PyReason

Arizona State University

School of Computing and Augmented Intelligence

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 $\wedge$ B $\rightarrow$ C

Propositional Logic

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

A ∧ B → C

PyReason

- Atoms: A, B, C, . . . (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:

$$f' \lor f'' \rightarrow f$$

body $\rightarrow$ *head(atoms/negations)*

- 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 (Γ):

- An application of Γ 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 (Γ):

- An application of Γ 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,
  - $U = \{a_1, a_2, a_3\}$
  - $R = \{a_1 \leftarrow, a_2 \leftarrow a_1, a_3 \leftarrow a_2\}$
  - $w_1 = \{a_2\}$
  - $\Gamma_R(w_1) = \{a_1, a_2, a_3\}$

V2 LAB

The Fixpoint Operator (Γ):

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

  - $w_2 = \{a_3\}$
  - $\Gamma_R(w_2) = \{a_1, a_3\}$

  - $w_3 = \{\}$
  - $\Gamma_R(w_3) = \{a_1\}$

LAB V2

The Fixpoint Operator (Γ):

- An application of Γ 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.

- Can be written as:

$$\Gamma_R(w) = w \cup \mathbf{U}_{r \in R}\{head(r)\ such\ that\ body(r) \subseteq w\}$$

- Γ can be iteratively applied multiple times as:

$$\Gamma_R^{(i)}(w) = \Gamma_R\left(\Gamma_R^{(i-1)}(w)\right)$$

The Fixpoint Operator (Γ):

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

  - $w_3 = \{ \}$
  - $\Gamma_R^{(1)}(w_3) = \{a_1\}$ $a_1 \leftarrow$
  - $\Gamma_R^{(2)}(w_3) = \{a_1 , a_2\}$ $a_2 \leftarrow a_1$
  - $\Gamma_R^{(3)}(w_3) = \{a_1 , a_2, a_3\}$ $a_3 \leftarrow a_2$

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

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

friend(v_1 , v_2)

jack, phil are two people (constants)

friend(jack, phil)

LAB V2

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, 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

friend(v_1 , v_2)

jack, phil are two people (constants)

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

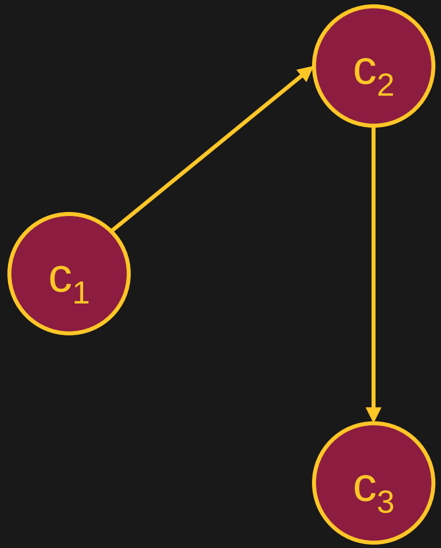
Predicate + Constant(s) = (Ground) atomic proposition

$$\text{friend}(v_1, v_2) \xrightarrow{\text{Grounding}} \text{friend(jack, phil)}$$

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

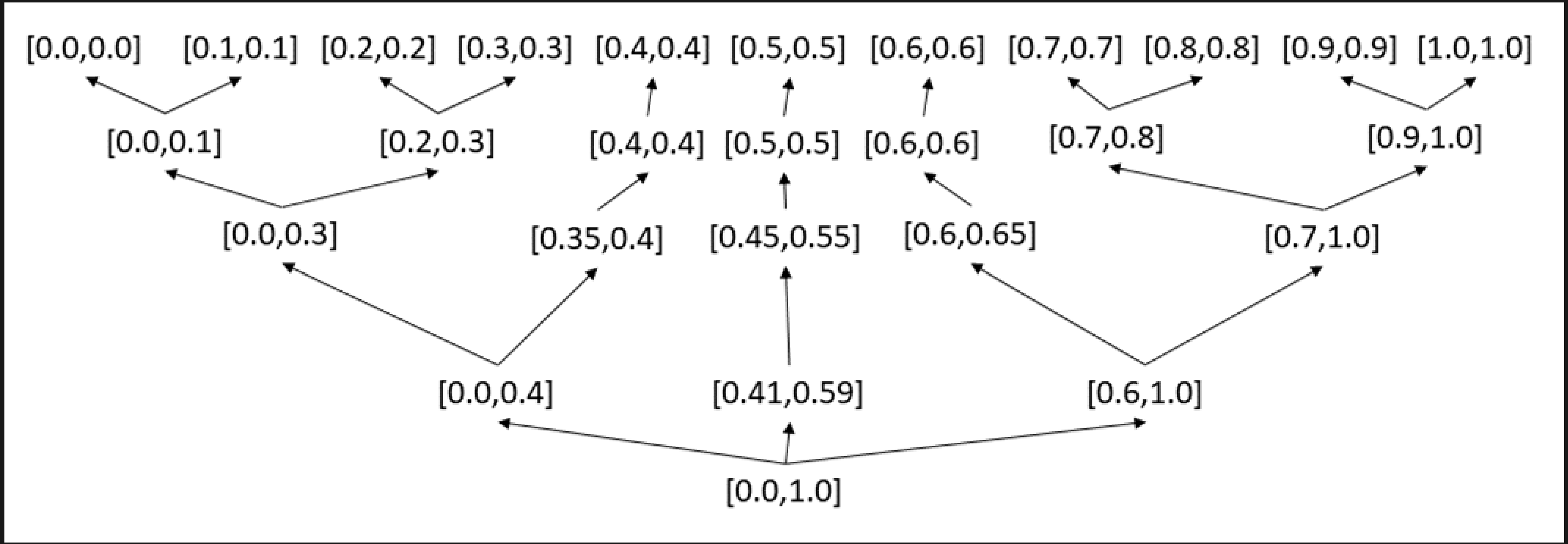
Type-checking and Consistency checking

Generalized Annotated Logic

PyReason performs reasoning about first-order and propositional logic statements,

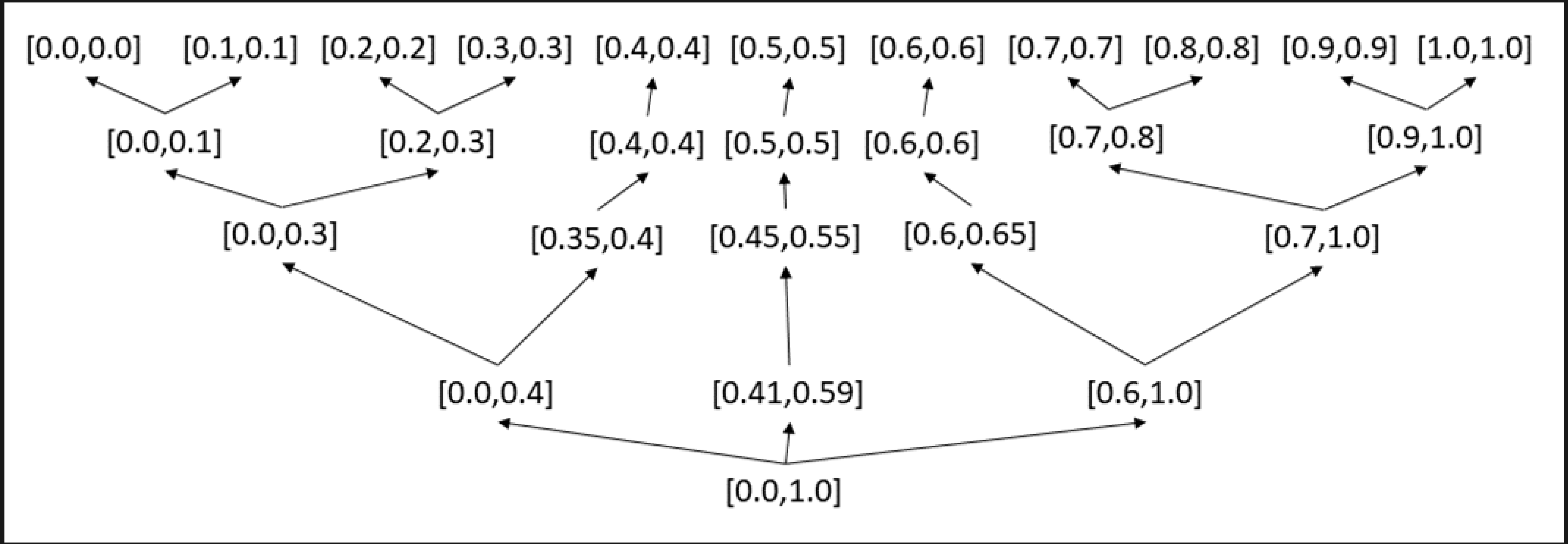
Generalized Annotated Logic

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



Generalized Annotated Logic

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(friend(jack,phil)) = [0,1]

While still supporting the propositional cases:

I(friend(jack,phil)) = [1,1] *friends*

I(friend(jack,phil)) = [0,0] *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,

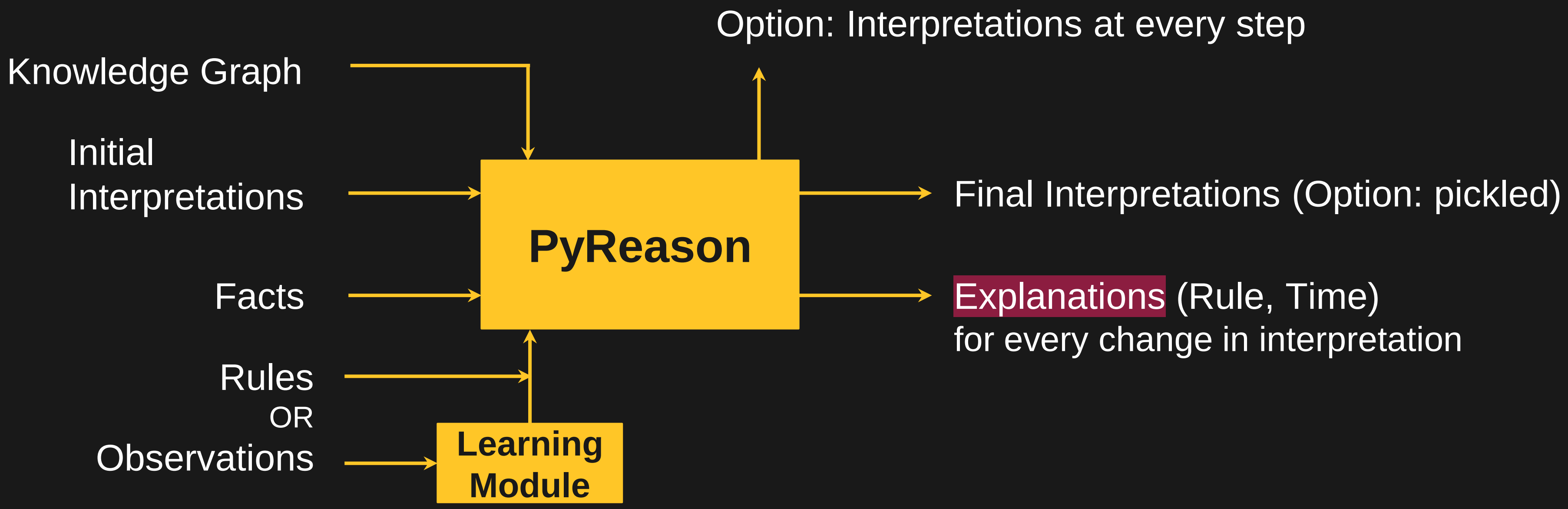
I(friend(jack,phil), jan) = [0,0] *not friends in January*

I(friend(jack,phil), feb) = [0,1] *no info about February*

open-world assumption

I(friend(jack,phil), mar) = [1,1] *friends in March*

PyReason Input/Outputs



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.

V2 LAB

Logical Rules Reasoning within a node

Universally quantified non-ground rule

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

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)

V2 LAB

Logical Rules Reasoning across an edge

Universally quantified non-ground rule

$$\forall X: pred(X): f(x_1, \ldots, x_n) \leftarrow_{\Delta t} \boxed{\exists_k} X': \boxed{rel(X, X'): [1,1]} \wedge \bigwedge_{pred_q \in BinSet} pred_q(X, X'): x_q$$

$$\wedge \bigwedge_{pred_r \in UnaSet} pred_r(X): x_r \wedge \bigwedge_{pred_s \in UnaSet'} pred_s(X'): x_s$$

Existential quantifier

(design feature)

'rel' is a reserved word.

Marked portion denotes that -

an edge exists between X and X'

Logical Rules Examples

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

- $promoted(X): [1,1] \leftarrow_{1year} gpa(X): [0.3, 1] \wedge \bigwedge_{Y} takes(X, Y): [1,1] \wedge \bigwedge_{Y} passed(X, Y): [1,1]$
  adds an additional condition that to get promoted, X must pass all of the courses they take.

- $gpa(X): [avg(x_s)] \leftarrow_{1year} \; \exists_2 \, takes(X, Y): [1,1] \wedge score(X, Y): x_s$
  shows a way to compute gpa using two classes taken by X.

V2 LAB

Design Feature: Type-checking

student(eve) ✓

student(cal) ✓

student(math) ✗

subject(math) ✓

subject(eve) ✗

subject(cal) ✗

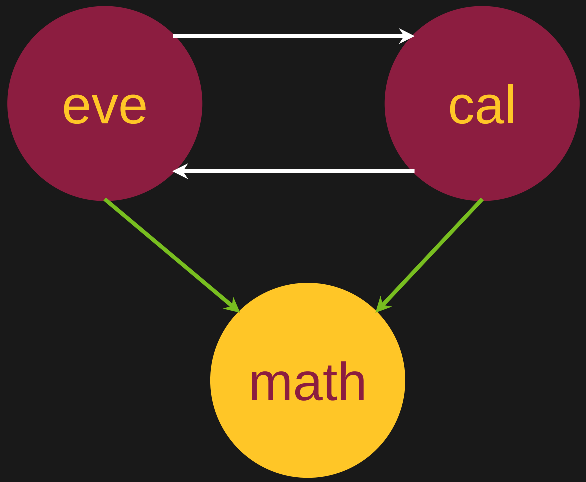
friend(eve, cal) ✓

friend(math, cal) ✗

takes(eve, math) ✓

takes(eve, cal) ✗

takes(math, eve) ✗

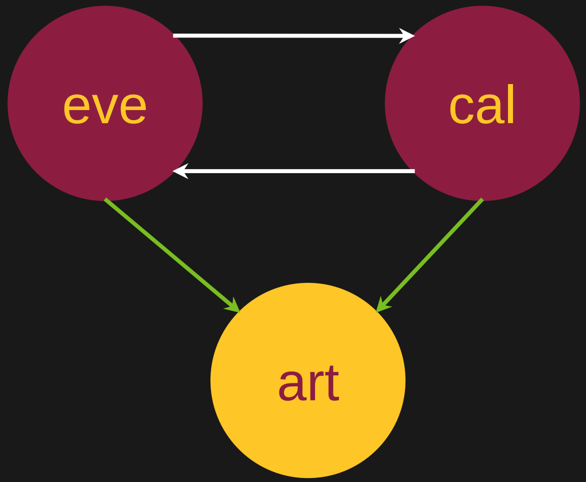


Design Feature: Type-checking

- Avoids silly errors like: *“Math is driving a car”.*

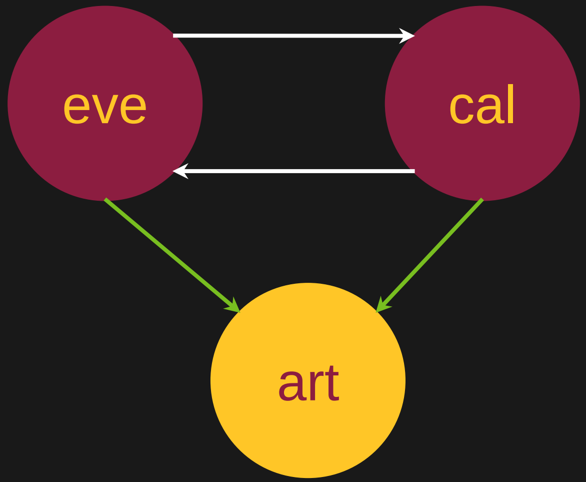
- Provides drastic reduction to complexity induced by the grounding problem, by increasing graph sparsity, reducing storage and computations.

- 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.

V2 LAB

Design Feature: Consistency for pairs

Defining literals:

I(at_home(x),t) = [1,1] and I(not_home(x),t) = [1,1] ✗

Modelling relationships between pairs which might become inconsistent:

I(bachelor(x),t) = [1,1] and I(married(x),t) = [1,1] ✗

I(fit(x),t) = [0.6,1] and I(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 during execution

Checking:

$I_{current} = [L_1, U_1]$

$I_{new} = [L_2, U_2]$ *from outcome of a rule*

Then they are consistent,

Iff $L_1 \le U_2$ and $U_1 \ge L_2$ and $L_2 \le U_2$ *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 *e.g. Advancement changes location of an unit*
  - Movement of missiles are facts

Modeling the Game World

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

  . . .

- 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
  - Causal effects of actions *e.g. Overlap leads to mutual destruction*

    . . .

- 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

Thank You

V2 LAB