)}\}$

Learnt Theory:
$\texttt{flies(X)} \leftarrow \texttt{plane(X)}.$

The Set-Cover Loop

1. Select positive example.
2. Generate rule that covers the example.
3. Generalize.
4. Remove covered positives.
5. Repeat until no positives left (or other stopping criterion).

$\{\texttt{bird(sparrow)}, \texttt{flies(sparrow)}\}$
$\{\texttt{bird(eagle)}, \texttt{flies(eagle)}\}$

$\{\texttt{penguin(joe)}, \texttt{not flies(joe)}\}$

Learnt Theory:
$\texttt{flies(X)} \leftarrow \texttt{plane(X)}.$
$\texttt{flies(sparrow)} \leftarrow \texttt{bird(sparrow)} \wedge \texttt{not penguin(sparrow)}.$

The Set-Cover Loop

1. Select positive example.
2. Generate rule that covers the example.
3. Generalize.
4. Remove covered positives.
5. Repeat until no positives left (or other stopping criterion).

$\{\texttt{bird(sparrow)}, \texttt{flies(sparrow)}\}$
$\{\texttt{bird(eagle)}, \texttt{flies(eagle)}\}$

$\{\texttt{penguin(joe)}, \texttt{not flies(joe)}\}$

Learnt Theory:
$\texttt{flies(X)} \leftarrow \texttt{plane(X)}.$
$\texttt{flies(X)} \leftarrow \texttt{bird(X)} \wedge \texttt{not penguin(X)}.$

The Set-Cover Loop

1. Select positive example.
2. Generate rule that covers the example.
3. Generalize.
4. Remove covered positives.
5. Repeat until no positives left (or other stopping criterion).

$\{\texttt{bird(sparrow)}, \texttt{flies(sparrow)}\}$
$\{\texttt{bird(eagle)}, \texttt{flies(eagle)}\}$

$\{\texttt{penguin(joe)}, \texttt{not flies(joe)}\}$

Learnt Theory:
$\texttt{flies(X)} \leftarrow \texttt{plane(X)}.$
$\texttt{flies(X)} \leftarrow \texttt{bird(X)} \wedge \texttt{not penguin(X)}.$

The Set-Cover Loop

1. Select positive example.
2. Generate rule that covers the example.
3. Generalize.
4. Remove covered positives.
5. Repeat until no positives left (or other stopping criterion).

$\{\texttt{penguin(joe)}, \texttt{not flies(joe)}\}$

Learnt Theory:
$\texttt{flies(X)} \leftarrow \texttt{plane(X)}.$
$\texttt{flies(X)} \leftarrow \texttt{bird(X)} \wedge \texttt{not penguin(X)}.$

The Set-Cover Loop

1. Select positive example.
2. Generate rule that covers the example.
3. Generalize.
4. Remove covered positives.
5. Repeat until no positives left (or other stopping criterion).

Stop!

Learnt Theory:
$\texttt{flies(X)} \leftarrow \texttt{plane(X)}.$
$\texttt{flies(X)} \leftarrow \texttt{bird(X)} \land \texttt{not penguin(X)}.$

Learning a Rule

- Heuristic search in a space ordered by generality.

Generality

- Generality relation
- A rule r_1 is more general that a rule r_2 if
  - $\{$examples covered by $r_2\} \subseteq \{$examples covered by $r_1\}$.
- Rule of thumb: "less constraint" rules are more general
  - Therefore they cover more examples.

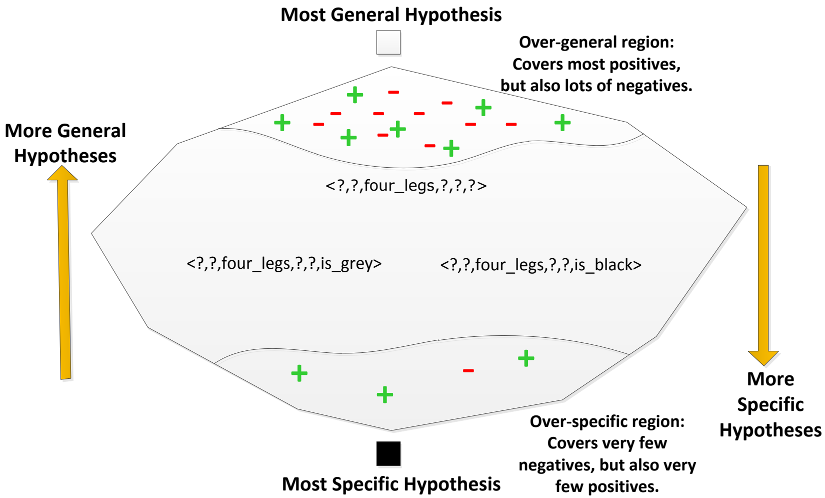
Generality

- Generality relation
- A rule r_1 is more general that a rule r_2 if
  - $\{$examples covered by $r_2\} \subseteq \{$examples covered by $r_1\}$.
- Rule of thumb: "less constraint" rules are more general
  - Therefore they cover more examples.
- For example, which one is more general?

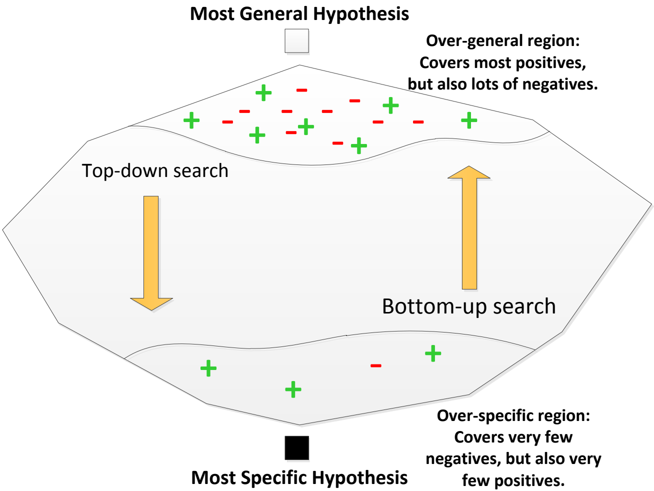
```
IF has_four_legs & has_whiskers THEN class = cat
IF has_four_legs & has_whiskers & is_grey THEN class = cat
```

Version Spaces

- Space ordered by generality.

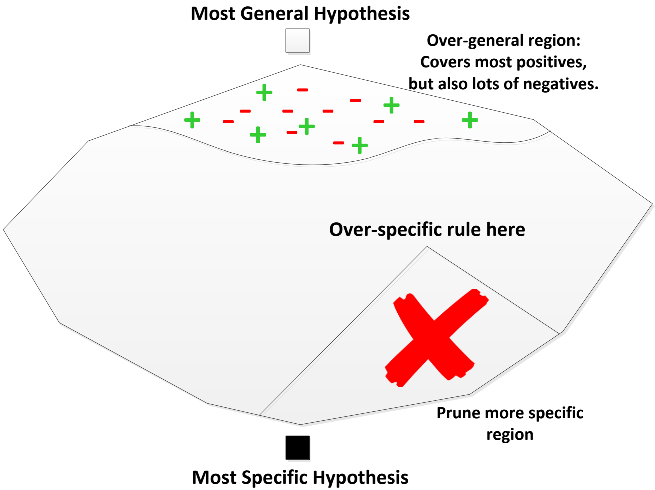


Version Spaces



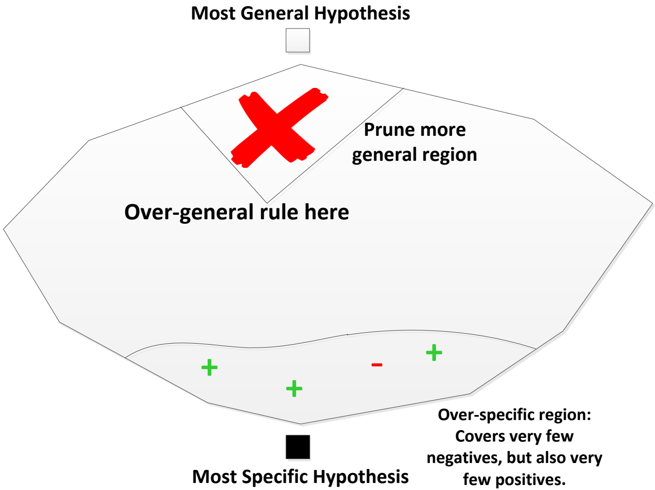
- Top-down search: Start from over-general, gradually specialize to exclude negatives.
- Bottom-up search: Start from over-specific, gradually generalize to cover positives.

Version Spaces



- Pruning heuristics
- If a rule is already over-specific (covers too few positives), there is no need to look into its more specific region.

Version Spaces



- Pruning heuristics
- If a rule is over-general (covers too many negatives), there is no need to look into its more general region.

ILP Basics: Learning & Search

- The version space model has some nice properties.
- Can we "upgrade" it to First-Order Logic?
- What do we need?

ILP Basics: Learning & Search

- The version space model has some nice properties.
- Can we "upgrade" it to First-Order Logic?
- What do we need?
  - A "covers" relation (logical entailment).
  - A generality notion based on that.

ILP Basics: Learning & Search

- The version space model has some nice properties.
- Can we "upgrade" it to First-Order Logic?
- What do we need?
  - A "covers" relation (logical entailment).
  - A generality notion based on that.
    - We have that:
    - A rule r_1 is more general than a rule r_2 is $covers(r_2) \subseteq covers(r_1) \Leftrightarrow r_1 \vDash r_2$.

ILP Basics: Learning & Search

- The version space model has some nice properties.
- Can we "upgrade" it to First-Order Logic?
- What do we need?
  - A "covers" relation (logical entailment).
  - A generality notion based on that.
    - We have that:
    - A rule r_1 is more general than a rule r_2 is $covers(r_2) \subseteq covers(r_1) \Leftrightarrow r_1 \models r_2$.
- Problem: $r_1 \models_? r_2$ for arbitrary r_1, r_2 is undecidable (even for Horn logic).

ILP Basics: Learning & Search

- The version space model has some nice properties.
- Can we "upgrade" it to First-Order Logic?
- What do we need?
  - A "covers" relation (logical entailment).
  - A generality notion based on that.
    - We have that:
    - A rule r_1 is more general than a rule r_2 is $covers(r_2) \subseteq covers(r_1) \Leftrightarrow r_1 \models r_2$.
- Problem: $r_1 \models_? r_2$ for arbitrary r_1, r_2 is undecidable (even for Horn logic).
- Use θ-subsumption as surrogate.

θ-subsumption

- Substitution $\theta = [X_1/t_1, \ldots, X_n/t_n]$: an assignment of terms t_i to variables X_i.
- A rule r_1 θ-subsumes a rule r_2 ($r_1 \preceq r_2$) iff
  $\exists\theta : [\mathit{head}(r_1)\theta = \mathit{head}(r_2)\ \&\ \mathit{body}(r_1)\theta \subseteq \mathit{body}(r_2)]$.

θ-subsumption

- Substitution $\theta = [X_1/t_1, \ldots, X_n/t_n]$: an assignment of terms t_i to variables X_i.
- A rule r_1 θ-subsumes a rule r_2 ($r_1 \preceq r_2$) iff
  $\exists\theta : [head(r_1)\theta = head(r_2)\ \&\ body(r_1)\theta \subseteq body(r_2)]$.

$r_1 = \texttt{flies(X)} \leftarrow \texttt{bird(X)}$
$r_2 = \texttt{flies(Y)} \leftarrow \texttt{bird(Y)} \wedge \texttt{not penguin(Y)}$
$r_3 = \texttt{flies(tweety)} \leftarrow \texttt{bird(tweety)} \wedge \texttt{not penguin(tweety)}$
Then:
$r_1 \preceq r_2$ with $\theta = [X/Y]$.
$r_1 \preceq r_3$ with $\theta = [X/tweety]$.

θ-subsumption

- Substitution $\theta = [X_1/t_1, \ldots, X_n/t_n]$: an assignment of terms t_i to variables X_i.
- A rule r_1 θ-subsumes a rule r_2 ($r_1 \preceq r_2$) iff
  $\exists\theta : [\mathit{head}(r_1)\theta = \mathit{head}(r_2)\ \&\ \mathit{body}(r_1)\theta \subseteq \mathit{body}(r_2)]$.

$r_1 = \texttt{flies(X)} \leftarrow \texttt{bird(X)}$
$r_2 = \texttt{flies(Y)} \leftarrow \texttt{bird(Y)} \wedge \texttt{not penguin(Y)}$
$r_3 = \texttt{flies(tweety)} \leftarrow \texttt{bird(tweety)} \wedge \texttt{not penguin(tweety)}$
Then:
$r_1 \preceq r_2$ with $\theta = [X/Y]$.
$r_1 \preceq r_3$ with $\theta = [X/\mathit{tweety}]$.

- A theory H_1 θ-subsumes a theory H_2 iff
  $\forall r_1 \in H_1, \exists r_2 \in H_2 : r_1 \preceq r_2$.

θ-subsumption

Properties of θ-subsumption:

- If $r_1 \preceq r_2$ then $r_1 \models r_2$.
  - *the inverse does not hold*.
- If $r_1 \preceq r_2$ then $covers(r_2) \subseteq covers(r_1)$.
- θ-subsumption is decidable (but still, NP-complete).

θ-subsumption

Properties of θ-subsumption:

- If $r_1 \preceq r_2$ then $r_1 \vDash r_2$.
  - *the inverse does not hold*.
- If $r_1 \preceq r_2$ then $covers(r_2) \subseteq covers(r_1)$.
- θ-subsumption is decidable (but still, NP-complete).

Therefore:

- θ-subsumption can be used as a surrogate for logical entailment.
- Version space (propositional) $\leftrightarrow$ subsumption lattice (relational).
- We essentially have the same search and pruning heuristics.

Learning a Rule

- Construct a search space (subsumption lattice).
  - Upper-bound: Most-general rule (rule with an empty body)
  - Lower-bound: Most-specific rule that covers a single example.
- Use search and pruning heuristics, along with a quality criterion (e.g. precision, recall, compression, info gain...) to find a "good" rule.

Learning a Rule: The subsumption Lattice

Top Clause ⊤: initiatedAt(*meet*(X, Y), T) $\leftarrow$

initiatedAt(*meet*(X, Y), T) $\leftarrow$
happensAt(*active*(X), T).

initiatedAt(*meet*(X, Y), T) $\leftarrow$
happensAt(*inactive*(Y), T).

...

initiatedAt(*meet*(X, Y), T) $\leftarrow$
holdsAt(*orientation*(X, Y, *45*), T).

Rest of clauses generated by adding one condition from ⊥ to the body of ⊤

initiatedAt(*meet*(X, Y), T) $\leftarrow$
happensAt(*active*(X), T),
happensAt(*inactive*(Y), T).

initiatedAt(*meet*(X, Y), T) $\leftarrow$
happensAt(*active*(X), T),
holdsAt(*close*(X, Y, *25*), T).

... ...

initiatedAt(*meet*(X, Y), T) $\leftarrow$
happensAt(*active*(X), T),
happensAt(*inactive*(Y), T),
holdsAt(*close*(X, Y, *25*), T).

... ...

... ...

Bottom Clause ⊥ :
initiatedAt(*meet*(X, Y), T) $\leftarrow$
happensAt(*active*(X), T), happensAt(*inactive*(Y), T), holdsAt(*close*(X, Y, *25*), T),
holdsAt(*close*(Y, X, *25*), T), *not* happensAt(*inactive*(X), T), *not* happensAt(*abrupt*(X), T),
not happensAt(*running*(X), T), happensAt(*inactive*(Y), T), *not* happensAt(*active*(Y), T),
not happensAt(*running*(Y), T), *not* happensAt(*abrupt*(Y), T), holdsAt(*orientation*(X, Y, *45*), T).

Learning a Rule: The subsumption Lattice

Top Clause ⊤: initiatedAt(*meet*(X, Y), T) ←

initiatedAt(*meet*(X, Y), T) ←
happensAt(*active*(X), T).

initiatedAt(*meet*(X, Y), T) ←
happensAt(*inactive*(Y), T).

...

initiatedAt(*meet*(X, Y), T) ←
holdsAt(*orientation*(X, Y, *45*), T).

Rest of clauses generated by adding one condition from ⊥ to the body of ⊤

initiatedAt(*meet*(X, Y), T) ←
happensAt(*active*(X), T),
happensAt(*inactive*(Y), T).

initiatedAt(*meet*(X, Y), T) ←
happensAt(*active*(X), T),
holdsAt(*close*(X, Y, *25*), T).

... ...

initiatedAt(*meet*(X, Y), T) ←
happensAt(*active*(X), T),
happensAt(*inactive*(Y), T),
holdsAt(*close*(X, Y, *25*), T).

...

... ...

Ordered by θ-subsumption $\preceq$:
If $r_1 \preceq r_2$ then covers$(r_2, e) \Rightarrow$ covers(r_1, e)
for all examples e

...

Bottom Clause ⊥ :
initiatedAt(*meet*(X, Y), T) ←
happensAt(*active*(X), T), happensAt(*inactive*(Y), T), holdsAt(*close*(X, Y, *25*), T),
holdsAt(*close*(Y, X, *25*), T), *not* happensAt(*inactive*(X), T), *not* happensAt(*abrupt*(X), T),
not happensAt(*running*(X), T), happensAt(*inactive*(Y), T), *not* happensAt(*active*(Y), T),
not happensAt(*running*(Y), T), *not* happensAt(*abrupt*(Y), T), holdsAt(*orientation*(X, Y, *45*), T).

Learning a Rule: Search

Positives not covered

Top Clause ⊤: initiatedAt(*meet*(*X*, *Y*), *T*) ←

initiatedAt(*meet*(*X*, *Y*), *T*) ← happensAt(*active*(*X*), *T*).

initiatedAt(*meet*(*X*, *Y*), *T*) ← happensAt(*inactive*(*Y*), *T*).

...

...

initiatedAt(*meet*(*X*, *Y*), *T*) ← happensAt(*active*(*X*), *T*), happensAt(*inactive*(*Y*), *T*).

initiatedAt(*meet*(*X*, *Y*), *T*) ← happensAt(*active*(*X*), *T*), holdsAt(*close*(*X*, *Y*, *25*), *T*).

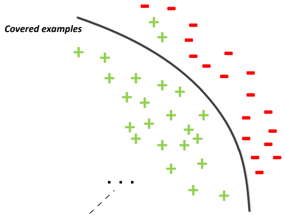
initiatedAt(*meet*(*X*, *Y*), *T*) ←
happensAt(*active*(*X*), *T*),
happensAt(*inactive*(*Y*), *T*),
holdsAt(*close*(*X*, *Y*, *25*), *T*).

...

...

...

...



Bottom Clause ⊥ : initiatedAt(*meet*(*X*, *Y*), *T*) ←
happensAt(*active*(*X*), *T*), happensAt(*inactive*(*Y*), *T*), holdsAt(*close*(*X*, *Y*, *25*), *T*),
holdsAt(*close*(*Y*, *X*, *25*), *T*), *not* happensAt(*inactive*(*X*), *T*), *not* happensAt(*abrupt*(*X*), *T*),
not happensAt(*running*(*X*), *T*), happensAt(*inactive*(*Y*), *T*), *not* happensAt(*active*(*Y*), *T*),
not happensAt(*running*(*Y*), *T*), *not* happensAt(*abrupt*(*Y*), *T*), holdsAt(*orientation*(*X*, *Y*, *45*), *T*).

Learning a Rule: Search

Generalization ↑

Top Clause ⊤: initiatedAt(*meet*(X, Y), T) ←

initiatedAt(*meet*(X, Y), T) ←
happensAt(*active*(X), T).

initiatedAt(*meet*(X, Y), T) ←
happensAt(*inactive*(Y), T).

. . .

. . .

initiatedAt(*meet*(X, Y), T) ←
happensAt(*active*(X), T),
happensAt(*inactive*(Y), T).

initiatedAt(*meet*(X, Y), T) ←
happensAt(*active*(X), T),
holdsAt(*close*(X, Y, *25*), T).

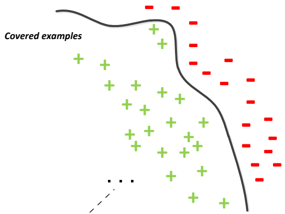
initiatedAt(*meet*(X, Y), T) ←
happensAt(*active*(X), T),
happensAt(*inactive*(Y), T),
holdsAt(*close*(X, Y, *25*), T).

. . .

. . .

. . .

. . .



Bottom Clause ⊥ : initiatedAt(*meet*(X, Y), T) ←
happensAt(*active*(X), T), happensAt(*inactive*(Y), T), holdsAt(*close*(X, Y, *25*), T),
holdsAt(*close*(Y, X, *25*), T), *not* happensAt(*inactive*(X), T), *not* happensAt(*abrupt*(X), T),
not happensAt(*running*(X), T), happensAt(*inactive*(Y), T), *not* happensAt(*active*(Y), T),
not happensAt(*running*(Y), T), *not* happensAt(*abrupt*(Y), T), holdsAt(*orientation*(X, Y, *45*), T).

Learning a Rule: Search

Negatives covered

Top Clause ⊤: initiatedAt(*meet*(X, Y), T) ←

initiatedAt(*meet*(X, Y), T) ←
happensAt(*active*(X), T).

initiatedAt(*meet*(X, Y), T) ←
happensAt(*inactive*(Y), T).

. . .

. . .

initiatedAt(*meet*(X, Y), T) ←
happensAt(*active*(X), T),
happensAt(*inactive*(Y), T).

initiatedAt(*meet*(X, Y), T) ←
happensAt(*active*(X), T),
holdsAt(*close*(X, Y, *25*), T).

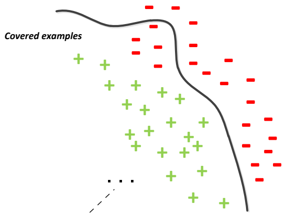
. . .

initiatedAt(*meet*(X, Y), T) ←
happensAt(*active*(X), T),
happensAt(*inactive*(Y), T),
holdsAt(*close*(X, Y, *25*), T).

. . .

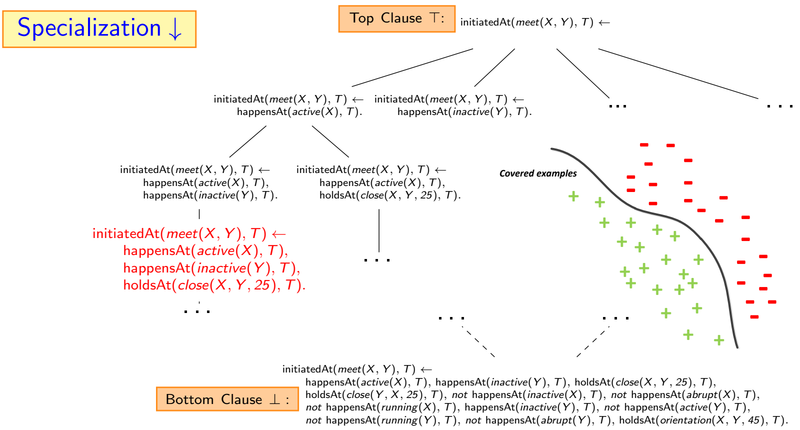
. . .

. . .

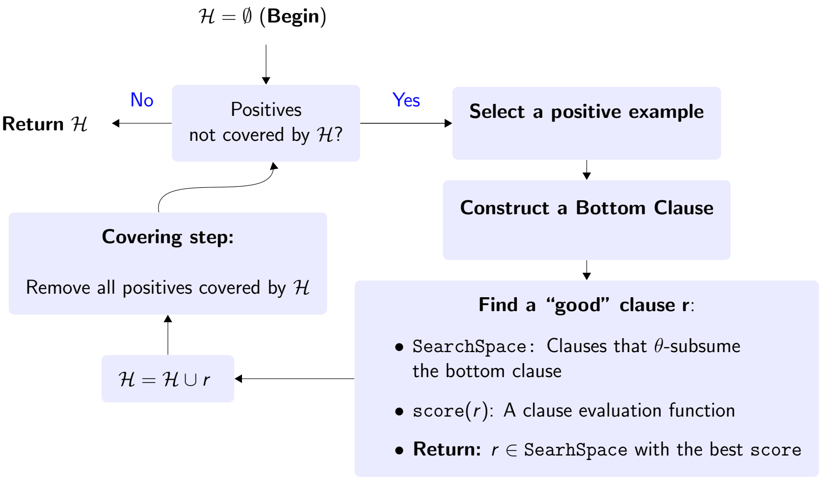


Bottom Clause ⊥ : initiatedAt(*meet*(X, Y), T) ←
happensAt(*active*(X), T), happensAt(*inactive*(Y), T), holdsAt(*close*(X, Y, *25*), T),
holdsAt(*close*(Y, X, *25*), T), *not* happensAt(*inactive*(X), T), *not* happensAt(*abrupt*(X), T),
not happensAt(*running*(X), T), happensAt(*inactive*(Y), T), *not* happensAt(*active*(Y), T),
not happensAt(*running*(Y), T), *not* happensAt(*abrupt*(Y), T), holdsAt(*orientation*(X, Y, *45*), T).

Learning a Rule: Search



Putting it All Together



ILP Basics: Summary

- We are given training examples representing true/false "snapshots" of the target concept.
- Some BK encoding things we know about the domain.
- Some hypothesis language.
- We use a "covers" relation based on logical entailment.
- and θ-subsumption as a notion of generality.
- Based on generality we build search spaces.
- Using a rule quality criterion, in addition to search and pruning heuristics we look for "good" rules within the search spaces.
- We learn one rule at a time until some stopping criterion is met.
  - For example, until we cover all positives, until we reach a maximum theory size etc.

Overview

- Part I: Dealing with Interacting Entities in Dynamic Domains.
  - Norm-governed systems.
  - Complex Event Recognition systems.
  - Logical specifications of domain dynamics.
  - The Event Calculus.
- Part II: Learning logical specifications of domain dynamics.
  - Basics of Logical & Relational Learning.
  - Learning with the Event Calculus.
  - Abductive-Inductive learning.
  - Learning from relational data streams.
- Part IV: Statistical Relational Learning.
  - Markov Logic Networks.
  - Online structure & weight learning.

Learning with the Event Calculus

- The classical ILP strategy works for
  - Learning a single concept.
  - Learning multiple **independent** concepts.
- This is not the case with the Event Calculus.
  - We want to learn theories of temporal specifications.
  - In the form of initiation & termination rules.
  - Which are **not** independent from each other.

Learning with the Event Calculus

- Inferences are non-monotonic:
  - Given: $\texttt{initiatedAt}(\texttt{event}_1, 10)$.
  - ? $\texttt{holdsAt}(\texttt{event}_1, 20)$.
    - yes.

Learning with the Event Calculus

- Inferences are non-monotonic:
  - Given: $\texttt{initiatedAt}(\texttt{event}_1, 10)$.
  - ? $\texttt{holdsAt}(\texttt{event}_1, 20)$.
    - yes.
  - Later we're also told that $\texttt{terminatedAt}(\texttt{event}_1, 15)$.

Learning with the Event Calculus

- Inferences are non-monotonic:
  - Given: $\texttt{initiatedAt}(\texttt{event}_1, 10)$.
  - ? $\texttt{holdsAt}(\texttt{event}_1, 20)$.
    - yes.
  - Later we're also told that $\texttt{terminatedAt}(\texttt{event}_1, 15)$.
  - As a result, we need to retract our previous answer.

Learning with the Event Calculus

- Inferences are non-monotonic:
  - Given: $\texttt{initiatedAt}(\texttt{event}_1, 10)$.
  - ? $\texttt{holdsAt}(\texttt{event}_1, 20)$.
    - yes.
  - Later we're also told that $\texttt{terminatedAt}(\texttt{event}_1, 15)$.
  - As a result, we need to retract our previous answer.
- As a result of non-monotonicity
  - Iterative (covering) rule learning techniques do not work.

Learning with the Event Calculus

- Inferences are non-monotonic:
  - Given: $\texttt{initiatedAt}(\texttt{event}_1, 10)$.
  - ? $\texttt{holdsAt}(\texttt{event}_1, 20)$.
    - yes.
  - Later we're also told that $\texttt{terminatedAt}(\texttt{event}_1, 15)$.
  - As a result, we need to retract our previous answer.
- As a result of non-monotonicity
  - Iterative (covering) rule learning techniques do not work.
  - They are based on the monotonicity assumption that adding new rules to a theory increases the example coverage of the theory.
  - Which is not the case, E.g. adding a termination rules reduces the example coverage of the theory.

Learning with the Event Calculus

Event Calculus as a Reasoning Engine

$\mathsf{holdsAt}(F,\, T+1) \leftarrow$
$\quad \mathsf{initiatedAt}F,\, T)$

$\mathsf{holdsAtAt}(F,\, T+1) \leftarrow$
$\quad \mathsf{holdsAt}(F,\, T),$
$\quad \mathit{not}\ \mathsf{terminatedAt}(F,\, T).$

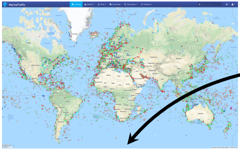
Very efficient inference: *Artikis et al. An Event Calculus for Event Recognition, TKDE, 2015.*

Input ▶ Recognition ▶ Output ■

· · · Simple Events · · ·

Event Recognition System

· · · Complex Events · · ·



AIS signals,

contextual info (ports, weather, environment...)

Complex Event Definitions

$\mathsf{holdsAt}(\mathit{rendezVous}(\mathit{vessel}_1, \mathit{vessel}_2), 10)$
$\mathsf{holdsAt}(\mathit{rendezVous}(\mathit{vessel}_1, \mathit{vessel}_2), 11)$
$\mathsf{holdsAt}(\mathit{rendezVous}(\mathit{vessel}_1, \mathit{vessel}_2), 12)$
. . .

Learning this

$\mathsf{initiatedAt}(\mathit{rendezVous}(X, Y), T) \leftarrow$
$\quad \mathsf{happensAt}(\mathit{stopped}(X), T),$
$\quad \mathsf{happensAt}(\mathit{lowSpeed}(Y), T),$
$\quad \mathit{farFromPorts}(X, T),$
$\quad \mathit{farFromPorts}(Y, T),$
$\quad \mathit{closeProximity}(X, Y, T).$

$\mathsf{terminatedAt}(\mathit{rendezVous}(X, Y), T) \leftarrow$
$\quad \mathsf{happensAt}(\mathit{travelSpeed}(X), T),$
$\quad \mathit{not}\ \mathit{closeProximity}(X, Y, T).$

From These

Learning with the Event Calculus

Event Calculus as a Reasoning Engine

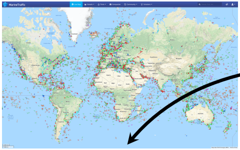
holdsAt(F, $T+1$) ←
  initiatedAtF, T)

holdsAtAt(F, $T+1$) ←
  holdsAt(F, T),
  not terminatedAt(F, T).

Very efficient inference: *Artikis et al. An Event Calculus for Event Recognition, TKDE, 2015.*

Input ▶ Recognition ▶ Output ■

Simple Events → Event Recognition System → Complex Events



AIS signals,
contextual info (ports, weather, environment...)

Complex Event Definitions

holdsAt($rendezVous(vessel_1, vessel_2), 10$)
holdsAt($rendezVous(vessel_1, vessel_2), 11$)
holdsAt($rendezVous(vessel_1, vessel_2), 12$)
...

Learn this

initiatedAt($rendezVous(X, Y), T$) ←
  happensAt($stopped(X), T$),
  happensAt($lowSpeed(Y), T$),
  $farFromPorts(X, T)$,
  $farFromPorts(Y, T)$,
  $closeProximity(X, Y, T)$.

terminatedAt($rendezVous(X, Y), T$) ←
  happensAt($travelSpeed(X), T$),
  not $closeProximity(X, Y, T)$.

From These

Learning with the Event Calculus

Event Calculus as a Reasoning Engine

holdsAt(F, $T+1$) $\leftarrow$
  initiatedAtF, T)

holdsAtAt(F, $T+1$) $\leftarrow$
  holdsAt(F, T),
  not terminatedAt(F, T).

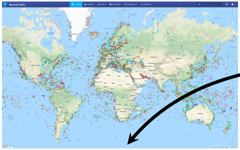
Very efficient inference: *Artikis et al. An Event Calculus for Event Recognition, TKDE, 2015.*

Input ▶ Recognition ▶ Output ■

Simple Events

Event Recognition System

Complex Events



AIS signals,

contextual info (ports, weather, environment...)

Complex Event

Ground Truth predicates differ from target predicates!

holdsAt($rendezVous(vessel_1, vessel_2), 10$)
holdsAt($rendezVous(vessel_1, vessel_2), 11$)
holdsAt($rendezVous(vessel_1, vessel_2), 12$)
...

Learn this

initiatedAt($rendezVous(X, Y)$, T) $\leftarrow$
  happensAt($stopped(X)$, T),
  happensAt($lowSpeed(Y)$, T),
  $farFromPorts(X, T)$,
  $farFromPorts(Y, T)$,
  $closeProximity(X, Y, T)$.

terminatedAt($rendezVous(X, Y)$, T) $\leftarrow$
  happensAt($travelSpeed(X)$, T),
  $not\ closeProximity(X, Y, T)$.

From These

Abductive Logic Programming (ALP)

- Abduction:
  - Hypothetical reasoning to the best explanation under incomplete information.
- ALP
  - Given:
    - A logic program H.
    - A set of observations O (logical facts).
    - A set of integrity constraints IC.
    - A set of predicate symbols A.
  - Find:
    - A set of *abductive explanations* $\Delta \subseteq A$ s.t. $H \cup \Delta \models O$ and $H \cup \Delta \cup IC$ is consistent.
  - Formal correspondence between NAF semantics and ALP.

ALP: A Simple Example

Observations:
$\texttt{holdsAt}(\texttt{meeting}(\texttt{id}_1, \texttt{id}_2), 10).$
$\texttt{holdsAt}(\texttt{meeting}(\texttt{id}_1, \texttt{id}_2), 11).$
$\texttt{not holdsAt}(\texttt{meeting}(\texttt{id}_1, \texttt{id}_2), 12).$
$\texttt{time}(1..20).$

H : The axioms of inertia (Event Calculus).

$A = \{\texttt{initiatedAt}/2, \texttt{terminatedAt}/2\}.$

$IC = \texttt{false} \leftarrow \texttt{initiatedAt}(\texttt{F}, \texttt{T}) \wedge \texttt{terminatedAt}(\texttt{F}, \texttt{T}).$

ALP: A Simple Example

Observations:
$\texttt{holdsAt}(\texttt{meeting}(\texttt{id}_1, \texttt{id}_2), 10).$
$\texttt{holdsAt}(\texttt{meeting}(\texttt{id}_1, \texttt{id}_2), 11).$
$\texttt{not holdsAt}(\texttt{meeting}(\texttt{id}_1, \texttt{id}_2), 12).$
$\texttt{time}(1..20).$

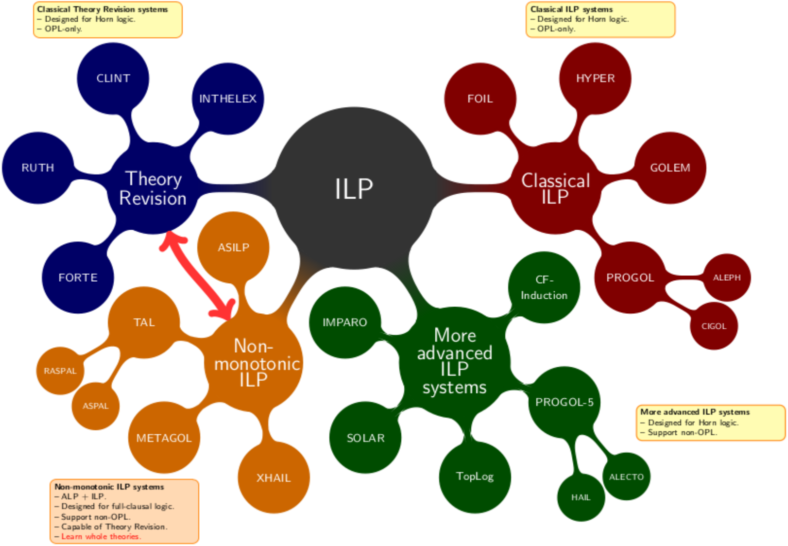
H : The axioms of inertia (Event Calculus).

$A = \{\texttt{initiatedAt}/2, \texttt{terminatedAt}/2\}.$

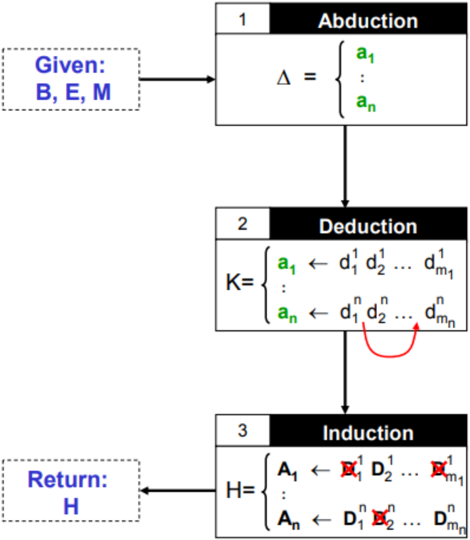
$IC = \texttt{false} \leftarrow \texttt{initiatedAt}(\texttt{F}, \texttt{T}) \wedge \texttt{terminatedAt}(\texttt{F}, \texttt{T}).$

$\Delta = \{\texttt{initiatedAt}(\texttt{meeting}(\texttt{id}_1, \texttt{id}_2), 9), \texttt{terminatedAt}(\texttt{meeting}(\texttt{id}_1, \texttt{id}_2), 11)\}.$

Non-monotonic Learning



Non-monotonic Learning (The XHAIL Algorithm)



Find a set of (head) atoms that abductively explain the positive and negative examples:

i.e. $B \wedge a_1 \wedge \ldots \wedge a_n \models E$

Find a set of (body) literals that deductively follow from the prior knowledge:

i.e. $B \models d_1^1 \wedge \ldots \wedge d_{m_n}^n$

Find a compressive theory that theta-subsumes the Kernel Set and explains the examples

i.e. $H \geq K$ and $B \wedge H \models E$

Slide from Ray O., *Using Abduction for Induction of Normal Logic Programs*, AIAI-2006.

The XHAIL Algorithm (Example)

Examples:

$\texttt{happensAt}(\texttt{walking}(\texttt{id}_1), 9)$
$\texttt{coords}(\texttt{id}_1, 201, 454, 9)$
$\texttt{holdsAt}(\texttt{moving}(\texttt{id}_1, \texttt{id}_2), 10)$
$\ldots$
$\texttt{happensAt}(\texttt{running}(\texttt{id}_5), 20)$
$\texttt{direction}(\texttt{id}_3, 270, 20)$
$\texttt{not holdsAt}(\texttt{moving}(\texttt{id}_3, \texttt{id}_5), 21)$
$\ldots$

Abduction:

$\texttt{initiatedAt}(\texttt{moving}(\texttt{id}_1, \texttt{id}_2), 9)$
$\texttt{terminatedAt}(\texttt{moving}(\texttt{id}_3, \texttt{id}_5), 20)$

The XHAIL Algorithm (Example)

Examples:

$\texttt{happensAt}(\texttt{walking}(\texttt{id}_1), 9)$
$\texttt{coords}(\texttt{id}_1, 201, 454, 9)$
$\texttt{holdsAt}(\texttt{moving}(\texttt{id}_1, \texttt{id}_2), 10)$
...
$\texttt{happensAt}(\texttt{running}(\texttt{id}_5), 20)$
$\texttt{direction}(\texttt{id}_3, 270, 20)$
$\texttt{not holdsAt}(\texttt{moving}(\texttt{id}_3, \texttt{id}_5), 21)$
...

Abduction:

$\texttt{initiatedAt}(\texttt{moving}(\texttt{id}_1, \texttt{id}_2), 9)$
$\texttt{terminatedAt}(\texttt{moving}(\texttt{id}_3, \texttt{id}_5), 20)$

Bottom Theory/Kernel Set:

$\texttt{initiatedAt}(\texttt{moving}(\texttt{id}_1, \texttt{id}_2), 9) \leftarrow$
$\quad \texttt{happensAt}(\texttt{walking}(\texttt{id}_1), 9),$
$\quad \texttt{happensAt}(\texttt{walking}(\texttt{id}_2), 9),$
$\quad \texttt{not happensAt}(\texttt{running}(\texttt{id}_1), 9),$
$\quad \texttt{not happensAt}(\texttt{active}(\texttt{id}_2), 9),$
$\quad \texttt{holdsAt}(\texttt{close}(\texttt{id}_1, \texttt{id}_2, 35), 9),$
$\quad \texttt{holdsAt}(\texttt{orientation}(\texttt{id}_1, \texttt{id}_2, 45), 9).$

$\texttt{terminatedAt}(\texttt{moving}(\texttt{id}_3, \texttt{id}_5), 20) \leftarrow$
$\quad \texttt{happensAt}(\texttt{walking}(\texttt{id}_3), 20),$
$\quad \texttt{happensAt}(\texttt{running}(\texttt{id}_5), 20),$
$\quad \texttt{not happensAt}(\texttt{abrupt}(\texttt{id}_1), 9),$
$\quad \texttt{not happensAt}(\texttt{active}(\texttt{id}_2), 9),$
$\quad \texttt{not holdsAt}(\texttt{close}(\texttt{id}_1, \texttt{id}_2, 35), 20),$
$\quad \texttt{not holdsAt}(\texttt{orientation}(\texttt{id}_1, \texttt{id}_2, 45), 20).$

The XHAIL Algorithm (Example)

Examples:

happensAt(walking(id_1), 9)
coords(id_1, 201, 454, 9)
holdsAt(moving(id_1, id_2), 10)
...
happensAt(running(id_5), 20)
direction(id_3, 270, 20)
not holdsAt(moving(id_3, id_5), 21)
...

Abduction:

initiatedAt(moving(id_1, id_2), 9)
terminatedAt(moving(id_3, id_5), 20)

Bottom Theory/Kernel Set:

initiatedAt(moving(X, Y), T) ←
  happensAt(walking(X), T),
  happensAt(walking(Y), T),
  holdsAt(close(X, Y, 35), T),
  not happensAt(running(X), T),
  not happensAt(active(Y), T),
  holdsAt(orientation(X, Y, 45), T).

terminatedAt(moving(X, Y), T) ←
  happensAt(walking(X), T),
  happensAt(running(Y), T),
  not happensAt(abrupt(X), T),
  not happensAt(active(Y), T),
  not holdsAt(close(X, Y, 35), T),
  not holdsAt(orientation(X, Y, 45), T).

The XHAIL Algorithm (Example)

Examples:

happensAt(walking(id_1), 9)
coords(id_1, 201, 454, 9)
holdsAt(moving(id_1, id_2), 10)
...
happensAt(running(id_5), 20)
direction(id_3, 270, 20)
not holdsAt(moving(id_3, id_5), 21)
...

Abduction:

initiatedAt(moving(id_1, id_2), 9)
terminatedAt(moving(id_3, id_5), 20)

Bottom Theory/Kernel Set:

initiatedAt(moving(X, Y), T) ←
happensAt(walking(X), T),
happensAt(walking(Y), T),
holdsAt(close(X, Y, 35), T),
~~not happensAt(running(X),T)~~,
~~not happensAt(active(Y),T)~~,
holdsAt(orientation(X, Y, 45), T).

terminatedAt(moving(X, Y), T) ←
~~happensAt(walking(X),T)~~,
happensAt(running(Y), T),
~~not happensAt(abrupt(X),T)~~,
~~not happensAt(active(Y),T)~~,
not holdsAt(close(X, Y, 35), T),
~~not holdsAt(orientation(X, Y, 45),T).~~

Hypothesis:

initiatedAt(moving(X, Y), T) ←
happensAt(walking(X), T),
happensAt(walking(X), T),
holdsAt(close(X, Y, 35), T),
holdsAt(orientation(X, Y, 45), T),

terminatedAt(moving(X, Y), T) ←
happensAt(running(Y), T),
not holdsAt(close(X, Y, 35), T).

Induction as Abductive Search

Bottom Theory/Kernel Set:

$\mathtt{initiatedAt(moving(X,Y),T)} \leftarrow$
$\quad \mathtt{happensAt(walking(X),T)},$
$\quad \mathtt{happensAt(walking(Y),T)},$
$\quad \mathtt{holdsAt(close(X,Y,35),T)},$
$\quad \cancel{\text{not happensAt(running(X),T)}},$
$\quad \cancel{\text{not happensAt(active(Y),T)}},$
$\quad \mathtt{holdsAt(orientation(X,Y,45),T)}.$

$\mathtt{terminatedAt(moving(X,Y),T)} \leftarrow$
$\quad \cancel{\text{happensAt(walking(X),T)}},$
$\quad \mathtt{happensAt(running(Y),T)},$
$\quad \cancel{\text{not happensAt(abrupt(X),T)}},$
$\quad \cancel{\text{not happensAt(active(Y),T)}},$
$\quad \mathtt{not\ holdsAt(close(X,Y,35),T)},$
$\quad \cancel{\text{not holdsAt(orientation(X, Y, 45),T).}}$

$\longrightarrow$

Hypothesis:

$\mathtt{initiatedAt(moving(X,Y),T)} \leftarrow$
$\quad \mathtt{happensAt(walking(X),T)},$
$\quad \mathtt{happensAt(walking(X),T)},$
$\quad \mathtt{holdsAt(close(X,Y,35),T)},$
$\quad \mathtt{holdsAt(orientation(X,Y,45),T)},$

$\mathtt{terminatedAt(moving(X,Y),T)} \leftarrow$
$\quad \mathtt{happensAt(running(Y),T)},$
$\quad \mathtt{not\ holdsAt(close(X,Y,35),T)}.$

Induction as Abductive Search

Bottom Theory/Kernel Set:

initiatedAt(moving(X, Y), T) ←
happensAt(walking(X), T),
happensAt(walking(Y), T),
holdsAt(close(X, Y, 35), T),
~~not happensAt(running(X), T)~~,
~~not happensAt(active(Y), T)~~,
holdsAt(orientation(X, Y, 45), T).

Hypothesis:

initiatedAt(moving(X, Y), T) ←
happensAt(walking(X), T),
happensAt(walking(X), T),
holdsAt(close(X, Y, 35), T),
X, Y, 45), T),

), T) ←
T),
, 35), T).

Replace the i-th rule $\alpha^i \leftarrow \delta_1^i, \ldots, \delta_n^i$ in the Kernel Set with the following program:

$\alpha^{\mathtt{i}} \leftarrow \mathtt{use(i, 0), try(i, 1,}\ \delta_{\mathtt{1}}^{\mathtt{i}}\mathtt{)}.$
$\mathtt{try(i, 1,}\ \delta_{\mathtt{1}}^{\mathtt{i}}\mathtt{)} \leftarrow \mathtt{not\ use(i, 1)}.$
$\mathtt{try(i, 1,}\ \delta_{\mathtt{1}}^{\mathtt{i}}\mathtt{)} \leftarrow \mathtt{use(i, 1),}\ \delta_{\mathtt{1}}^{\mathtt{i}}.$
...
$\alpha^i \leftarrow use(i, 0), try(i, n, \delta_n^i).$
$\mathtt{try(i, n,}\ \delta_{\mathtt{n}}^{\mathtt{i}}\mathtt{)} \leftarrow \mathtt{not\ use(i, n)}.$
$\mathtt{try(i, n,}\ \delta_{\mathtt{1}}^{\mathtt{i}}\mathtt{)} \leftarrow \mathtt{use(i, n),}\ \delta_{\mathtt{n}}^{\mathtt{i}}.$

Specify goal & solve (directly, with an Answer Set Solver!):

truePos(E) ← holdsAt(E, T), positiveExmpl(E).
falsePos(E) ← holdsAt(E, T), not positiveExmpl(E).
{use(I, J)} ← ruleId(I), literalId(J). *(Abduction via choice rule.)*
maximize{truePos/1}.
minimize{falsePos/1, use/2}.

H is obtained from the Kernel Set by removing every body atom $\delta_{\mathtt{i}}^{\mathtt{j}}$ for which the abducible use(i, j) is not in Δ, and removing every rule whose head atom $\mathtt{a_i}$ does not have a corresponding atom use(0, i) in Δ.

Theory Revision

- XHAIL:
  - Formal semantics.
  - Capable of learning multi-concept theories, recursive concepts etc.
  - Does not scale.

Theory Revision

- XHAIL:
  - Formal semantics.
  - Capable of learning multi-concept theories, recursive concepts etc.
  - Does not scale.
- The basic ideas behind XHAIL can be used for **Theory Revision (TR).**
  - Possible remedy for scalability issues.
- Data are presented in batches/chunks.
- A theory is constructed from the first batch.
- As new batches arrive the theory is continuously revised to account for the new observations.
  - Add new rules.
  - Remove existing (redundant) rules.
  - Specialize rules.
  - The TR process is guided by optimizing (locally, current batch only) example coverage + theory complexity.

Theory Revision

- XHAIL:
  - Formal semantics.
  - Capable of learning multi-concept theories, recursive concepts etc.
  - Does not scale.
- The basic ideas behind XHAIL can be used for **Theory Revision (TR).**
  - Possible remedy for scalability issues.
- Data are presented in batches/chunks.
- A theory is constructed from the first batch.
- As new batches arrive the theory is continuously revised to account for the new observations.
  - Add new rules.
  - Remove existing (redundant) rules.
  - Specialize rules.
  - The TR process is guided by optimizing (locally, current batch only) example coverage + theory complexity.
- How large should the batches be?
  - Larger batches mean more "globally-good" revisions, but also, harder to process.

Overview

- Part I: Dealing with Interacting Entities in Dynamic Domains.
  - Norm-governed systems.
  - Complex Event Recognition systems.
  - Logical specifications of domain dynamics.
  - The Event Calculus.
- Part II: Learning logical specifications of domain dynamics.
  - Basics of Logical & Relational Learning.
  - Learning with the Event Calculus.
  - Abductive-Inductive learning.
- Part III: Scalable learning.
  - Non-monotonic Theory Revision.
  - Learning from relational data streams.
- Part IV: Statistical Relational Learning.
  - Markov Logic Networks.
  - Online structure & weight learning.

Online Inductive Logic Programming

Challenge:

- Inductive Logic Programming algorithms are batch learners.
  - Each candidate in the search space is evaluated on the entire dataset.

Goal:

- Online learning:
  - Examples arrive in a stream.
  - Each example is “seen” once.

Approach:

- Make decisions from subsets of the stream:
  - Decisions are optimal “locally”.
  - Decisions are optimal “globally”...
    - within an error margin ϵ,
    - with probability $1-\delta$.

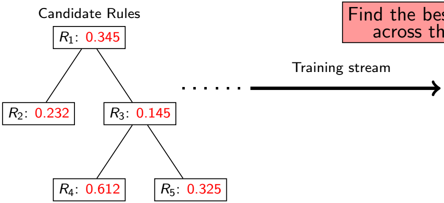
The Hoeffding Bound

- X is a random variable.
- $X_1, \ldots, X_N$ are N independent observations of X's values.
- Let $\bar{X}$ be the known, observed mean of X.
- Let $\hat{X}$ be the unknown, true mean of X.

The Hoeffding Bound

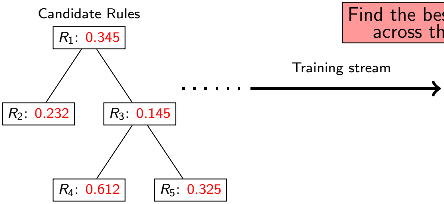
- X is a random variable.
- $X_1, \ldots, X_N$ are N independent observations of X's values.
- Let $\bar{X}$ be the known, observed mean of X.
- Let $\hat{X}$ be the unknown, true mean of X.
- Then:
  $\bar{X} - \epsilon \leq \hat{X} \leq \bar{X} + \epsilon$, with probability $1 - \delta$, where $\epsilon = \sqrt{\frac{ln(1/\delta)}{2N}}$

Online Rule Learning



Find the best candidate across the stream

Online Rule Learning

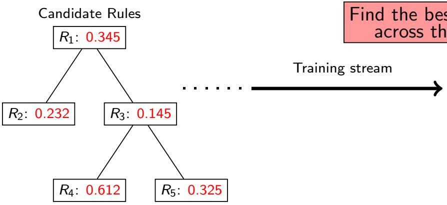


Find the best candidate across the stream

As examples stream in...

Monitor $\bar{X} = \overline{score}_{BestRule} - \overline{score}_{SecondBestRule}$

Online Rule Learning



Find the best candidate across the stream

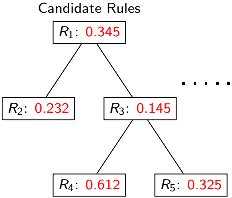
As examples stream in...

Monitor $\bar{X} = \overline{score}_{BestRule} - \overline{score}_{SecondBestRule}$

Continue until the number N of examples

makes $\bar{X} > \epsilon = \sqrt{\frac{ln(1/\delta)}{2N}}$

Online Rule Learning



Find the best candidate across the stream

Training stream

$\bar{X} - \epsilon \leq \hat{X} \leq \bar{X} + \epsilon$, where $\epsilon = \sqrt{\frac{ln(1/\delta)}{2N}}$

As examples stream in...

Monitor $\bar{X} = \overline{score}_{BestRule} - \overline{score}_{SecondBestRule}$

Continue until the number N of examples

makes $\bar{X} > \epsilon = \sqrt{\frac{ln(1/\delta)}{2N}}$

Then

$\bar{X} - \epsilon > 0 \Rightarrow$

$\hat{X} > 0 \Rightarrow$

BestRule is indeed the best rule,

with probability $1-\delta$.

Online Hill-Climbing

Top Clause $\top$: initiatedAt(*meet*(X, Y), T) $\leftarrow$

initiatedAt(*meet*(X, Y), T) $\leftarrow$ happensAt(*active*(X), T).

initiatedAt(*meet*(X, Y), T) $\leftarrow$ happensAt(*inactive*(Y), T).

...

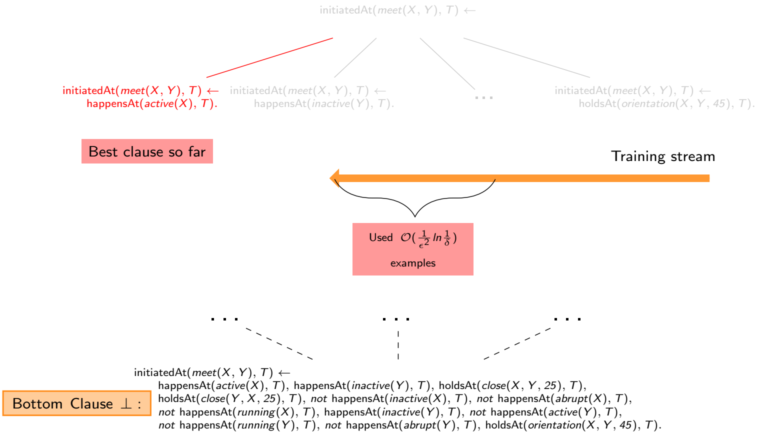
initiatedAt(*meet*(X, Y), T) $\leftarrow$ holdsAt(*orientation*(X, Y, *45*), T).

Training stream

... ...

Bottom Clause $\bot$: initiatedAt(*meet*(X, Y), T) $\leftarrow$
happensAt(*active*(X), T), happensAt(*inactive*(Y), T), holdsAt(*close*(X, Y, *25*), T),
holdsAt(*close*(Y, X, *25*), T), *not* happensAt(*inactive*(X), T), *not* happensAt(*abrupt*(X), T),
not happensAt(*running*(X), T), happensAt(*inactive*(Y), T), *not* happensAt(*active*(Y), T),
not happensAt(*running*(Y), T), *not* happensAt(*abrupt*(Y), T), holdsAt(*orientation*(X, Y, *45*), T).

Online Hill-Climbing



Online Hill-Climbing

initiatedAt(*meet*(*X*, *Y*), *T*) ←

initiatedAt(*meet*(*X*, *Y*), *T*) ←
happensAt(*active*(*X*), *T*).

initiatedAt(*meet*(*X*, *Y*), *T*) ←
happensAt(*inactive*(*Y*), *T*).

...

initiatedAt(*meet*(*X*, *Y*), *T*) ←
holdsAt(*orientation*(*X*, *Y*, *45*), *T*).

initiatedAt(*meet*(*X*, *Y*), *T*) ←
happensAt(*active*(*X*), *T*),
happensAt(*inactive*(*Y*), *T*).

initiatedAt(*meet*(*X*, *Y*), *T*) ←
happensAt(*active*(*X*), *T*),
holdsAt(*close*(*X*, *Y*, *25*), *T*).

Training stream

Used $\mathcal{O}(\frac{1}{\epsilon^2} ln \frac{1}{\delta})$

examples

... ...

Bottom Clause ⊥ :

initiatedAt(*meet*(*X*, *Y*), *T*) ←
happensAt(*active*(*X*), *T*), happensAt(*inactive*(*Y*), *T*), holdsAt(*close*(*X*, *Y*, *25*), *T*),
holdsAt(*close*(*Y*, *X*, *25*), *T*), *not* happensAt(*inactive*(*X*), *T*), *not* happensAt(*abrupt*(*X*), *T*),
not happensAt(*running*(*X*), *T*), happensAt(*inactive*(*Y*), *T*), *not* happensAt(*active*(*Y*), *T*),
not happensAt(*running*(*Y*), *T*), *not* happensAt(*abrupt*(*Y*), *T*), holdsAt(*orientation*(*X*, *Y*, *45*), *T*).

Online Hill-Climbing

initiatedAt(*meet*(X, Y), T) ←

initiatedAt(*meet*(X, Y), T) ← happensAt(*active*(X), T).

initiatedAt(*meet*(X, Y), T) ← happensAt(*inactive*(Y), T).

...

initiatedAt(*meet*(X, Y), T) ← holdsAt(*orientation*(X, Y, *45*), T).

initiatedAt(*meet*(X, Y), T) ← happensAt(*active*(X), T), happensAt(*inactive*(Y), T).

initiatedAt(*meet*(X, Y), T) ← happensAt(*active*(X), T), holdsAt(*close*(X, Y, *25*), T).

Best clause so far

Training stream

Used $\mathcal{O}(\frac{1}{\epsilon^2} \ln \frac{1}{\delta})$ examples

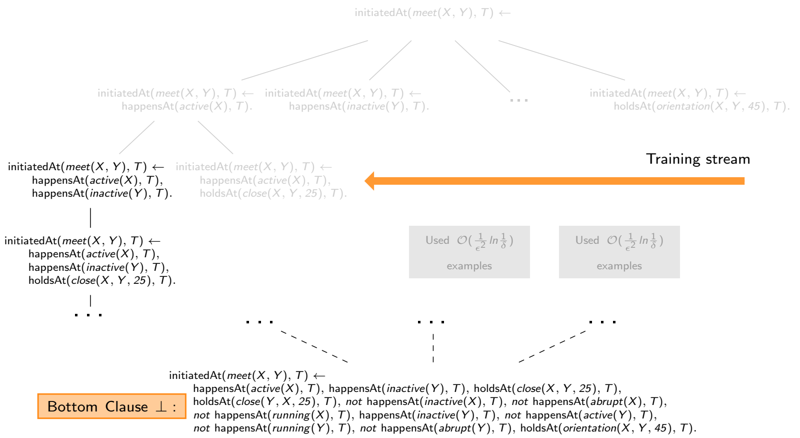
Used $\mathcal{O}(\frac{1}{\epsilon^2} \ln \frac{1}{\delta})$ examples

... ...

Bottom Clause $\perp$:

initiatedAt(*meet*(X, Y), T) ←
happensAt(*active*(X), T), happensAt(*inactive*(Y), T), holdsAt(*close*(X, Y, *25*), T),
holdsAt(*close*(Y, X, *25*), T), *not* happensAt(*inactive*(X), T), *not* happensAt(*abrupt*(X), T),
not happensAt(*running*(X), T), happensAt(*inactive*(Y), T), *not* happensAt(*active*(Y), T),
not happensAt(*running*(Y), T), *not* happensAt(*abrupt*(Y), T), holdsAt(*orientation*(X, Y, *45*), T).

Online Hill-Climbing



Features

Clause pruning:

- Often, “bad” clauses are constructed (e.g. from noisy examples).
- These are discarded when:
  - they do not “change” (get specialized) for sufficiently enough time
  - and their score is below a threshold.

Features

Clause pruning:

- Often, “bad” clauses are constructed (e.g. from noisy examples).
- These are discarded when:
  - they do not “change” (get specialized) for sufficiently enough time
  - and their score is below a threshold.

Warm-up period:

- Any-time algorithm
- In practice an output clause must have been evaluated on N_{min} examples.

Features

Clause pruning:

- Often, “bad” clauses are constructed (e.g. from noisy examples).
- These are discarded when:
  - they do not “change” (get specialized) for sufficiently enough time
  - and their score is below a threshold.

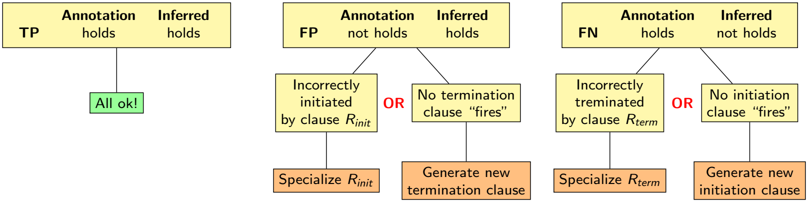
Warm-up period:

- Any-time algorithm
- In practice an output clause must have been evaluated on N_{min} examples.

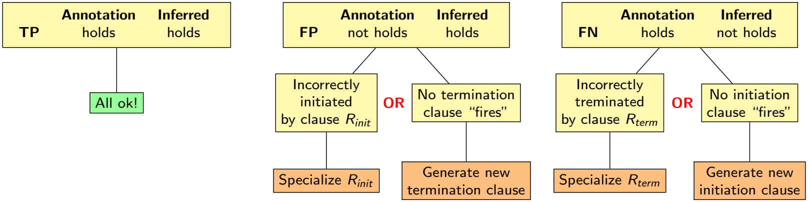
Tie breaking:

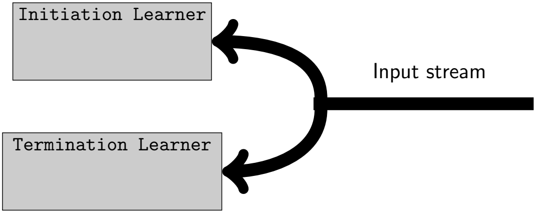
- When the best & the second-best clause have very similar scores...
- Break ties based on a pre-defined threshold, instead of waiting until the Hoeffding bound test succeeds.

Learning Sets of Clauses

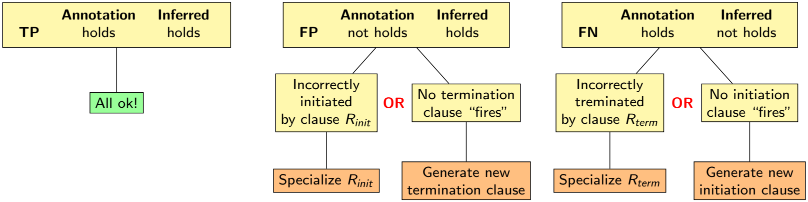


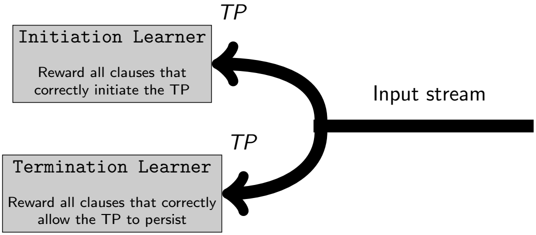
Learning Sets of Clauses



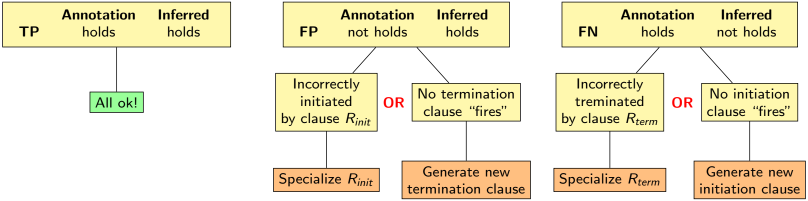


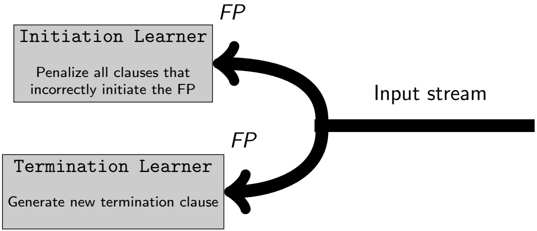
Learning Sets of Clauses



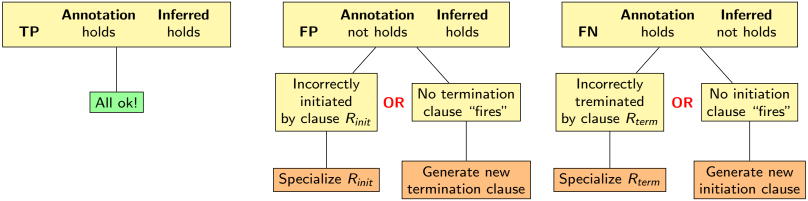


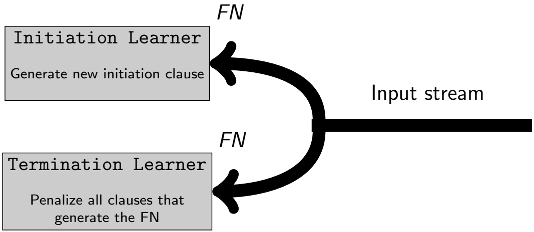
Learning Sets of Clauses



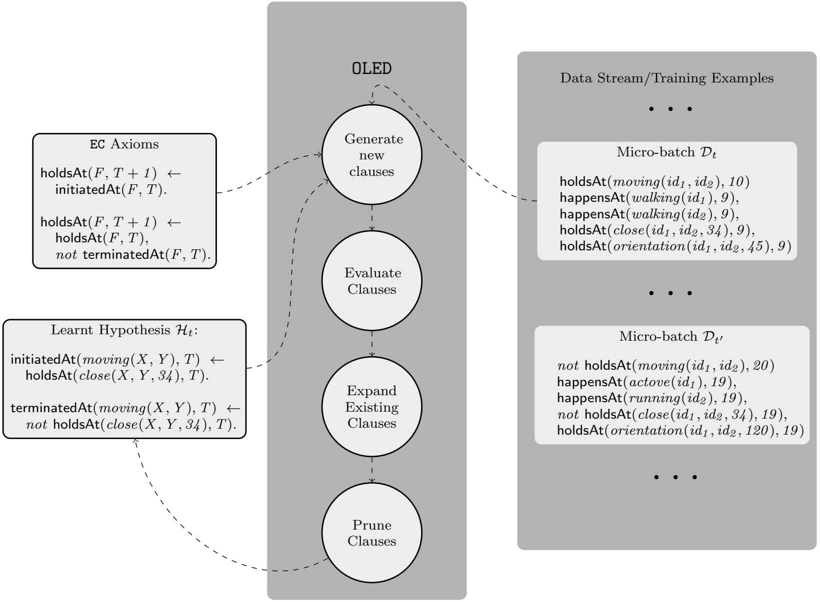


Learning Sets of Clauses





OLED


OLED
Generate new clauses
Evaluate Clauses
Expand Existing Clauses
Prune Clauses
EC Axioms
holdsAt$(F, T+1) \leftarrow$
initiatedAt(F, T).
holdsAt$(F, T+1) \leftarrow$
holdsAt(F, T),
not terminatedAt(F, T).
Learnt Hypothesis $\mathcal{H}_t$:
initiatedAt$(moving(X, Y), T) \leftarrow$
holdsAt$(close(X, Y, 34), T)$.
terminatedAt$(moving(X, Y), T) \leftarrow$
not holdsAt$(close(X, Y, 34), T)$.
Data Stream/Training Examples
Micro-batch $\mathcal{D}_t$
holdsAt$(moving(id_1, id_2), 10)$
happensAt$(walking(id_1), 9)$,
happensAt$(walking(id_2), 9)$,
holdsAt$(close(id_1, id_2, 34), 9)$,
holdsAt$(orientation(id_1, id_2, 45), 9)$
Micro-batch $\mathcal{D}_{t'}$
not holdsAt$(moving(id_1, id_2), 20)$
happensAt$(actove(id_1), 19)$,
happensAt$(running(id_2), 19)$,
not holdsAt$(close(id_1, id_2, 34), 19)$,
holdsAt$(orientation(id_1, id_2, 120), 19)$

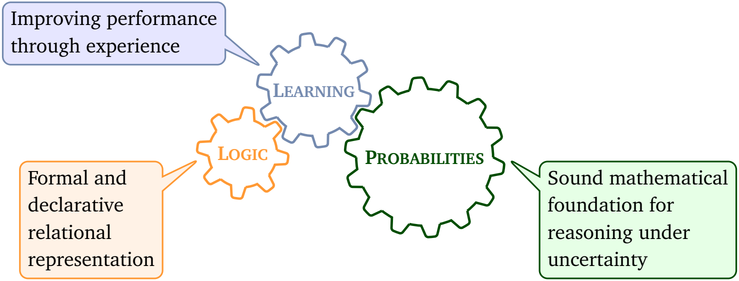
Comparison

	Method	F_1-score	Theory size	Time (sec)
Moving	EC_{crisp}	0.751	28	–
	EC_{MM}	**0.890**	28	1692
	XHAIL	0.841	**14**	7836
	OLED	0.812	34	**12**
Meeting	EC_{crisp}	0.762	23	–
	EC_{MM}	**0.863**	23	1133
	XHAIL	0.861	**15**	7248
	OLED	0.836	29	**23**

Overview

- Part I: Dealing with Interacting Entities in Dynamic Domains.
  - Norm-governed systems.
  - Complex Event Recognition systems.
  - Logical specifications of domain dynamics.
  - The Event Calculus.
- Part II: Learning logical specifications of domain dynamics.
  - Basics of Logical & Relational Learning.
  - Learning with the Event Calculus.
  - Abductive-Inductive learning.
- Part III: Scalable learning.
  - Non-monotonic Theory Revision.
  - Learning from relational data streams.
- Part IV: Statistical Relational Learning.
  - Markov Logic Networks.
  - Online structure & weight learning.

Statistical Relational Learning



ProbLog

- A probabilistic logic programming language.
- Allows for independent probabilistic facts prob::fact.
  - prob indicates the probability that fact is part of a possible world.
  - Facts are random variables.

ProbLog

- A probabilistic logic programming language.
- Allows for independent probabilistic facts prob::fact.
  - prob indicates the probability that fact is part of a possible world.
  - Facts are random variables.
- Rules are written as in classic Prolog.

ProbLog

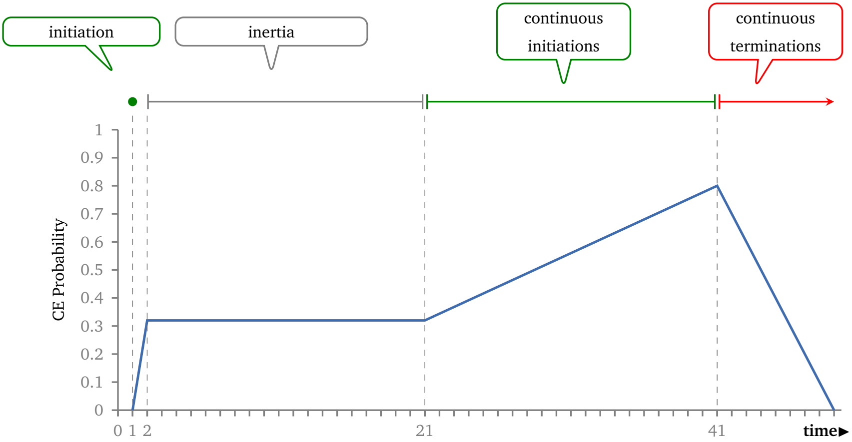
- A probabilistic logic programming language.
- Allows for independent probabilistic facts prob::fact.
  - prob indicates the probability that fact is part of a possible world.
  - Facts are random variables.
- Rules are written as in classic Prolog.
- The probability of a query q imposed on a ProbLog database *(success probability)* is computed by the following formula:

$$P_s(q) = P\left(\bigvee_{e \in Proofs(q)} \; \bigwedge_{f_i \in e} f_i \right)$$

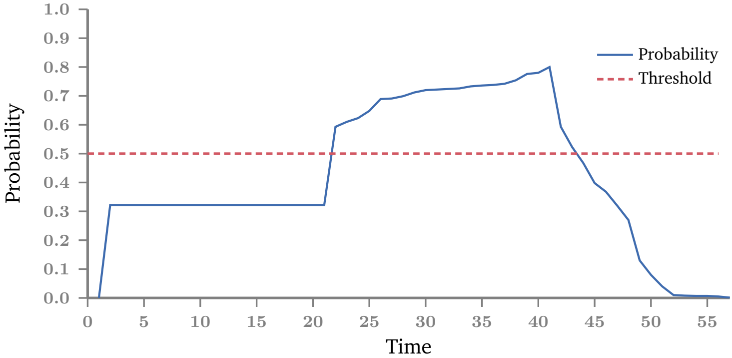
Event Recognition using ProbLog

Input	Output
340 0.45 :: $inactive(id_0)$	340 0.41 :: $left\_object(id_1, id_0)$
340 0.80 :: $p(id_0) = (20.88, -11.90)$	340 0.55 :: $moving(id_2, id_3)$
340 0.55 :: $appear(id_0)$	
340 0.15 :: $walking(id_2)$	
340 0.80 :: $p(id_2) = (25.88, -19.80)$	
340 0.25 :: $active(id_1)$	
340 0.66 :: $p(id_1) = (20.88, -11.90)$	
340 0.70 :: $walking(id_3)$	
340 0.46 :: $p(id_3) = (24.78, -18.77)$	

Event Calculus in ProbLog



Event Calculus in ProbLog



Markov Logic Networks (MLN)

Syntax: weighted first-order logic formulas (w_i, F_i)

When input events SDE_A **and** SDE_B occur at T,
then the output event CE is initiated:

3.18 $happensAt(SDE_A,\ T) \wedge happensAt(SDE_B,\ T) \Rightarrow initiatedAt(CE,\ T)$

Semantics: (w_i, F_i) represents a probability distribution over possible worlds

SDEs

weight of the i-th formula

$$P(Y{=}\mathbf{y} \mid X{=}\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \exp\left(\sum_i w_i\, n_i(\mathbf{x},\ \mathbf{y})\right)$$

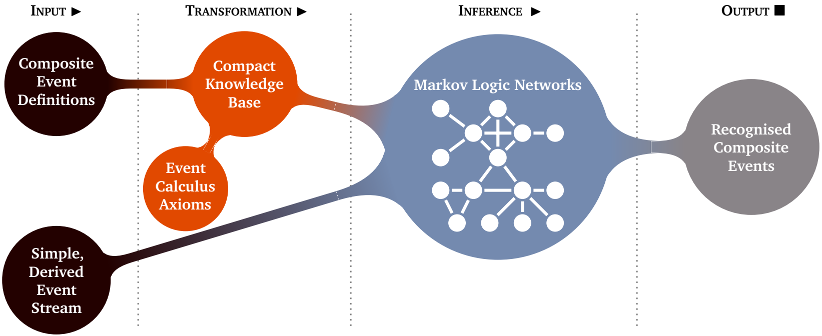
Possible world: CEs

Partition function

number of satisfied groundings

A world violating formulas becomes less probable, but not impossible!

Event Calculus in Markov Logic Networks (MLN-EC)



MLN-EC: Inertia

Marginal Inference:

- For all time points T, calculate the probability of each **CE** being true (recognised), given all **input SDEs** (evidence)

$$P(\texttt{holdsAt}(CE,\ T){=}true \mid SDEs)$$

- Marginal inference is #P-complete $\rightarrow$ approximate inference
- MC-SAT algorithm (Markov Chain Monte Carlo techniques with SAT solver)

MLN-EC: Inertia

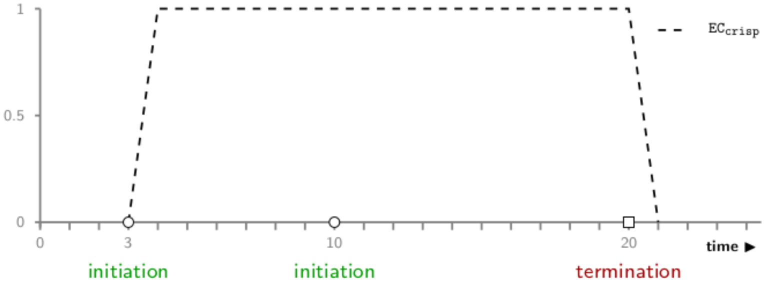
Maximum a Posteriori (MAP) Inference:

- Find the world with the highest probability
- **Input:** truth values for all **input SDEs** (evidence)
- **Output:** truth values of the **output CEs** that maximise the probability (recognition)

$$\underset{\mathtt{holdsAt}(CE,\,T)}{\operatorname{argmax}} \left(P(\mathtt{holdsAt}(CE,\ T) | SDEs) \right)$$

- MAP Inference is NP-hard $\rightarrow$ approximate inference
- Various methods: local search, linear programming, etc.

MLN-EC: Inertia



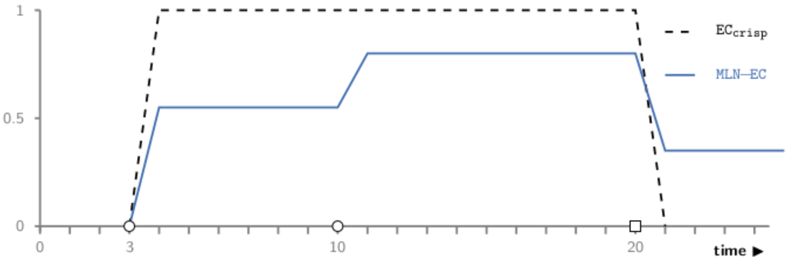
∞ $\text{holdsAt}(CE,\ T{+}1) \Leftarrow$
$\quad [\texttt{Initiation Conditions}]$

∞ $\neg\text{holdsAt}(CE,\ T{+}1) \Leftarrow$
$\quad \neg\text{holdsAt}(CE,\ T) \wedge$
$\quad \neg[\texttt{Initiation Conditions}]$

∞ $\neg\text{holdsAt}(CE,\ T{+}1) \Leftarrow$
$\quad [\texttt{Termination Conditions}]$

∞ $\text{holdsAt}(CE,\ T{+}1) \Leftarrow$
$\quad \text{holdsAt}(CE,\ T) \wedge$
$\quad \neg[\texttt{Termination Conditions}]$

MLN-EC: Inertia



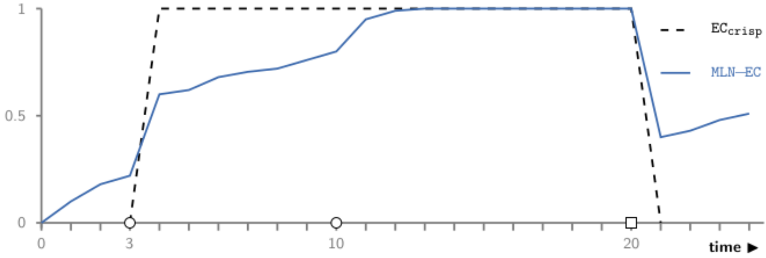
1.2 $\text{holdsAt}(CE,\ T{+}1) \Leftarrow$ [Initiation Conditions]

0.7 $\neg\text{holdsAt}(CE,\ T{+}1) \Leftarrow$ [Termination Conditions]

∞ $\neg\text{holdsAt}(CE,\ T{+}1) \Leftarrow$
$\neg\text{holdsAt}(CE,\ T) \land$
$\neg$[Initiation Conditions]

∞ $\text{holdsAt}(CE,\ T{+}1) \Leftarrow$
$\text{holdsAt}(CE,\ T) \land$
$\neg$[Termination Conditions]

MLN-EC: Inertia



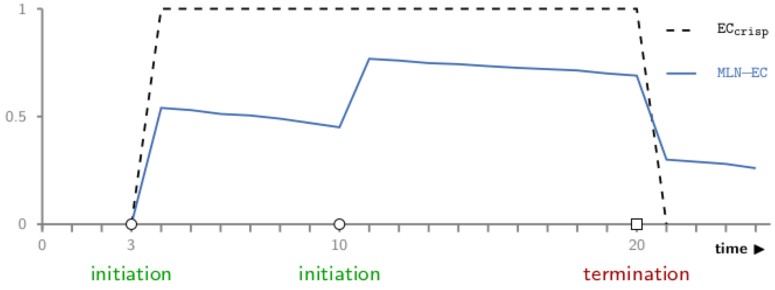
$1.2 \quad \text{holdsAt}(\text{CE},\ \text{T}{+}1) \Leftarrow$
$\quad [\texttt{Initiation Conditions}]$

$0.7 \quad \neg\text{holdsAt}(\text{CE},\ \text{T}{+}1) \Leftarrow$
$\quad [\texttt{Termination Conditions}]$

$2.3 \quad \neg\text{holdsAt}(\text{CE},\ \text{T}{+}1) \Leftarrow$
$\quad \neg\text{holdsAt}(\text{CE},\ \text{T}) \wedge$
$\quad \neg[\texttt{Initiation Conditions}]$

$\infty \quad \text{holdsAt}(\text{CE},\ \text{T}{+}1) \Leftarrow$
$\quad \text{holdsAt}(\text{CE},\ \text{T}) \wedge$
$\quad \neg[\texttt{Termination Conditions}]$

MLN-EC: Inertia



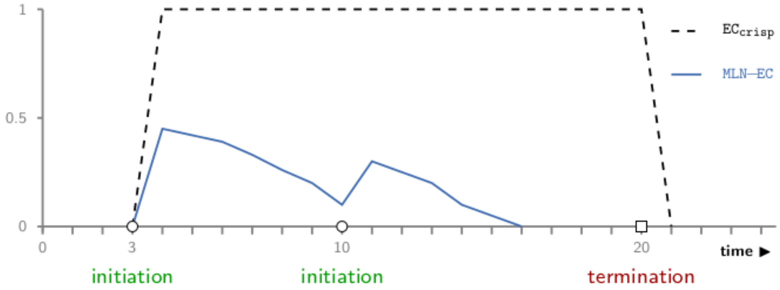
$1.2 \quad holdsAt(CE,\ T{+}1) \Leftarrow$
$\quad\quad [\texttt{Initiation Conditions}]$

$\infty \quad \neg holdsAt(CE,\ T{+}1) \Leftarrow$
$\quad\quad \neg holdsAt(CE,\ T) \land$
$\quad\quad \neg [\texttt{Initiation Conditions}]$

$0.7 \quad \neg holdsAt(CE,\ T{+}1) \Leftarrow$
$\quad\quad [\texttt{Termination Conditions}]$

$2.3 \quad holdsAt(CE,\ T{+}1) \Leftarrow$
$\quad\quad holdsAt(CE,\ T) \land$
$\quad\quad \neg [\texttt{Termination Conditions}]$

MLN-EC: Inertia



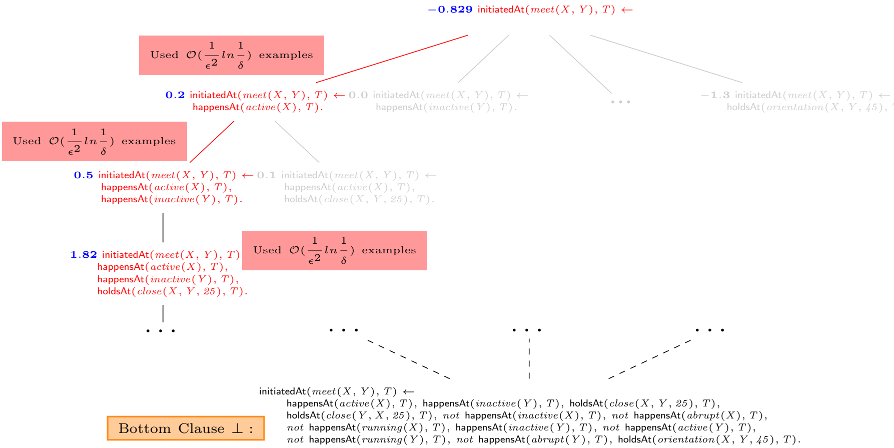
1.2 holdsAt(CE, T+1)⇐
[Initiation Conditions]

∞ ¬holdsAt(CE, T+1)⇐
¬holdsAt(CE, T) ∧
¬[Initiation Conditions]

0.7 ¬holdsAt(CE, T+1)⇐
[Termination Conditions]

0.6 holdsAt(CE, T+1)⇐
holdsAt(CE, T) ∧
¬[Termination Conditions]

Online Structure & Weight Learning in MLN



- Simultaneous structure & weight learning.
- Weight learning with AdaGrad.

The AdaGrad Weight Update Rule

Previous weight of the i-th rule

Learning rate

Rule's current mistakes

Regularization rate

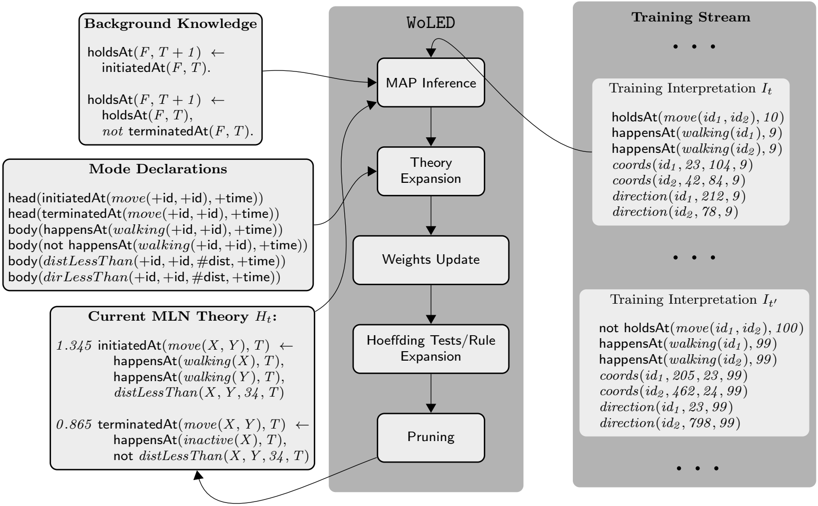
$$w_i^{t+1} = sign(w_i^t - \frac{\eta}{C_i^t}\Delta g_i^t)\ max\{0, |w_i^t - \frac{\eta}{C_i^t}\Delta g_i^t| - \lambda\frac{\eta}{C_i^t}\}$$

Current weight of the i-th rule

Term proportional to the rule's accumulated past mistakes

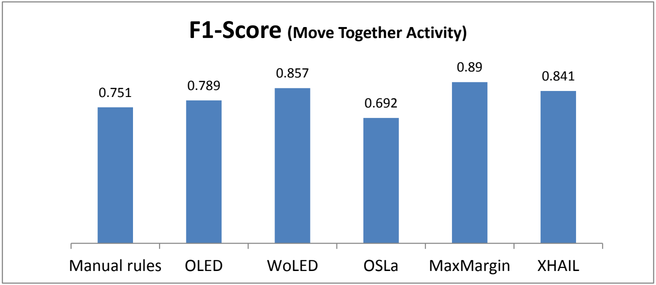
- Δg_i^t (i-th rule's mistakes at time t): difference in rule's true groundings in the true state and the MAP-inferred state.

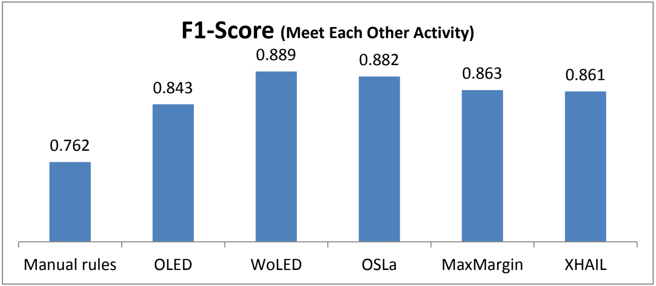
OLED-MLN



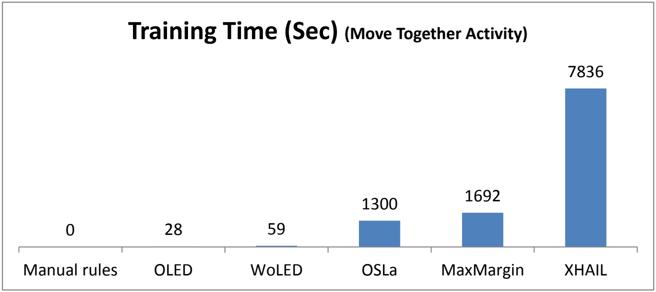
Background Knowledge
holdsAt(F, $T+1$) $\leftarrow$
initiatedAt(F, T).
holdsAt(F, $T+1$) $\leftarrow$
holdsAt(F, T),
not terminatedAt(F, T).
Mode Declarations
head(initiatedAt($move$(+id, +id), +time))
head(terminatedAt($move$(+id, +id), +time))
body(happensAt($walking$(+id, +id), +time))
body(not happensAt($walking$(+id, +id), +time))
body($distLessThan$(+id, +id, #dist, +time))
body($dirLessThan$(+id, +id, #dist, +time))
Current MLN Theory H_t:
1.345 initiatedAt($move(X, Y)$, T) $\leftarrow$
happensAt($walking(X)$, T),
happensAt($walking(Y)$, T),
$distLessThan(X, Y, 34, T)$
0.865 terminatedAt($move(X, Y)$, T) $\leftarrow$
happensAt($inactive(X)$, T),
not $distLessThan(X, Y, 34, T)$
WoLED
MAP Inference
Theory Expansion
Weights Update
Hoeffding Tests/Rule Expansion
Pruning
Training Stream
. . .
Training Interpretation I_t
holdsAt($move(id_1, id_2)$, 10)
happensAt($walking(id_1)$, 9)
happensAt($walking(id_2)$, 9)
$coords(id_1, 23, 104, 9)$
$coords(id_2, 42, 84, 9)$
$direction(id_1, 212, 9)$
$direction(id_2, 78, 9)$
. . .
Training Interpretation $I_{t'}$
not holdsAt($move(id_1, id_2)$, 100)
happensAt($walking(id_1)$, 99)
happensAt($walking(id_2)$, 99)
$coords(id_1, 205, 23, 99)$
$coords(id_2, 462, 24, 99)$
$direction(id_1, 23, 99)$
$direction(id_2, 798, 99)$
. . .

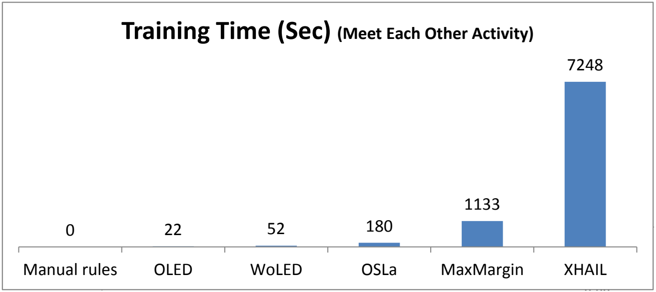
OLED-MLN Evaluation on the CAVIAR Dataset





OLED-MLN Evaluation on the CAVIAR Dataset





Some Useful Reading Readings

- *Logic, Reasoning & Logic Programming:*
  - Sergot M. Knowledge Representation. Course 491, Department of Computing, Imperial College London
  - https://www.doc.ic.ac.uk/~mjs/teaching/491.html
  - Mueller, Erik T.: Event calculus and temporal action logics compared. Artif. Intell. 170(11): 1017-1029 (2006).
  - Mueller, Erik T. Commonsense reasoning: an event calculus based approach. Morgan Kaufmann, 2014.
- *Relational Learning/Inductive Logic Programming:*
  - De Raedt, Luc. Logical and relational learning. Springer Science & Business Media, 2008.
- *Statistical Relational Learning:*
  - Koller D, Friedman N, Dzeroski S, Sutton C, McCallum A, Pfeffer A, Abbeel P, Wong MF, Heckerman D, Meek C, Neville J. Introduction to Statistical Relational Learning. MIT press; 2007.
- *Rule learning:*
  - Frnkranz, J., Gamberger, D., & Lavra, N. (2012). Foundations of rule learning. Springer Science & Business Media.

References I

- Event Calculus for Normative Specifications:
  - A. Artikis, M. Sergot, and J. Pitt, An Executable Specification of a Formal Argumentation Protocol, Artificial Intelligence Journal, 171(10-15):776-804, 2007.
  - A. Artikis, M. Sergot, and J. Pitt, Specifying Norm-Governed Computational Societies, ACM Transactions on Computational Logic, to appear in 2008.
- Event Calculus for Complex Event Recognition (RTEC paper):
  - Artikis A., Sergot M. and Paliouras G. An Event Calculus for Event Recognition. IEEE Transactions on Knowledge and Data Engineering (TKDE), 27(4):895-908, 2015.
- Event Calculus in ProbLog:
  - Skarlatidis A., Artikis A., Filippou J. and Paliouras G. A Probabilistic Logic Programming Event Calculus, Journal of Theory and Practice of Logic Programming (TPLP), 15(2):213-245, 2015.
- Event Calculus in Markov Logic Networks:
  - Skarlatidis A., Paliouras G., Artikis A., and Vouros G. Probabilistic Event Calculus for Event Recognition, ACM Transactions on Computational Logic, 16(2):1-37, 2015.

References II

- XHAIL paper:
  - Ray O. Nonmonotonic abductive inductive learning. Journal of Applied Logic. 2009 Sep 1;7(3):329-40.
- Learning with the Event Calculus papers:
  - Katzouris, N., Artikis, A. and Paliouras, G., 2015. Incremental learning of event definitions with inductive logic programming. Machine Learning, 100(2-3), pp.555-585.
  - Katzouris, N., Artikis, A. and Paliouras, G., 2016. Online learning of event definitions. Theory and Practice of Logic Programming, 16(5-6), pp.817-833.
  - Katzouris, N., Michelioudakis, E., Artikis, A. and Paliouras, G., Online learning of weighted relational rules for complex event recognition. In ECML-PKDD (pp. 396-413). Springer, Cham, 2018.