Agenda

- Technical Preliminaries
- The PyReason framework
- Planned integration with ARL Battlespace
Technical Preliminaries

Propositional Logic

Semantics

Implication/Rules

Fixpoint Operator

First Order Logic
Propositional Logic

Jack has school and school starts at 7 am. So Jack wakes up early.
Propositional Logic

Jack has school and school starts at 7 am. So Jack wakes up early.

A \land B \rightarrow C
Propositional Logic

Jack has school and school starts at 7 am. So Jack wakes up early.

\[ A \land B \rightarrow C \]

- **Atoms:** A, B, C, \ldots (either True or False)
- **Operators:** ¬, ∨, ∧, →, ↔
- **Formulas:**
  - A
  - ¬A
  - A ∨ B
  - (¬A) ∧ B → C
Semantics

- Consider a set of atoms
  \[ U = \{a_1, a_2, a_3\} \]

- Then we can define a world \( W \) as a subset of \( U \)
  \[ W = \{\}, \{a_1\}, \{a_2\}, \{a_3\}, \{a_1, a_2\}, \{a_1, a_3\}, \{a_2, a_3\}, \{a_1, a_2, a_3\} \]

**Intuition:** if an atom is a member of a world, it is considered true in that world otherwise it is false.
Implication / Rules

● Consider formulas: \( f, f', f'' \)

● Example of a rule:

\[
\begin{align*}
f' \lor f'' & \rightarrow f \\
\text{body} & \rightarrow \text{head(atoms/negations)}
\end{align*}
\]

● Alternatively, we can write this as:

\[
f \leftarrow f' \lor f''
\]

● A fact is a rule with no body (i.e. body is always true)

\[
f \leftarrow
\]
The Fixpoint Operator ($\Gamma$):

- An application of $\Gamma$ involves:
  - **Input** A set of atoms $U$, a world $w$, a set of rules $R$.
  - **Apply** all rules in $R$ satisfied by $w$.
  - **Output** $w$ + any atoms concluded from applied rules.
The Fixpoint Operator ($\Gamma$):

- An application of $\Gamma$ involves:
  - **Input**: A set of atoms $U$, a world $w$, a set of rules $R$.
  - **Apply**: all rules in $R$ satisfied by $w$.
  - **Output**: $w$ + any atoms concluded from applied rules.

- Consider,

  - $U = \{a_1, a_2, a_3\}$
  - $R = \{a_1 \leftarrow, a_2 \leftarrow a_1, a_3 \leftarrow a_2\}$

  - $w_1 = \{a_2\}$
  - $\Gamma_R(w_1) = \{a_1, a_2, a_3\}$
The Fixpoint Operator ($\Gamma$):

- Consider,
  - $U = \{a_1, a_2, a_3\}$
  - $R = \{a_1 \leftarrow, a_2 \leftarrow a_1, a_3 \leftarrow a_2\}$
  - $w_2 = \{a_3\}$
  - $\Gamma_R(w_2) = \{a_1, a_3\}$
  - $w_3 = \{\} $
  - $\Gamma_R(w_3) = \{a_1\}$
The Fixpoint Operator ($\Gamma$):

- An application of $\Gamma$ involves:
  - **Input** A set of atoms $\mathcal{U}$, A world $w$, A set of rules $R$.
  - **Apply** all rules in $R$ satisfied by $w$.
  - **Output** $w$ + any atoms concluded from applied rules.

- Can be written as:
  \[
  \Gamma_R(w) = w \cup \bigcup_{r \in R} \{ \text{head}(r) \text{ such that } \text{body}(r) \subseteq w \}
  \]

- $\Gamma$ can be iteratively applied multiple times as:
  \[
  \Gamma_R^{(i)}(w) = \Gamma_R \left( \Gamma_R^{(i-1)}(w) \right)
  \]
The Fixpoint Operator ($\Gamma$):

- Useful for making conclusions, as well as, explanations behind them:
  - $U = \{a_1, a_2, a_3\}$
  - $R = \{a_1 \leftarrow, a_2 \leftarrow a_1, a_3 \leftarrow a_2\}$

- $w_3 = \{\}$
  - $\Gamma_R^{(1)}(w_3) = \{a_1\}$
  - $\Gamma_R^{(2)}(w_3) = \{a_1, a_2\}$
  - $\Gamma_R^{(3)}(w_3) = \{a_1, a_2, a_3\}$
First Order Logic/Predicate Calculus

Predicates are a way to specify atomic propositions.
Consider,
“friend” is a predicate
$v_1, v_2$ are two variables

$\text{friend}(v_1, v_2)$
First Order Logic/Predicate Calculus

Predicates are a way to specify atomic propositions.
Consider,
“friend” is a predicate
$v_1, v_2$ are two variables
jack, phil are two people (constants)

friend(v₁, v₂)
friend(jack, phil)
First Order Logic/Predicate Calculus

Predicates are a way to specify atomic propositions. Consider, “friend” is a predicate. $v_1, v_2$ are two variables.

`friend(v_1, v_2)`

Jack and Phil are two people (constants)

`friend(jack, phil)`

Jack and Phil are friends
First Order Logic/Predicate Calculus

Predicates are a way to specify atomic propositions.
Consider,
“friend” is a predicate
$v_1, v_2$ are two variables

jack, phil are two people (constants)

friend(v₁, v₂)
friend(jack, phil)

Jack and Phil are friends
First Order Logic/Predicate Calculus

Non-ground atoms are the key item that differentiates Predicate Calculus from Propositional Calculus.

Predicate + Variable symbol(s) = (Non-ground) atomic proposition

Predicate + Constant(s) = (Ground) atomic proposition

\text{friend}(v_1, v_2)

\text{friend}(jack, phil)

Grounding
Predicate Calculus in Knowledge Graphs

- Unary predicates can model attributes of nodes.
  e.g. `student(c_1)`

- Binary predicates can model relationships between nodes (attributes of edges).
  e.g. `friend(c_1, c_2)`
The PyReason framework

Lattice structure and annotations

Support for Temporal Reasoning

Notion of Interpretation

Rules

Type-checking and Consistency checking
Generalized Annotated Logic

PyReason performs reasoning about first-order and propositional logic statements,
PyReason performs reasoning about first-order and propositional logic statements, that can be annotated with elements of a lattice structure.
PyReason performs reasoning about first-order and propositional logic statements, that can be annotated with elements of a lattice structure.

Interpretations map atomic propositions to elements of this lattice structure.
Design Feature: Uncertainty

Allowing Interpretation (I) to map atoms to bounds allows us to model uncertainty effectively.

For e.g.

When we have no information about friend(jack, phil),

\[ I(\text{friend(jack, phil)}) = [0, 1] \]

While still supporting the propositional cases:

\[ I(\text{friend(jack, phil)}) = [1, 1] \quad \text{friends} \]

\[ I(\text{friend(jack, phil)}) = [0, 0] \quad \text{not friends} \]
Design Feature: Temporal Reasoning

We additionally allow Interpretation (I) to map time (alongside atoms) to bounds and hence we can perform reasoning over time.

Continuing with the same example, we can have,

\[ I(\text{friend(jack,phil), jan}) = [0,0] \]  
not friends in January

\[ I(\text{friend(jack,phil), feb}) = [0,1] \]  
no info about February

\[ I(\text{friend(jack,phil), mar}) = [1,1] \]  
friends in March
PyReason Input/Outputs

Knowledge Graph
Initial Interpretations
Facts
Rules OR Observations
Learning Module

PyReason

Final Interpretations (Option: pickled)
Explanations (Rule, Time) for every change in interpretation

Option: Interpretations at every step
Design Feature: Explainability

By implementing the fixpoint operator directly (as opposed to a black box heuristic) the software enables full explainability of the result.

- We can recover a trace of every rule applied and its effect.
- We can uncover causal relationships between atomic propositions.
- We can detect logical flaws and inconsistencies.
Logical Rules

Universally quantified non-ground rule

$$\forall X: \text{pred}(X): f(x_1, \ldots, x_n) \leftarrow_{\Delta t} \bigwedge_{\text{pred}_i \in \text{UnaSet}} \text{pred}_i(X): x_i$$

Universal quantifier

*(design feature)*

Annotation is a function over the elements in lattice

*e.g. Max, Min, Avg,*

*Fuzzy t-norms and conorms*

*(design feature)*
Universally quantified non-ground rule

\[ \forall X: \text{pred}(X): f(x_1, \ldots, x_n) \leftarrow_{\Delta t} \exists_k X': \text{rel}(X, X'): [1,1] \land \bigwedge_{\text{pred}_q \in \text{BinSet}} \text{pred}_q(X, X'): x_q \]

\[ \land \bigwedge_{\text{pred}_r \in \text{UnaSet}} \text{pred}_r(X): x_r \land \bigwedge_{\text{pred}_s \in \text{UnaSet'}} \text{pred}_s(X'): x_s \]

Existential quantifier

*design feature*

`rel` is a reserved word. Marked portion denotes that - an edge exists between X and X'.
Logical Rules: Examples

- \textit{promoted}(X): [1,1] \leftarrow \text{\textit{1year}} \ gpa(X): [0.3, 1]
  Student X will get promoted at the end of the year if their overall gpa is in the top 70% of the class.

- \textit{promoted}(X): [1,1] \leftarrow \text{\textit{1year}} \ gpa(X): [0.3, 1] \land \forall Y \textit{takes}(X, Y): [1,1] \land \forall Y \textit{passed}(X, Y): [1,1]
  adds an additional condition that to get promoted, X must pass all of the courses they take.

- \textit{gpa}(X): [\textit{avg}(x_s)] \leftarrow \text{\textit{1year}} \ \exists 2 \textit{takes}(X, Y): [1,1] \land \textit{score}(X, Y): x_s
  shows a way to compute gpa using two classes taken by X.
Design Feature: Type-checking

- student(eve) ✔
- student(cal) ✔
- subject(math) ✔
- friend(eve, cal) ✔
- takes(eve, math) ✔
- student(math) ✗
- subject(eve) ✗
- subject(cal) ✗
- friend(math, cal) ✗
- takes(eve, cal) ✗
- takes(math, eve) ✗
Design Feature: Type-checking

- Avoids silly errors like: "Math is driving a car".
- Provides drastic reduction to complexity induced by the grounding problem, by increasing graph sparsity, reducing storage and computations.
- Significantly improves utility in a variety of application domains.
Design Feature: Type-checking

- Optional feature. Can be turned on/off.

- Specified at the time of building the graph: if we have prior knowledge about constraints over predicate-atom pairs.
Design Feature: Support for literals

A literal is any ground atom or a negation of a ground atom.

Traditional logic frameworks only support atoms or its negation, not both.

In PyReason -

- Atoms and their negations can be simultaneously implemented.
- We define both as separate ground atoms, and,
- We define a consistency constraint that prevents an atom and its negation to co-exist.
Design Feature: Consistency for pairs

Defining literals:

\[ I(\text{at\_home}(x),t) = [1,1] \quad \text{and} \quad I(\text{not\_home}(x),t) = [1,1] \]

✘

Modelling relationships between pairs which might become inconsistent:

\[ I(\text{bachelor}(x),t) = [1,1] \quad \text{and} \quad I(\text{married}(x),t) = [1,1] \]

✘

\[ I(\text{fit}(x),t) = [0.6,1] \quad \text{and} \quad I(\text{injured}(x),t) = [0,0.8] \]

✘
Design Feature: Consistency for pairs

Checking:
\[ I_1 = [L_1, U_1] \]
\[ I_2 = [L_2, U_2] \]
Then they are consistent,
Iff \( L_1 \leq 1 - L_2 \) and \( U_1 \geq 1 - U_2 \)

Resolution:
\[ I_1 = I_2 = [0, 1] \]
Design Feature: Consistency \textit{during execution}

Checking:

I_{current} = [L_1, U_1] \\
I_{new} = [L_2, U_2] \\
Then they are consistent, \\
Iff \ L_1 \leq U_2 \quad \text{and} \quad U_1 \geq L_2 \quad \text{and} \quad L_2 \leq U_2 \\
\textit{i.e. from same lower lattice}

Resolution:

I_{updated} = [0, 1]
Integration within ARL Battlespace

Modelling the Game World

Interfacing with PyReason
Modeling the Game World

- The game board is modeled as a graph:
  - Each **square** is a node.
  - Edges between **neighbouring squares**, air and land.
Modeling the Game World

- The game board is modeled as a graph:
  - Each **square** is a node.
  - Edges between **neighbouring squares**, air and land.

- Other components can be modeled as -
  - **Units** (Soldier, Tank, Truck, Flag, Airplane, Missiles) are nodes, with attribute ‘**type**’ = {air, ground, immovable}.
  - **Players** (A,B,C,D) are nodes.
Modeling the Game World

- The game board is modeled as a graph:
  - Each **square** is a node.
  - Edges between **neighbouring squares**, air and land.

- Other components can be modeled as:
  - **Units** (Soldier, Tank, Truck, Flag, Airplane, Missiles) are nodes, with attribute ‘**type**’ = {air, ground, immovable}.
  - **Players** (A,B,C,D) are nodes.
  - Edges between Units and Squares have **at(U,S)**.
  - Edges between Players and Units have **of(P,U)**.

...
Modeling the Game World

- Rules:
  - All possible actions

  e.g. Advancement changes location of an unit
  e.g. Rotate changes orientation.
Modeling the Game World

- Rules:
  - All possible actions
  - Movement of missiles are facts

  e.g. Advancement changes location of an unit
Modeling the Game World

- Rules:
  - All possible actions
  - Movement of missiles are facts
  - Causal effects of actions

  e.g. Advancement changes location of an unit

  e.g. Overlap leads to mutual destruction
  - Advance U1, S1
  - Advance U2, S1
  - Trigger: Destroy U1, U2
Modeling the Game World

- Rules:
  - All possible actions e.g. Advancement changes location of an unit
  - Movement of missiles are facts e.g. Overlap leads to mutual destruction
  - Causal effects of actions

- Termination conditions (flag capture / annihilation) is captured in the body of a rule, which when fired ends the game.
Modeling the Game World

- **Rules:**
  - All possible actions, e.g. Advancement changes location of an unit
  - Movement of missiles are facts, e.g. Overlap leads to mutual destruction
  - Causal effects of actions
  ...  

- **Termination** conditions (flag capture / annihilation) is captured in the body of a rule, which when fired ends the game.

- At initialization, type-checking ensures attributes are matched to appropriate nodes and edges in the graph.
Interfacing with PyReason

Input: Current State
List of interpretations with bounds in .yaml format

Course of Action
List of tuples of the form (action, player, unit, time)

Output: Next States
List of interpretations with bounds in .pkl format