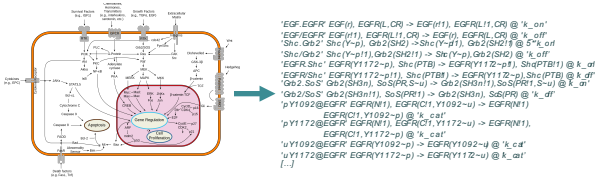
Logical Inference for Rule-Based Biological Models

Chelsea Voss (MIT), csvoss@mit.edu

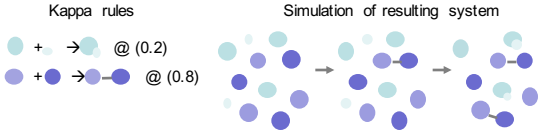
Goal: executable models

- Executable models help researchers examine systems and develop hypotheses.
- Example: models of cancer pathways.



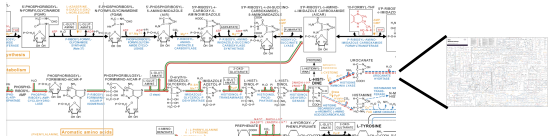
```
'EGF.EGFR' EGF(r), EGFR(L,CR) -> EGF(r!1), EGFR(L!1,CR) @ 'k_on'
'EGF/EGFR' EGF(r!1), EGFR(L!1,CR) -> EGF(r), EGFR(L,CR) @ 'k_off'
'Shc.Gb2' Shc(Y~p), Gb2(SH2) -> Shc(Y~p!1), Gb2(SH2!1) @ 5*k_on
'Shc/Gb2' Shc(Y~p!1),Gb2(SH2!1) -> Shc(Y~p),Gb2(SH2) @ 'k_off'
'EGFR.Shc' EGFR(Y1172~p), Shc(PTB) -> EGFR(Y1172~p!1), Shc(PTB!1) @ k_on
'EGFR/Shc' EGFR(Y1172~p!1), Shc(PTB!1) -> EGFR(Y1172~p),Shc(PTB) @ k_off
'Gb2.SoS' Gb2(SH3n), SoS(PR,S~u) -> Gb2(SH3n!1),SoS(PR!1,S~u) @ k_on
'Gb2/SoS' Gb2(SH3n!1), SoS(PR!1) -> Gb2(SH3n), SoS(PR) @ 'k_off'
'pY1092@EGFR' EGFR(Nt!1), EGFR(Cl!1,Y1092~u) -> EGFR(Nt!1)
                EGFR(Cl!1, Y1092~p) @ 'k_cat'
'pY1172@EGFR' EGFR(Nt!1), EGFR(Cl!1,Y1172~u) -> EGFR(Nt!1),
                EGFR(Cl!1,Y1172~p) @ 'k_cat'
'uY1092@EGFR' EGFR(Y1092~p) -> EGFR(Y1092~u) @ 'k_cat'
'uY1172@EGFR' EGFR(Y1172~p) -> EGFR(Y1172~u) @ 'k_cat'
[...]
```

- Such models have applications in personalized medicine, drug design, and basic research.
- **Kappa** is a language used to program and run rule-based biological models, with one rule per reaction.



Automatic model generation

- Biological knowledge is immense, complex, and always changing.
- Old models might not reflect new research.



- Researchers are working to fix this by generating rule-based models from NLP on research papers.

The problem

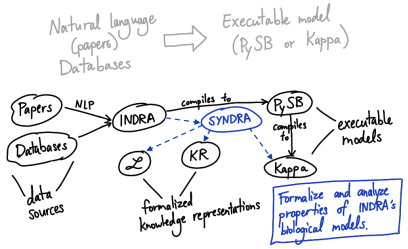
- Statements produced by NLP may be ambiguous or incomplete.
- We want to analyze these statements, and perform logical queries over them to detect issues.

```
A is a member of protein family P.
B is a member of protein family P.
A phosphorylates C at Ser-222.
B activates C.
```

→ *Does B phosphorylate C? Can we infer that B phosphorylates C from any other rules in the model? Would it contradict any other rule in the model for B to phosphorylate C?*

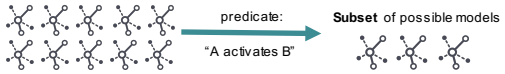
Contributions

- Developed approach for integrating semantic and chemical reasoning.
- Implemented tool, Syndra, using Python and Z3.
- Demonstrated feasibility of tool in resolving logical inferences on small real-world examples.
- Integrated tool with model generation framework, including NLP frontend.



How it works

- Translate NLP statements into predicates.
- Predicates constrain the space of possible models.



- Can perform logical queries over this space.

A logic for biological models

- Implemented **Iota**, a logic language designed to describe valid Kappa programs. [Husson & Krivine]
- Example: predicate "phosphorylated A binds to B" –

$$PreLabeled(A, phosphorylated) \wedge PreUnbound(A, B)$$
$$\wedge PostLabeled(A, phosphorylated) \wedge PostBound(A, B)$$

- A model is a set of reaction rules; each reaction rule is described formally by transformations on a graph.
- A predicate specifies each graph transformation by preconditions and postconditions on the graph state.

Implementing the logic

- Defined Z3 datatypes for Iota components.
- Implemented each logical operator of Iota.

Results

Functionality

Supports the following user-facing functionality:

- *Check satisfiability* of any predicate.
- *Construct a model* satisfying a predicate.
- *Check implication* between two predicates.

Proving implications

- Define *macros* for commonly-used predicates, formally defining what rules mean.
- Can prove that two rules imply another statement. Consider the example:

Rules

1. Protein A phosphorylates protein B.
2. When phosphorylated, protein B is active.
3. Protein A activates protein B.

Question: *do 1 and 2 necessarily imply 3?* ***Yes****, we can prove that they do!*

```
>>> p1 = macros.directly_phosphorylates("A", "B")
>>> p2 = macros.phosphorylated_is_active("B")
>>> p3 = macros.directly_activates("A", "B")
>>> predicate.And(p1, p2).check_implies(p3)
True
```

- This allows us to prove inference rules correct in terms of the underlying chemistry of each rule.

Interfacing with an NLP model-generator

- **INDRA** is a tool which generates Kappa models from NLP.
- Can convert INDRA statements into *iota* predicates, and perform queries over those predicates.
- Example: can detect that **s4** is redundant!

INDRA statements...

```
s1 = Phosphorylation(MAP2K1, MAPK1, PhosphorylationThreonine, 183)
s2 = Phosphorylation(MAP2K1, MAPK1, PhosphorylationTyrosine, 185)
s3 = ActivityModification(MAPK1, ['PhosphorylationThreonine',
    'PhosphorylationTyrosine'], ['183', '185'], increases, Activity)
s4 = ActivityActivity(MAP2K1, Kinase, increases, MAPK1, Kinase)
```

...converted to Syndra predicates

```
>>> pred = syndra_from_statements(s1, s2, s3)
>>> pred.check_sat()
True
>>> pred.check_implies(syndra_from_statements(s4))
True
```