Automated Generation of Control Concepts Annotation Rules Using Inductive Logic Programming

April 24, 2022

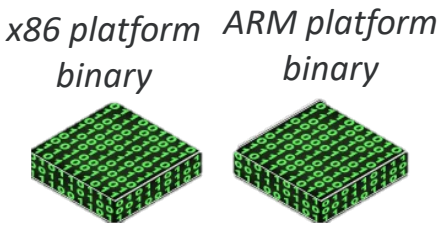
Basel Shbita
University of Southern California / GE Research (intern)

Abha Moitra
GE Research

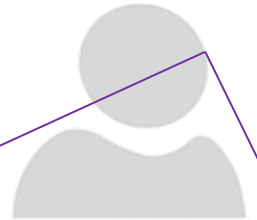
As part of the
Graphical Symbol Expression Network (GraSEN) project
DARPA's Recovery of Symbolic Mathematics from Code (ReMath) program

Intro

- Cyber-Physical Systems (CPS)
  - Physical systems in which mechanism is controlled by computer-based algorithms
  - Include micro-controller software
  - Control theory connects applied math to CPS engineering
    - Primitive and higher-level (composite) mathematical concepts
      - Constant, gain, saturation
      - Output signal, error signal
      - PID controller

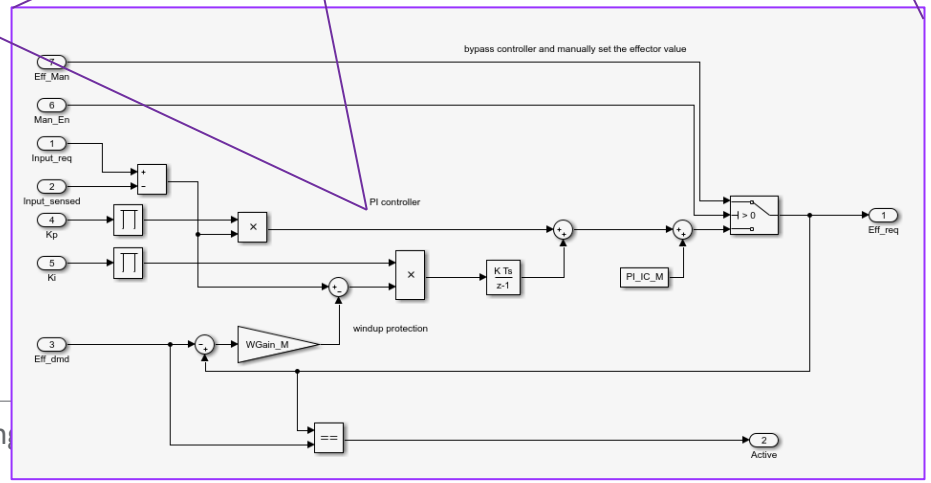


Engine



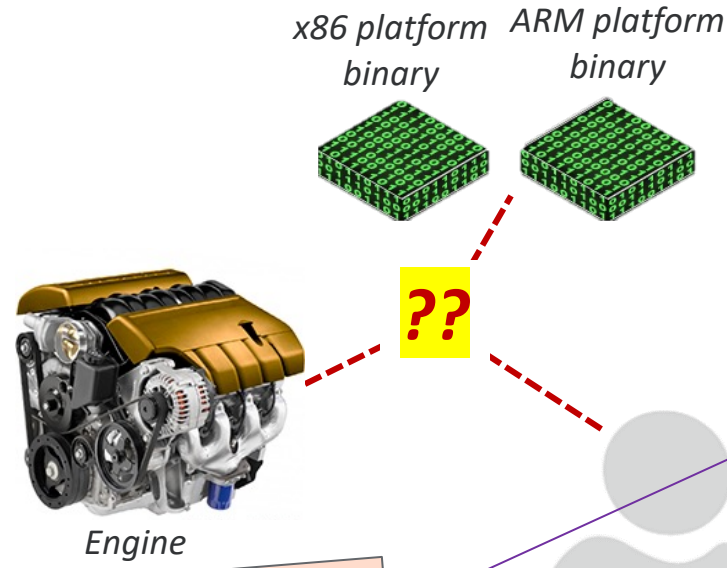
Expert

$$u(t) = K_p e(t) + K_i \int_{\tau=0}^{t} e(\tau)d\tau$$



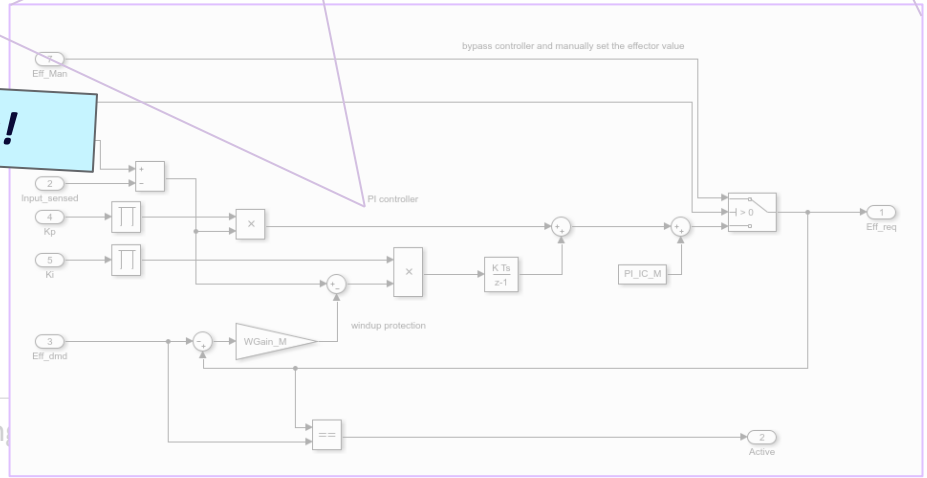
Intro

- Cyber-Physical Systems (CPS)
  - Physical systems in which mechanism is controlled by computer-based algorithms
  - Include micro-controller software
  - Control theory connects applied math to CPS engineering
    - Primitive and higher-level (composite) mathematical concepts
      - Constant, gain, saturation
      - Output signal, error signal
      - PID controller



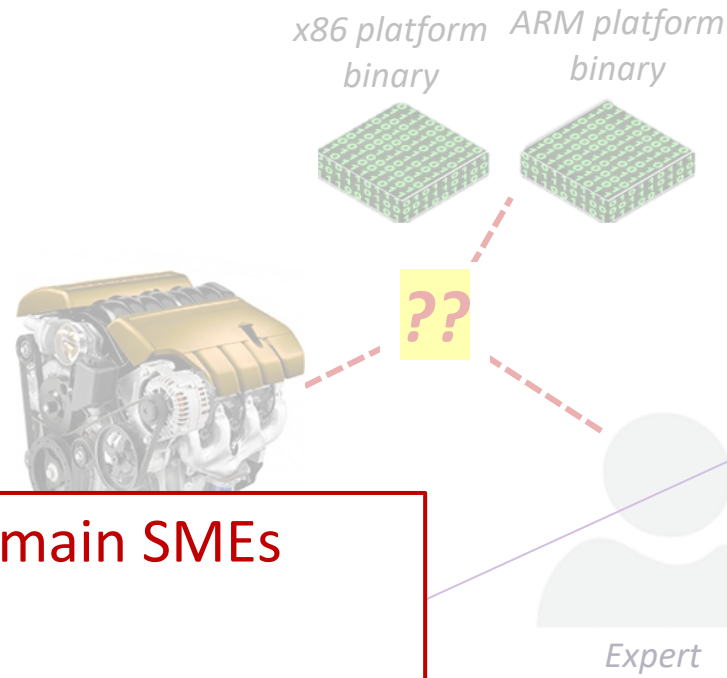
- *How can we can we **reverse engineer** a legacy product?*
- *How can we **route relevant parts** of the software to experts?*
- *How can we **recover mathematical structures** implemented in software?*

Requires fully manual analysis by highly specialized experts!

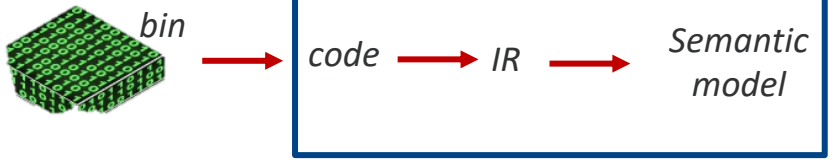


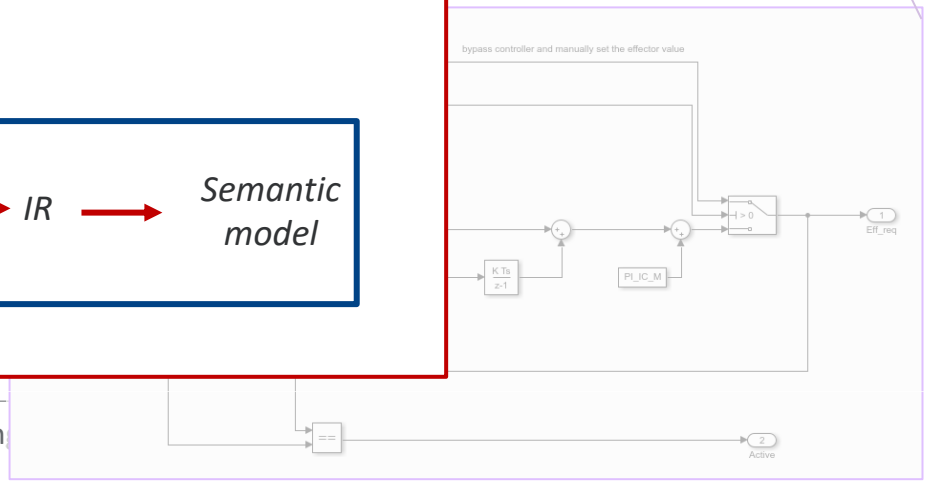
Intro

- Cyber-Physical Systems (CPS)
  - Physical systems in which mechanism is controlled by computer-based algorithms
  - Include micro-controller software
  - Control theory connects applied math to CPS engineering
    - Primitive and higher-level (composite) mathematical concepts



- Bridging the gap between binary code analysts & domain SMEs
- Binary → code analysis framework → Expert
- How?
  - binary decompilation
  - semantic code pattern analysis





Goal

```c
double PI_calc(double Input_dmd, double Input_sensed, double Kp_M, double Ki_M, double timestep)
{
    double error = Input_dmd - Input_sensed;

    integrator_state = integrator_state + timestep*error;

    return error*Kp_M + integrator_state*Ki_M;
}
```

Decompiled C file

Semantic Model

GrFNExtractionModel.sadl

```
uri "http://sadl.org/GrFNExtractionModel.sadl" alias grfnem.

Node is a class
    described by uid with a single value of type string
    described by description with a single value of type string
    described by identifier with a single value of type string
    described by metadata with a single value of type Metadata
.

GrFN is a type of Node
    described by date_created with a single value of type dateT
    described by functions with values of type Function
    described by hyper_edges with values of type HyperEdge
    described by subgraphs with values of type SubGraph
```

Pre-defined semantic model

SADL

GE_simple_PI_controller_combined.sadl

```
uri "http://aske.ge.com/GE_simple_PI_controller_combined" ali
    (note "This ontology was created by extraction from GrFN
import "http://sadl.org/GrFNExtractionModel.sadl" as grfnem.

// Individuals:
_00d5149c-d6fd-b1ec-56d2-84b41e9d26b9 is a grfnem:Variable,
    has grfnem:metadata (a grfnem:Metadata
    with grfnem:line_begin 48,
    with grfnem:from_source true),
```

SADL

GE_simple_PI_controller_combined.owl

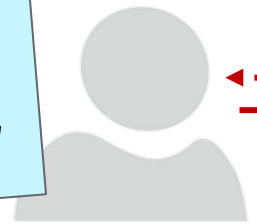
```
</grfnem:Type>
<grfnem:GrFN rdf:ID="c352a5d2-329c-d119-792f-1a08d6325148">
  <grfnem:subgraphs>
    <grfnem:SubGraph rdf:ID="c63bcc0d-891b-9335-4a4b-9030954c5158">
      <grfnem:nodes>
        <grfnem:ExpressionTree rdf:ID="ff9e652f-372f-e2ba-e68e-91669f27
          <grfnem:nodes>
            <grfnem:ExpNode rdf:ID="_81ab2dcb-3710-0dfe-842c-8a5ee8bccb
              <grfnem:exp_node_type>DEFINITION</grfnem:exp_node_type>
            </grfnem:ExpNode>
          </grfnem:nodes>
          <grfnem:nodes>
            <grfnem:ExpNode rdf:ID="_1fa58a74-8ceb-0435-7f38-388b1f924e
              <grfnem:node_value>0.0</grfnem:node_value>
              <grfnem:exp_node_type>VALUE</grfnem:exp_node_type>
            </grfnem:ExpNode>
```

OWL

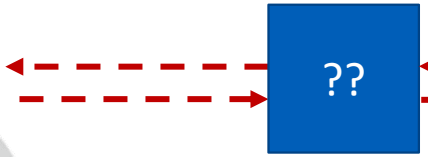
Generate Annotation Rules!

How?

UI, Data transformation, & Inductive Logic Programming!



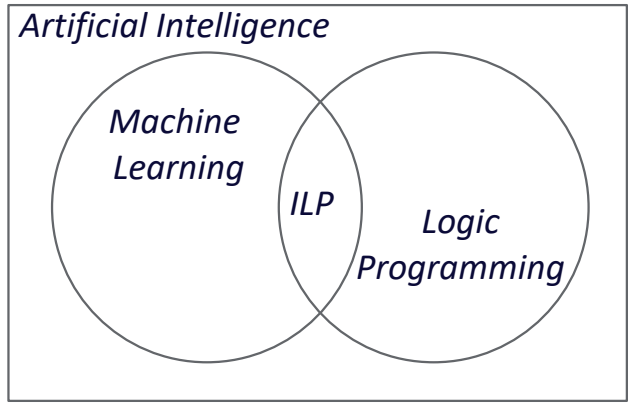
Expert



How can we leverage the semantic model for domain-knowledge acquisition & discovery to assist the SME?

Inductive Logic Programming (ILP)

- ILP
  - Advantages:
    - Requires small amounts of data
    - Uses expressive FOL (First-Order Logic)
    - Takes background knowledge into account
  - Used for classification & prediction
    - Bioinformatics, NLP
  - Given background knowledge (**B**) & specific observations (**E+**, **E-**) induce general rules (hypothesis, **H**)
  - *Aleph*, an ILP system
    - **A** **L**earning **E**ngine for **P**roposing **H**ypotheses
    - Learning from entailments (LFE)



$$B = \left\{ \begin{array}{l} \texttt{builder(alice).} \\ \texttt{builder(bob).} \\ \texttt{enjoys\_lego(alice).} \\ \texttt{enjoys\_lego(claire).} \end{array} \right\}$$

$$E^{+} = \{ \texttt{happy(alice).} \} \quad E^{-} = \left\{ \begin{array}{l} \texttt{happy(bob).} \\ \texttt{happy(claire).} \end{array} \right\}$$

$$H = \{ \forall \texttt{A. builder(A)} \wedge \texttt{enjoys\_lego(A)} \rightarrow \texttt{happy(A)} \}$$

Problem Definition

- Generate classification (annotation) rules for math & control concepts using ILP
  - Input
    - (OWL) data representing basic code elements in a CPS program (e.g., variables, functions, operators)
    - Positive & negative examples the objective concept (identified by the SME)
  - Output
    - (Prolog) program describing the objective concept (e.g., constants, error signals, PI controller)
  - Iterative process – if the generated rule identifies false positive → add example & repeat

Decompiled C file

```
8   double PI_calc(double Input_dmd, double Input_sensed, double Kp_M, double Ki_M, double timestep)
9   {
10      double error = Input_dmd - Input_sensed;
11
12      integrator_state = integrator_state + timestep*error;
13
14      return error*Kp_M + integrator_state*Ki_M;
15  }
```

OWL

Semantic Model

GE_simple_PI_controller_combined.owl

```
38  </grfnem:Type>
39  <grfnem:GrFN rdf:ID="c352a5d2-329c-d119-792f-1a08d6325148">
40    <grfnem:subgraphs>
41      <grfnem:SubGraph rdf:ID="c63bcc0d-891b-9335-4a4b-9030954c5158">
42        <grfnem:nodes>
43          <grfnem:ExpressionTree rdf:ID="ff9e652f-372f-e2ba-e68e-91669f27
44            <grfnem:nodes>
45              <grfnem:ExpNode rdf:ID="_81ab2dcb-3710-0dfe-842c-8a5ee8bccb
46                <grfnem:exp_node_type>DEFINITION</grfnem:exp_node_type>
47              </grfnem:ExpNode>
48            </grfnem:nodes>
49            <grfnem:nodes>
50              <grfnem:ExpNode rdf:ID="_1fa58a74-8ceb-0435-7f38-388b1f924e
51                <grfnem:node_value>0.0</grfnem:node_value>
52                <grfnem:exp_node_type>VALUE</grfnem:exp_node_type>
53              </grfnem:ExpNode>
```

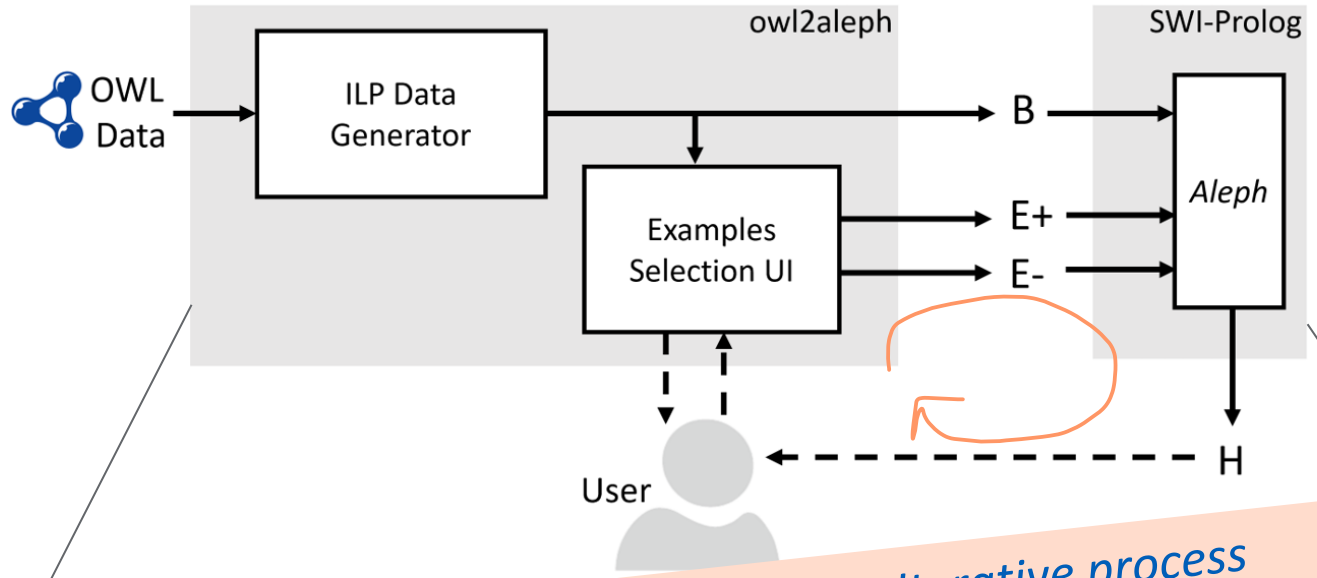
Expert

GraSEN ILP

```
constant(A) :-
   outputs(A,B), var_explicit(B), var_assigned_once(B),
   xfunction(A,C), func_not_in_loop_block(C).
```

Prolog program

Approach



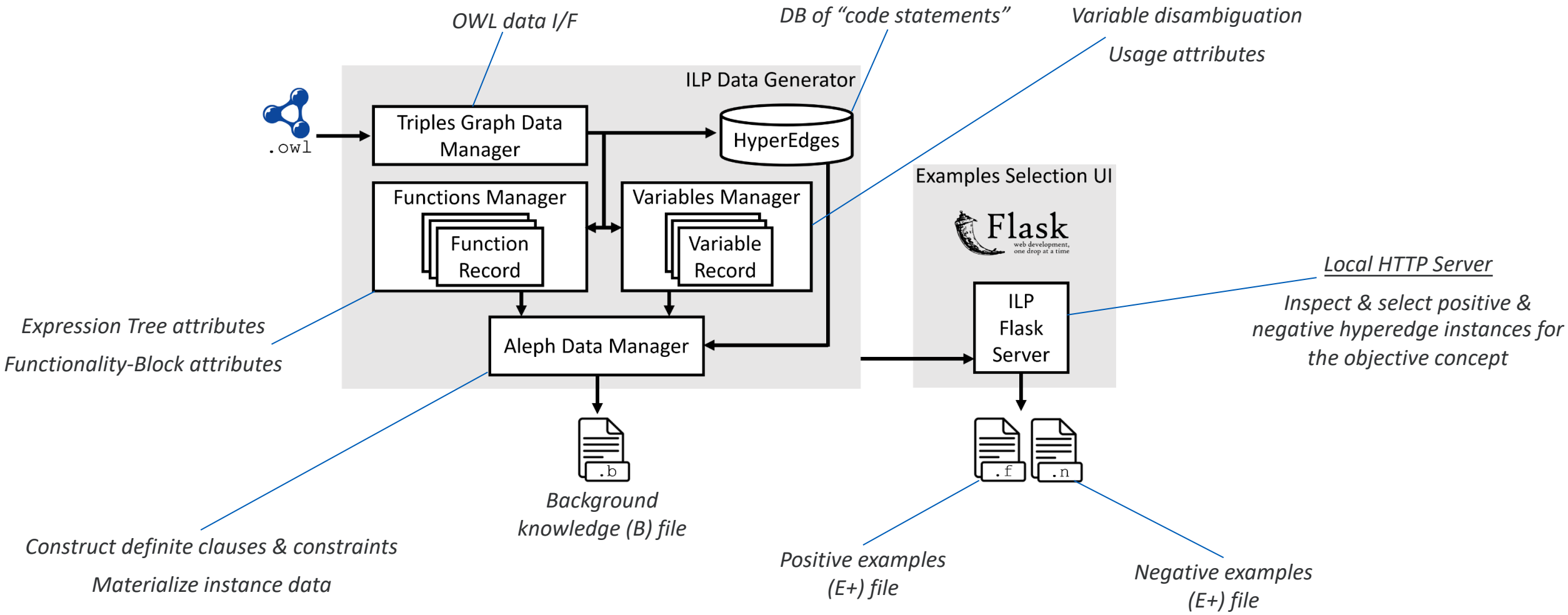
- Pythonic module
- Translates OWL to background knowledge (B)
  - Rules & instances
- Invokes an interactive UI
  - Inspect & select positive & negative instances (E+, E-)

Iterative process

Refine E+/E- & re-run until a satisfactory rule is achieved

- Runs *Aleph*
- Induce a hypothesized clause (H)
- Query new positives

Generating the ILP Data (owl2aleph)



Generating the ILP Data (owl2aleph)

```
10 :- modeh(1,newfeature(+hyperedge)).
```

hypothesis definition

```
30 :- modeb(*,outputs_of(+xvariable,-hyperedge)).
31 :- modeb(*,xfunction(+hyperedge,-xnode)).
32 :- modeb(*,has_operator_mult(+xnode)).
33 :- modeb(*,has_operator_ifexpr(+xnode)).
34 :- modeb(*,xinterface(+xnode)).
35 :- modeb(*,var_assigned_once(+xvariable)).
36 :- modeb(*,var_multi_assigned(+xvariable)).
```

signatures of functions

```
50 :- determination(newfeature/1,outputs/2).
51 :- determination(newfeature/1,xliteral/1).
52 :- determination(newfeature/1,xfunction/2).
53 :- determination(newfeature/1,inputs/2).
54 :- determination(newfeature/1,xassign/1).
```

concepts for rules

```
86 hyperedge(xa1f96097abb845469e519fb7f237f9f9).
87 outputs(xa1f96097abb845469e519fb7f237f9f9,variable3).
88 xnode(node2).
89 xassign(node2).
90 has_operator_div(node2).
91 xfunction(xa1f96097abb845469e519fb7f237f9f9,node2).
```

instance data

Variable disambiguation

Usage attributes

Local HTTP Server

Inspect & select positive & negative hyperedge instances for the objective concept

ILP Flask Server

knowledge (B) file

Construct definite clauses & constraints

Materialize instance data

```
1
2
3 newfeature(x22528789304c467f88c2a707bdaf28c1).
4 newfeature(xcf986e40683c402fafaefac6dbcbdfac).
5
6
```

positive/negative hyperedges

Generating the ILP Data (owl2aleph)

```
10 :- modeh(1,newfeature(+hyperedge)).
```

hypothesis definition

GraSEN ILP Sandbox

generate .f & .n files

Clear | Generate Example Files

Positives:

newfeature(xf3407a4281654856a4f1d692e51d58f9).
newfeature(xf899593b6b954b098b7a9fdf8aba0bab).

positive hyperedges

Negatives:

newfeature(xb1b9c84b70824e668b52a527a20f9298).

negative hyperedges

HyperEdge	line start	ftype	lambda	add to...
xf849fbef3f59455c8d9b0aae61a7fe02	16	xassign	`lambda timestep,error,integrator_state: ((timestep * error) + integrator_state)`	+ -
xfbcab097a142468380f366753096577e	18	xassign	`lambda error,Kp_M,Ki_M,integrator_state: ((error * Kp_M) + (Ki_M * integrator_state))`	+ -
x80e45762c9184fdca6728b0348263cc0	21	xassign	`lambda input,gain: (input * gain)`	+ -
x3f03818ce2464beb899eac82f12e0d0d	27	xliteral	`lambda : 100.5`	+ -

add as positive

add as negative

positive/negative hyperedges

Rule Generation from ILP Data (SWI-Prolog)

- *Aleph*
  - Takes *{.b, .f, .n} files as input*
  - Runs via SWI-Prolog
  - Outcome is a classification rule in domain terms

- The classification rules generated can be:
  - Inspected and used to query all positives
  - Re-generated with new data
    - New examples (+/-)
    - New knowledge in the form of clauses
  - Manually refined & vetted

Bottom (most-specific) clause

```
[bottom clause]
newfeature(A) :-
   outputs(A,B), xfunction(A,C), single_output_edge(A), var_explicit(B),
   inputs_of(B,D), outputs_of(B,A), xfunction_of(C,A), xliteral(C),
   func_not_in_loop_block(C), var_assigned_once(B).
```

+ positive examples	- negative examples
double Ki_M = 20.0; **double** Kp_M = 75.0;	**double** error = Input_dmd - Input_sensed;
int num_steps = t_final / time_step;	sensed_output = 0.0;

Best (reduced) clause

```
newfeature(A) :-
   xfunction(A,B), xliteral(B).

[Training set performance]
          Actual
          +          -
      + 2          0          2
Pred
      - 0          1          1

          2          1          3

Accuracy = 1
[Training set summary] [[2,0,0,1]]
[time taken] [0.015625]
```

Best (reduced) clause

```
newfeature(A) :-
   outputs(A,B), var_assigned_once(B), xfunction(A,C),
   func_not_in_loop_block(C).

[Training set performance]
          Actual
          +          -
      + 3          0          3
Pred
      - 0          2          2

          3          2          5

Accuracy = 1
[Training set summary] [[3,0,0,2]]
[time taken] [0.015625]
```

```
?- newfeature(X).
X = x83c119d8602b4636a7134fb92274fac8 ;
X = xc4f28673610d4ef7b737a51f5d502c09 ;
X = x40160a643a2b48d6a6754a2bd255b948 ;
```

Evaluation

- Dataset
  - three OWL files
  - 8,974 triples
  - 61 math & control instances
    - 9 different classes of concepts
- Results
  - 7/9: perfect F1 score
    - Small training data
    - Significant reduction in # literals
  - 2/9: low F1
    - not enough data to separate positives from negatives
  - Time complexity is excellent < 1s

Concept	(E+, E−)	Bottom clause size	Reduced clause size	Time [seconds]	True positives	False positives	False negatives	Precision	Recall	F1
Difference	(2, 1)	24	3	0.063	4	0	0	1.0	1.0	**1.0**
Sum	(2, 2)	29	3	0.063	9	0	0	1.0	1.0	**1.0**
Gain	(2, 1)	23	3	0.047	9	0	0	1.0	1.0	**1.0**
Division	(2, 2)	24	3	0.094	3	0	0	1.0	1.0	**1.0**
Constant	(2, 5)	11	6	0.125	23	0	0	1.0	1.0	**1.0**
Error signal	(2, 5)	24	6	0.859	2	0	0	1.0	1.0	**1.0**
PI controller	(2, 5)	45	5	0.453	3	0	0	1.0	1.0	**1.0**
Output signal	(2, 3)	12	12	0.859	3	3	0	0.50	1.0	0.67
Reference signal	(2, 2)	11	11	0.375	3	17	0	0.15	1.0	0.26
Switch	Not enough positives ($E+$)									
Magnitude saturation										
PID controller										

Evaluation

- Dataset
  - three OW
  - 8,974 trip
  - 61 math & control instances
    - 9 different classes of concepts
- Results
  - 7/9: perfect F1 score
    - Small training data
    - Significant reduction in # literals
  - 2/9: low F1
    - not enough data to
      positives from nega
  - Time complexity is

Concept	(E+, E−)	Bottom clause size	Reduced clause size	Time [seconds]	True positives	False positives	False negatives	Precision	Recall	F1
Difference	(2, 1)	24	3	0.063	4	0	0	1.0	1.0	**1.0**
				063	9	0	0	1.0	1.0	**1.0**
				047	9	0	0	1.0	1.0	**1.0**
										1.0
										1.0
										1.0
										1.0
Output signal	(2, 3)	12	12	0.859	3	3	0	0.50	1.0	0.67
Reference signal	(2, 2)	11	11	0.375	3	17	0	0.15	1.0	0.26
Switch	Not enough positives ($E+$)									
Magnitude saturation										

```
gain(A) :-
   xfunction(A,B), has_operator_mult(B).
```

```
constant(A) :-
   outputs(A,B), var_explicit(B), var_assigned_once(B),
   xfunction(A,C), func_not_in_loop_block(C).
```

```
picontroller(A) :-
   outputs(A,B), var_implicit(B),
   xfunction(A,C), has_operator_add(C).
```

Discussion

- Platform is feasible & effective in terms of time, completeness & robustness
  - Results are satisfying (simple & composite concepts)
  - Quality of ILP generated rules depends on supplied input
- Still, many challenges exist:
  - Local vs. global variables (structs, enumerations, etc…)
  - In-line code (implicit vs. explicit)
  - Pointers
  - Implementation “style” (e.g., saturation: if/else vs. arithmetic)
- Looking forward:
  - Rules can be folded back into the semantic model
    - Learn multiple level of knowledge
  - Use probabilistic soft-logic on instance data to introduce confidence
  - Joint-classification
    - Incorporate higher level (or system) knowledge (e.g., there is a single integrator in the code)
  - Augmentation
    - Layout features
    - Memory allocations

Conclusion

- An automated platform for generating classification rules for math & control concepts
  - SME not required to know language formalism to describe knowledge in detail
  - Fast, iterative induction with human-in-the-loop
  - Structured & semantic
  - Easily extended & refined, automatically or manually
  - Handy during design/engineering
    - Incorporate feedback
    - Generate explanations