Better call Saul:
Flexible Programming for Learning and Inference in NLP

Parisa Kordjamshidi, *Tulane University, pkordjam@tulane.edu*
Daniel Khashabi, *University of Illinois at Urbana-Champaign*
Christos Christodoulopoulos, *University of Illinois at Urbana-Champaign*
Bhargav Mangipudi, *University of Illinois at Urbana-Champaign*
Sameer Singh, *University of California, Irvine*
Dan Roth, *University of Illinois at Urbana-Champaign*

COLING-2016
December 2016, Osaka, Japan

WHAT

WHAT

- Introducing Saul which is a declarative Learning based programming language. [Kordjamshidi et. al. IJCAI-2015]

WHAT

- Introducing Saul which is a declarative Learning based programming language. [Kordjamshidi et. al. IJCAI-2015]

- Particularly, the way we have augmented it with the abstraction levels and facilities for designing various NLP tasks with **arbitrary output structure** with **various linguistic granularities**.

WHAT

- Introducing Saul which is a declarative Learning based programming language. [Kordjamshidi et. al. IJCAI-2015]

- Particularly, the way we have augmented it with the abstraction levels and facilities for designing various NLP tasks with **arbitrary output structure** with **various linguistic granularities**.

  - Word level
  - Phrase level
  - Sentence level …

WHY

WHY

Most of the NLP learning tasks seek for a mapping from input structures to output structures.

WHY

Most of the NLP learning tasks seek for a mapping from input structures to output structures.

- Syntactic => Part of speech tagging
- Information Extraction => Entity mention / Relation extraction
- Semantic => Semantic role labeling

WHY

Most of the NLP learning tasks seek for a mapping from input structures to output structures.

- Syntactic => Part of speech tagging
- Information Extraction => Entity mention / Relation extraction
- Semantic => Semantic role labeling

We often need to make a lot of programming effort and hard code to:

WHY

Most of the NLP learning tasks seek for a mapping from input structures to output structures.

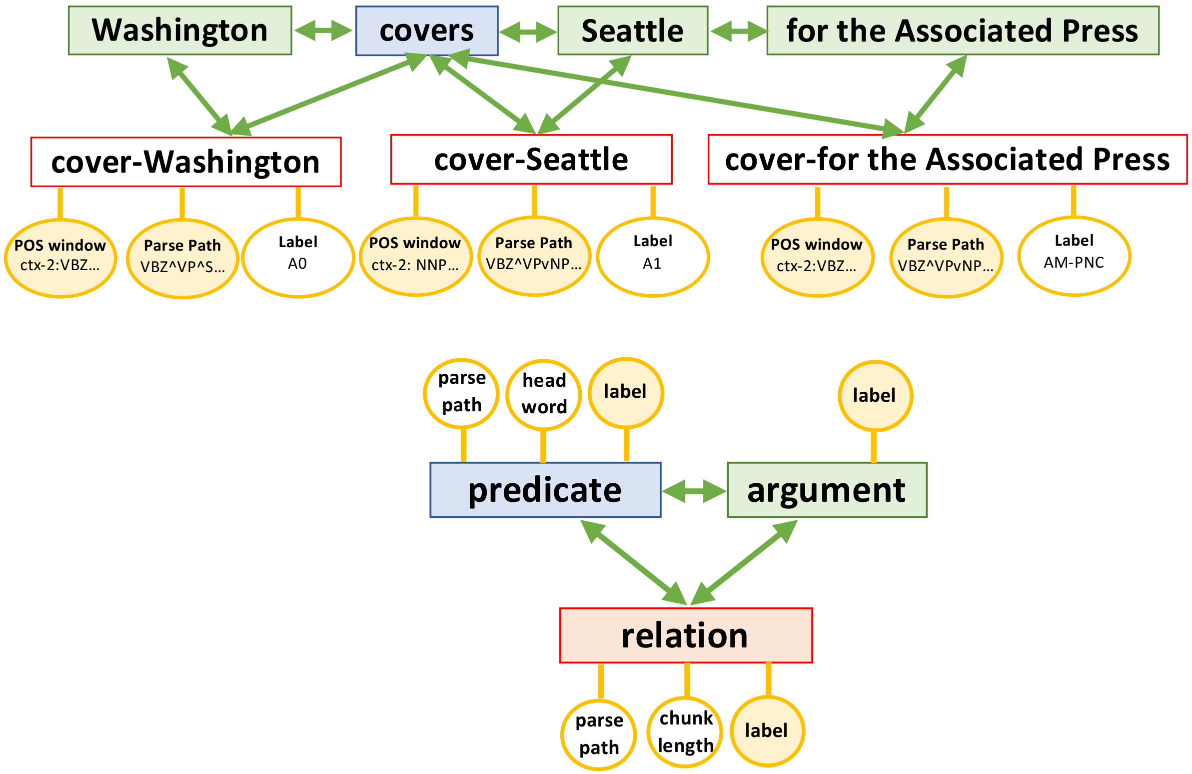
- Syntactic => Part of speech tagging
- Information Extraction => Entity mention / Relation extraction
- Semantic => Semantic role labeling

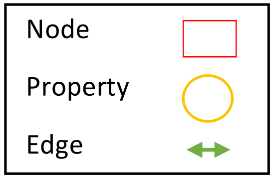
We often need to make a lot of programming effort and hard code to:

- Benefit from a specific structure
- (Relational) Feature extraction (Even when using representation learning techniques)

Example Tasks (1)

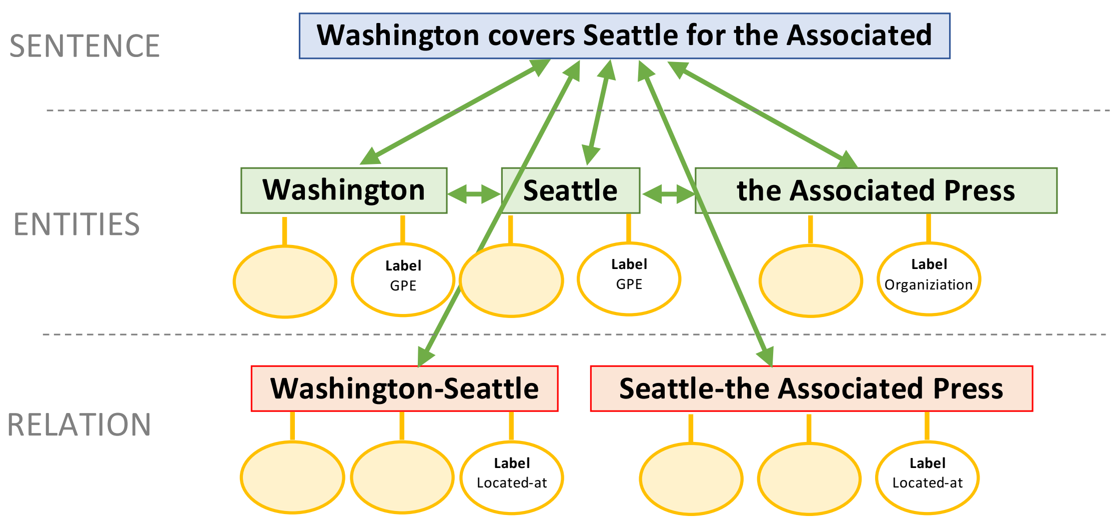
Semantics: Semantic role labeling





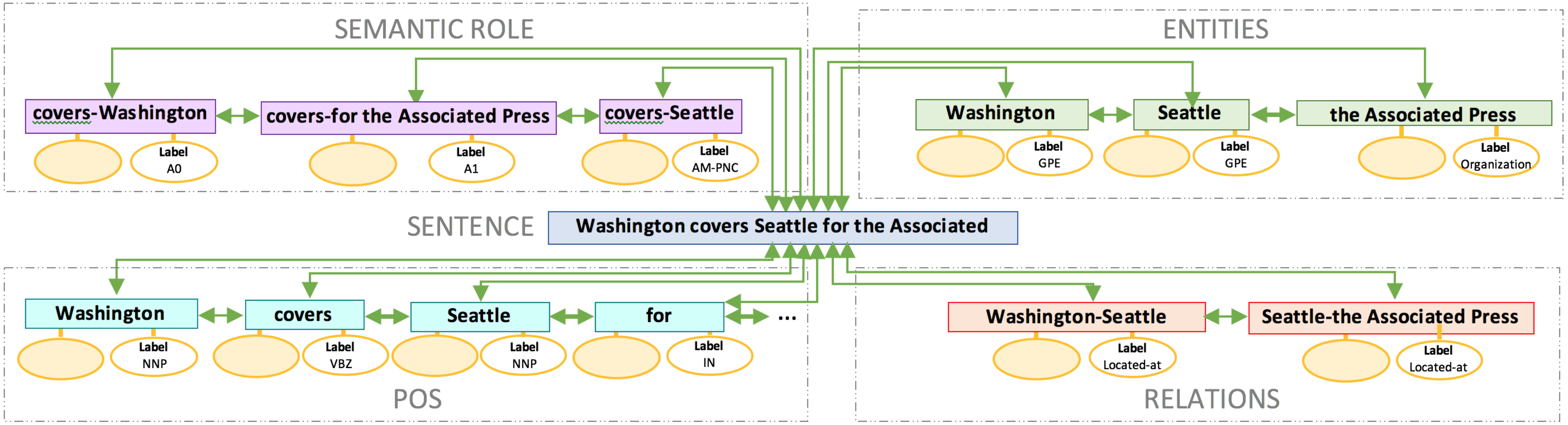
Example Tasks (2)

Information extraction: Entity mention relation extraction



Example Tasks (3)

All annotations: Syntax, Semantics, Mentions, Relations …



Structured Output Models: Common Practice

For extraction and representation of features as well as for exploiting global output structure we do

Structured Output Models: Common Practice

For extraction and representation of features as well as for exploiting global output structure we do

- Task Specific Programming for Data Structures
- Model Specific Programming for Inference and Learning
- It will be hard to generalize
- It will be hard to Reuse and Reproduce results

Structured Output Models:
Learning and Inference Paradigms

Structured Output Models: Learning and Inference Paradigms

- Local Models: Local classifiers trained/output components predicted independently (LO)

Structured Output Models: Learning and Inference Paradigms

- Local Models: Local classifiers trained/output components predicted independently (LO)
- Pipelines

Structured Output Models: Learning and Inference Paradigms

- Local Models: Local classifiers trained/output components predicted independently (LO)
- Pipelines
- Global Models
  - L+I: Training LO, global prediction
  - IBT: Global training and global prediction

Saul and NLP

Saul and NLP

Idea: High level abstraction for programming various configurations from local to global learning as well as building pipelines over NLP data structures.

Saul and NLP

Idea: High level abstraction for programming various configurations from local to global learning as well as building pipelines over NLP data structures.

- Data Model
  - Graph (typed nodes, edges and properties)
  - Sensors (black box functions that operate on graph's base types)

Saul and NLP

Idea: High level abstraction for programming various configurations from local to global learning as well as building pipelines over NLP data structures.

- Data Model
  - Graph (typed nodes, edges and properties)
  - Sensors (black box functions that operate on graph's base types)
- Templates for learning and inference decomposition
  - Classifiers
  - Constraints

Underlying Computational Model

Underlying Computational Model

Learning: $h : \mathcal{X} \rightarrow \mathcal{Y}$

Underlying Computational Model

Learning: $h : \mathcal{X} \to \mathcal{Y}$

Structured output learning: $g : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$

Underlying Computational Model

Learning: $h : \mathcal{X} \to \mathcal{Y}$

Structured output learning: $g : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$

$$h(x; W) = \arg\max_{y \in \mathcal{Y}} g(x, y; W)$$

Underlying Computational Model

Learning: $h : \mathcal{X} \rightarrow \mathcal{Y}$

Structured output learning: $g : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}$

Inference

$$h(x; W) = \arg\max_{y \in \mathcal{Y}} g(x, y; W)$$

Underlying Computational Model

Learning: $h : \mathcal{X} \rightarrow \mathcal{Y}$

Structured output learning: $g : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}$

Inference

Decoding/ Prediction time inference

$$h(x; W) = \arg\max_{y \in \mathcal{Y}} g(x, y; W)$$

Underlying Computational Model

Learning: $h : \mathcal{X} \to \mathcal{Y}$

Structured output learning: $g : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$

Decoding/ Prediction time inference

Inference

$$h(x; W) = \arg\max_{y \in \mathcal{Y}} g(x, y; W)$$

$$g(x, y; W) = \langle W, f(x, y) \rangle$$

Underlying Computational Model

Learning: $h : \mathcal{X} \to \mathcal{Y}$

Structured output learning: $g : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$

Decoding/ Prediction time inference

Inference

$$h(x; W) = \arg\max_{y \in \mathcal{Y}} g(x, y; W)$$

$$g(x, y; W) = \langle W, f(x, y) \rangle$$

Weight vector

Underlying Computational Model

Learning: $h : \mathcal{X} \rightarrow \mathcal{Y}$

Structured output learning: $g : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}$

Decoding/ Prediction time inference

Inference

$$h(x;W) = \arg\max_{y \in \mathcal{Y}} g(x,y;W)$$

$$g(x,y;W) = \langle W, f(x,y) \rangle$$

Weight vector

Joint feature function

Underlying Computational Model

Learning: $h : \mathcal{X} \rightarrow \mathcal{Y}$

Structured output learning: $g : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}$

Decoding/ Prediction time inference

Inference

$$h(x; W) = \arg\max_{y \in \mathcal{Y}} g(x, y; W)$$

$$g(x, y; W) = \langle W, f(x, y) \rangle$$

Weight vector

Joint feature function

$$l(W) = \sum_{i=1}^{N} \max_{y \in \mathcal{Y}} (g(x^i, y; W) - g(x^i, y^i; W) + \Delta(y^i, y))$$

Underlying Computational Model : Input/Output

Joint feature function

$$g(x, y; W) = \langle W, f(x, y) \rangle$$

Weight vector

$$\{x_1..x_K\} \qquad \mathbf{l} = \{l_1, .., l_P\}$$

$$g(x, y; W) = \sum_{l_p \in \mathbf{l}} \sum_{x_k \in C_{l_p}} \langle W_p, f_p(x_k, l_p) \rangle = \sum_{l_p \in \mathbf{l}} \sum_{x_k \in C_{l_p}} \langle W_p, \phi_p(x_k) \rangle l_{pk} =$$

$$\sum_{l_p \in \mathbf{l}} \langle W_p, \sum_{x_k \in C_{l_p}} (\phi_p(x_k) l_{pk}) \rangle$$

Underlying Computational Model : Input/Output

Joint feature function

$$g(x, y; W) = \langle W, f(x, y) \rangle$$

Weight vector

$\{x_1..x_K\}$ $\quad \mathbf{l} = \{l_1, .., l_P\}$

$$g(x, y; W) = \sum_{l_p \in \mathbf{l}} \sum_{x_k \in C_{l_p}} \langle W_p, f_p(x_k, l_p) \rangle = \sum_{l_p \in \mathbf{l}} \sum_{x_k \in C_{l_p}} \langle W_p, \phi_p(x_k) \rangle l_{pk} =$$

$$\sum_{l_p \in \mathbf{l}} \langle W_p, \sum_{x_k \in C_{l_p}} (\phi_p(x_k) l_{pk}) \rangle$$

in addition to the **constraints between labels**!

Underlying Computational Model : Global Constraints

Constrained Conditional Models (CCM)

[Roth & Yih ‘04, 07; Chang, et.al.,’08,’12]

Underlying Computational Model : Global Constraints

Constrained Conditional Models (CCM)

- Prediction function: assign values that maximize objective

[Roth & Yih ‘04, 07; Chang, et.al.,’08,’12]

Underlying Computational Model : Global Constraints

Constrained Conditional Models (CCM)

- Prediction function: assign values that maximize objective

$$h(x) = \arg\max_{y \in Y(x)} g(x, y; W)$$

[Roth & Yih '04, 07; Chang, et.al.,'08,'12]

Underlying Computational Model : Global Constraints

Constrained Conditional Models (CCM)

- Prediction function: assign values that maximize objective

$$h(x) = \arg\max_{y \in Y(x)} g(x, y; W)$$

[Roth & Yih '04, 07; Chang, et.al.,'08,'12]

Underlying Computational Model : Global Constraints

Constrained Conditional Models (CCM)

- Prediction function: assign values that maximize objective

$$h(x) = \arg\max_{y \in Y(x)} g(x, y; W)$$

[Roth & Yih '04, 07; Chang, et.al.,'08,'12]

Underlying Computational Model : Global Constraints

Constrained Conditional Models (CCM)

- Prediction function: assign values that maximize objective

$$h(x) = \arg\max_{y \in Y(x)} g(x, y; W)$$

- Objective is linear in features and constraints

[Roth & Yih '04, 07; Chang, et.al.,'08,'12]

Underlying Computational Model : Global Constraints

Constrained Conditional Models (CCM)

- Prediction function: assign values that maximize objective

$$h(x) = \arg\max_{y \in Y(x)} g(x, y; W)$$

- Objective is linear in features and constraints

[Roth & Yih ‘04, 07; Chang, et.al.,’08,’12]

Underlying Computational Model : Global Constraints

Constrained Conditional Models (CCM)

- Prediction function: assign values that maximize objective

$$h(x) = \arg\max_{y \in Y(x)} g(x, y; W)$$

- Objective is linear in features and constraints

[Roth & Yih '04, 07; Chang, et.al.,'08,'12]

Underlying Computational Model : Global Constraints

Constrained Conditional Models (CCM)

- Prediction function: assign values that maximize objective

$$h(x) = \arg\max_{y \in Y(x)} g(x, y; W)$$

- Objective is linear in features and constraints

$$g = \langle W, f(x, y) \rangle - \langle \rho, c(x, y) \rangle$$

[Roth & Yih ‘04, 07; Chang, et.al.,’08,’12]

Underlying Computational Model : Global Constraints

Constrained Conditional Models (CCM)

- Prediction function: assign values that maximize objective

$$h(x) = \arg\max_{y \in Y(x)} g(x, y; W)$$

- Objective is linear in features and constraints

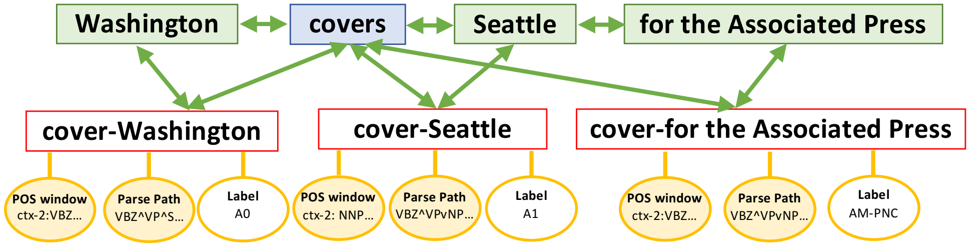
$$g = \langle W, f(x, y) \rangle - \langle \rho, c(x, y) \rangle$$

Compile everything in an Integer Linear Program: expressive enough to support decision making in the context of any probabilistic modeling.

[Roth & Yih ‘04, 07; Chang, et.al.,’08,’12]

Semantic Role Labeling : Data Model

```
val sentences = node[TextAnnotation]
val predicates = node[Constituent]
val arguments = node[Constituent]
val pairs =  node[Relations]
val pos-tag = property(arguments)
val word-form = property(arguments)
val relationsToArguments = edge(relations, arguments)
relationsToArguments.addSensor(relToArgument _)
```



A graph in terms of typed nodes, edges and Properties

Semantic Role Labeling : Classifiers

$\boldsymbol{x} = \{x_1, \ldots, x_4\}$

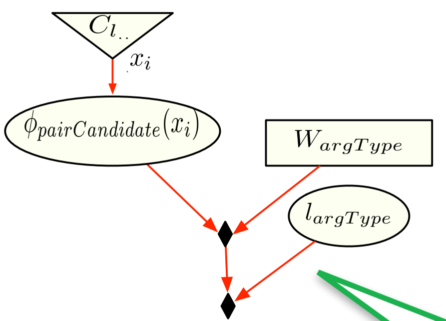
$\boldsymbol{l} = \{l_{isPred}, l_{isArg}, l_{argType}\}$

link-label

single-label

$\phi_{constituent}(x_i)$

$\phi_{pair}(x_i, x_j)$




```
object ArgTypeLearner extends Learnable(pairs){
    def label = argumentLabelGold
    def feature = using(containsMOD, containsNEG,
    clauseFeatures, chunkPathPattern, chunkEmbedding,
    chunkLength, constituentLength, argPOSWindow,
    argWordWindow, headwordRelation, syntacticFrameRelation
    , pathRelation, phraseTypeRelation, predPosTag,
    predLemmaR, linearPosition)
```

a Learning Template for argument types

Semantic Role Labeling : Combined Feature Functions

- Single or composed components of the input are represented with typed nodes in the graph
- All features are defined as the properties of the nodes
- Labels also applied to single components or composed components of the input (called link labels in the latter case)
- Edges are established between nodes
- The edges and properties are defined and computed based on a set of given **NLP sensors**

Semantic Role Labeling : Constraints

Only legal arguments of a predicate could be assigned as a type to the candidate arguments. The legality is checked according to the Propbank frames.

```
val legalArgumentsConstraint = constraint(sentences) { x =>
    val constraints = for {
      predicate <- sentences(x) ~> sentenceToPredicates
      candidateRelations = (predicates(y) ~> -relationsToPredicates)
      argLegalList = legalArguments(y)
      relation <- candidateRelations
    } yield classiferLabelIsLegal(argumentTypeLearner, relation, argLegalList)
        or (argumentTypeLearner on relation is "none")
}

def classiferLabelIsLegal(classifier, relation, legalLabels) = {
    legalLabels._exists { l => (classifier on relation is l) }
}
```

Semantic Role Labeling : Constrained Classifiers

```
object ArgTypeConstraintClassifier extends ConstrainedClassifier(ArgTypeLearner)
{
    def subjectTo = srlConstraints
}
```

This Constrained Classifier now applies on a given pair candidate, BUT it uses the global constraints at the sentence level. The sentence is accessed via the edges defined in the data model that connect the relation to its original sentence.

Program Structure

```
val srlDataModelObject = PopulateSRLDataModel(…)

val AllMyConstrainedClassifiers= List(argTypeConstraintClassifier,…)

JointTrain(sentences, AllMyConstrainedClassifiers)

ClassifierUtils.TestClassifiers(AllMyConstrainedClassifiers)
```

Program Structure

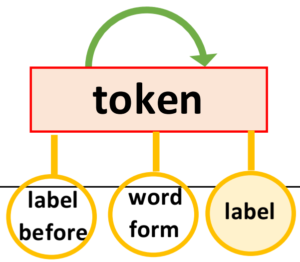
```
val srlDataModelObject = PopulateSRLDataModel(…)

val AllMyConstrainedClassifiers= List(argTypeConstraintClassifier,…)

JointTrain(sentences, AllMyConstrainedClassifiers)

ClassifierUtils.TestClassifiers(AllMyConstrainedClassifiers)
```

Same amount of code for other paradigms!

Other tasks

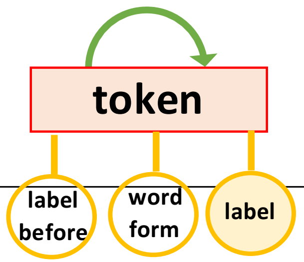
Other tasks

```
val labelTwoBefore = property(tokens) { x: Constituent =>
  // Use edges to jump to the previous constituent
  val cons = (tokens(x) ~> constituentBefore ~> constituentBefore).head
  if (POSTaggerKnown.isTraining)
    POSLabel(cons)
  else POSTaggerKnown(cons)
}
```



Other tasks

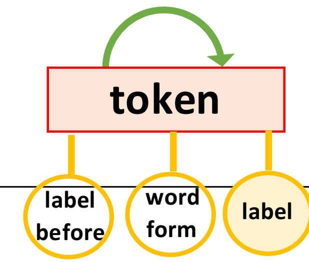
```
val labelTwoBefore = property(tokens) { x: Constituent =>
  // Use edges to jump to the previous constituent
  val cons = (tokens(x) ~> constituentBefore ~> constituentBefore).head
  if (POSTaggerKnown.isTraining)
    POSLabel(cons)
  else POSTaggerKnown(cons)
}
```



Pos-Tag Contextual Feature

Other tasks

```
val labelTwoBefore = property(tokens) { x: Constituent =>
  // Use edges to jump to the previous constituent
  val cons = (tokens(x) ~> constituentBefore ~> constituentBefore).head
  if (POSTaggerKnown.isTraining)
    POSLabel(cons)
  else POSTaggerKnown(cons)
}
```

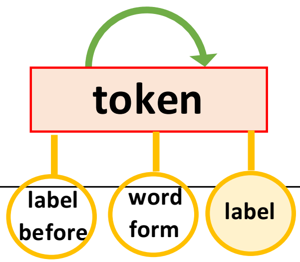


Pos-Tag Contextual Feature

```
(WorkFor(x) is true then PER(x.firstArg) is true and ORG(x.secondArg) is true
```

Other tasks

```
val labelTwoBefore = property(tokens) { x: Constituent =>
  // Use edges to jump to the previous constituent
  val cons = (tokens(x) ~> constituentBefore ~> constituentBefore).head
  if (POSTaggerKnown.isTraining)
    POSLabel(cons)
  else POSTaggerKnown(cons)
}
```



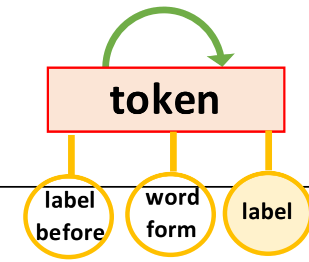
Pos-Tag Contextual Feature

```
(WorkFor(x) is true then PER(x.firstArg) is true and ORG(x.secondArg) is true
```

ER constraint imposed on mentioned and relations

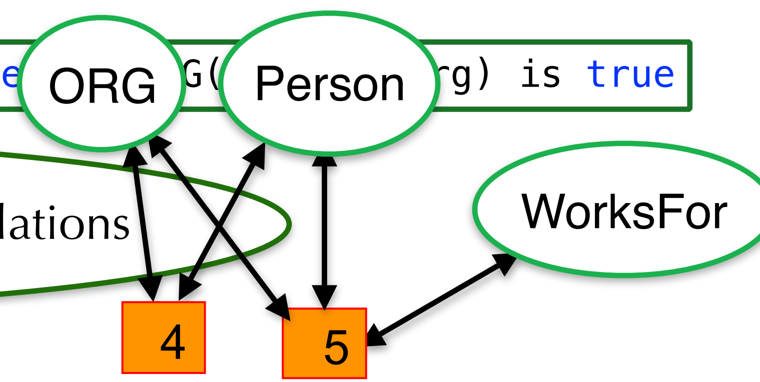
Other tasks

```
val labelTwoBefore = property(tokens) { x: Constituent =>
  // Use edges to jump to the previous constituent
  val cons = (tokens(x) ~> constituentBefore ~> constituentBefore).head
  if (POSTaggerKnown.isTraining)
    POSLabel(cons)
  else POSTaggerKnown(cons)
}
```



Pos-Tag Contextual Feature

(WorkFor(x) is true then PER(x.firstArg) is true ... G(... rg) is true



ER constraint imposed on mentioned and relations

Results : Semantic Role Labeling

Model	Precision	Recall	F1
`ArgTypeLearner`$^{\mathrm{G}}$(GOLDPREDS)	85.35	85.35	85.35
`ArgTypeLearner`$^{\mathrm{G}}$(GOLDPREDS) + C	85.35	85.36	85.35
`ArgTypeLearner`$^{\mathrm{Xue}}$(GOLDPREDS)	82.32	80.97	81.64
`ArgTypeLearner`$^{\mathrm{Xue}}$(GOLDPREDS) + C	82.90	80.70	81.79
`ArgTypeLearner`$^{\mathrm{Xue}}$(PREDPREDS)	82.47	80.79	81.62
`ArgTypeLearner`$^{\mathrm{Xue}}$(PREDPREDS) + C	83.62	80.54	82.05
`ArgIdentifier`$^{\mathrm{Xue}}$ $\mid$ `ArgTypeLearner`$^{\mathrm{Xue}}$(PREDPREDS)	82.55	81.59	82.07
`ArgIdentifier`$^{\mathrm{G}}$(PREDPREDS)	95.51	94.19	94.85

Table 1: Evaluation of SRL various labels and configurations. The superscripts over the different `Learner`s refer to the whether gold argument boundaries (G) or the Xue-Palmer heuristics (Xue) were used to generate argument candidates as input. GOLD/PREDPREDS refers to whether the `Learner` used gold or predicted predicates. ‘C’ refers to the use of constraints during prediction and $\mid$denotes the pipeline architecture.

Results: PoS-Tagging and ER

Setting	Accuracy
Count-based baseline	91.80%
Unknown Classifier	77.09%
Known Classifier	94.92 %
Combined Known-Unknown	96.69%

Table 2: The performance of the POStagger, tested on sections 22–24 of the WSJ portion of the Penn Treebank (Marcus et al., 1993).

	Scenario	Precision	Recall	F1
E	Mention Coarse-Label	77.14	70.62	73.73
	Mention Fine-Label	73.49	65.46	69.24
R	Basic	54.09	43.89	50.48
	+ Sampling	52.48	56.78	54.54
	+ Sampling + Brown	54.43	54.23	54.33
	+ Sampling + Brown + HCons	55.82	53.42	54.59

Table 3: 5-fold CV performance of the fine-grained entity (E) and relation (R) extraction on Newswire and Broadcast News section of ACE-2005.

Conclusion

https://github.com/IllinoisCogComp/saul

Conclusion

- Saul language **facilitates modeling** NLP applications that need
  - **Learning** and **reasoning** over Structures
  - **Relational Feature Extraction**
  - **D**irect use of **expert knowledge** beyond data instances

https://github.com/IllinoisCogComp/saul

Conclusion

- Saul language **facilitates modeling** NLP applications that need
  - **Learning** and **reasoning** over Structures
  - **Relational Feature Extraction**
  - **Direct** use of **expert knowledge** beyond data instances
- Saul saves **Programming time**
  - in designing various configurations and experimentation
    - each configuration is expressed in a **few lines of declarative code**.

https://github.com/IllinoisCogComp/saul

Conclusion

- Saul language **facilitates modeling** NLP applications that need
  - **Learning** and **reasoning** over Structures
  - **Relational Feature Extraction**
  - **Direct** use of **expert knowledge** beyond data instances
- Saul saves **Programming time**
  - in designing various configurations and experimentation
    - each configuration is expressed in a **few lines of declarative code.**
- Saul increases the **reusability** of codes
  - when **data** and **knowledge** about the problem **increases**
  - when we get to use **new emerging algorithms**

https://github.com/IllinoisCogComp/saul

Conclusion

- Saul language **facilitates modeling** NLP applications that need
  - **Learning** and **reasoning** over Structures
  - **Relational Feature Extraction**
  - **Direct** use of **expert knowledge** beyond data instances
- Saul saves **Programming time**
  - in designing various configurations and experimentation
    - each configuration is expressed in a **few lines of declarative code.**
- Saul increases the **reusability** of codes
  - when **data** and **knowledge** about the problem **increases**
  - when we get to use **new emerging algorithms**

The Novel Ideas include:

https://github.com/IllinoisCogComp/saul

Conclusion

- Saul language **facilitates modeling** NLP applications that need
  - **Learning** and **reasoning** over Structures
  - **Relational Feature Extraction**
  - **D**irect use of **expert knowledge** beyond data instances
- Saul saves **Programming time**
  - in designing various configurations and experimentation
    - each configuration is expressed in a **few lines of declarative code.**
- Saul increases the **reusability** of codes
  - when **data** and **knowledge** about the problem **increases**
  - when we get to use **new emerging algorithms**

The Novel Ideas include:

- Programming for decompositions of a global optimizations that are used in training and prediction

https://github.com/IllinoisCogComp/saul

Conclusion

- Saul language **facilitates modeling** NLP applications that need
  - **Learning** and **reasoning** over Structures
  - **Relational Feature Extraction**
  - **Direct** use of **expert knowledge** beyond data instances
- Saul saves **Programming time**
  - in designing various configurations and experimentation
    - each configuration is expressed in a **few lines of declarative code.**
- Saul increases the **reusability** of codes
  - when **data** and **knowledge** about the problem **increases**
  - when we get to use **new emerging algorithms**

The Novel Ideas include:

- Programming for decompositions of a global optimizations that are used in training and prediction
- An abstraction for unifying various formalisms for learning and reasoning which is an ongoing work

https://github.com/IllinoisCogComp/saul

Conclusion

- Saul language **facilitates modeling** NLP applications that need
  - **Learning** and **reasoning** over Structures
  - **Relational Feature Extraction**
  - **Direct** use of **expert knowledge** beyond data instances
- Saul saves **Programming time**
  - in designing various configurations and experimentation
    - each configuration is expressed in a **few lines of declarative code.**
- Saul increases the **reusability** of codes
  - when **data** and **knowledge** about the problem **increases**
  - when we get to use **new emerging algorithms**

The Novel Ideas include:

- Programming for decompositions of a global optimizations that are used in training and prediction
- An abstraction for unifying various formalisms for learning and reasoning which is an ongoing work
- A unified language for graph querying and structured learning

https://github.com/IllinoisCogComp/saul

Conclusion

Thank you!

https://github.com/IllinoisCogComp/saul

- Saul language **facilitates modeling** NLP applications that need
  - **Learning** and **reasoning** over Structures
  - **Relational Feature Extraction**
  - **Direct** use of **expert knowledge** beyond data instances
- Saul saves **Programming time**
  - in designing various configurations and experimentation
    - each configuration is expressed in a **few lines of declarative code.**
- Saul increases the **reusability** of codes
  - when **data** and **knowledge** about the problem **increases**
  - when we get to use **new emerging algorithms**

The Novel Ideas include:

- Programming for decompositions of a global optimizations that are used in training and prediction
- An abstraction for unifying various formalisms for learning and reasoning which is an ongoing work
- A unified language for graph querying and structured learning

Conclusion

Thank you!

- Saul language **facilitates modeling** NLP applications that need
  - **Learning** and **reasoning** over Structures
  - **Relational Feature Extraction**
  - **Direct** use of **expert knowledge** beyond data instances
- Saul saves **Programming time**
  - in designing various configurations and experimentation
    - each configuration is expressed in a **few lines of declarative code.**
- Saul increases the **reusability** of codes
  - when **data** and **knowledge** about the problem **increases**
  - when we get to use **new emerging algorithms**

The Novel Ideas include:

- Programming for decompositions of a global optimizations that are used in training and prediction

I have two open PhD positions and one postdoc position, please contact me at pkordjam@tulane.edu, if you are interested!

https://github.com/IllinoisCogComp/saul

NLP Sensors

```
def textAnnotationToTree(ta: TextAnnotation): Tree[Constituent]

def textAnnotationToStringTree(ta: TextAnnotation): Tree[String]

def getPOS(x: Constituent): String

def getLemma(x: Constituent): String

def getSubtreeArguments(currentSubTrees: List[Tree[Constituent]]): List[Tree[Constituent]]

def xuPalmerCandidate(x: Constituent, y: Tree[String]): List[Relation]

def fexContextFeats(x: Constituent, featureExtractor: WordFeatureExtractor): String

…
```