Automatic Grammar Induction for Machine Translation

Presented by Benjamin Han (benhdj@cs.cmu.edu)

Outline

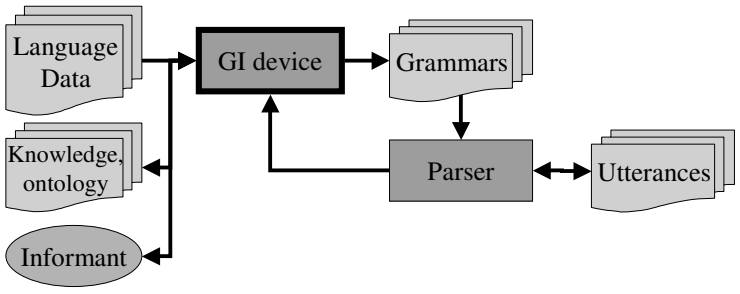
- Motivations
- GI+MT: Key Questions
- Principles and Practices of GI
- Incorporating GI into MT
- Conclusions

Roadmap

- **Motivations**
  - **What is GI**
  - **Why GI for MT?**
  - **GI and different MT paradigms**
  - **MT ≈ NLU**
- GI+MT: Key Questions
- Principles and Practices of GI
- Incorporating GI into MT
- Conclusions

Motivations

- What is Grammar Induction/Inference (GI)?
  - A process of acquiring the structural knowledge of languages themselves (vs. Mind Net, etc.).



Motivations

- Why GI for MT?
  - Grammars have played a central role in many MT systems.
- Transfer/interlingua-based MT
  - Manual development requires much time/skilled effort
  - Difficult to port across domains
  - Even partially automating the process is good

Motivations

- But how about corpus-based MT?
  - Using automatically generated word clustering and production rules reduces the amount of parallel text required while increasing translation accuracy in EBMT [Brown 2000].
  - Using structure-based models increases translation accuracy for structurally different language pairs (e.g. English vs. Japanese) and improves segmentation [Wang 1998].

Motivations

- MT can be adapted into Natural Language Understanding (NLU) problem
  - Automatic GI makes the system self-adapting/robust across domains/users.
  - Constraints from grammar help lexical acquisition [Woodward & Markman 1998].

Roadmap

- Motivations
- **GI+MT: Key Questions**
  - **What input do we need?**
  - **What output do we expect?**
- Principles and Practices of GI
- Incorporating GI into MT
- Conclusions

GI+MT: Key Questions

- What input do we need?
  - How much data do we have?
  - Do we have positive examples only? Or both positive/negative examples? Any additional constraints available on the input?
  - Are the input noisy?
  - Do we need language informants? By user interaction or by tagged corpora?
  - Do we need domain knowledge/ontology?

GI+MT: Key Questions

- What output do we expect?
  - What is the intrinsic complexity of the language to be learned? Regular? CFG? Or finer-grained?
  - What are the appropriate representations for the task? Rules? Rules with features? Automata? Networks?
  - Does the output need to be human-understandable? Do we consider the possibility of human/machine hybrid development?
  - Does the output need to be integrated with the existing knowledge?

Roadmap

- Motivations
- GI+MT: Key Questions
- **Principles and Practices of GI**
  - **Language learnability [Gold 1967]**
  - **Learning**
    - **Regular languages/finite state automata**
    - **Context-free grammars/push-down automata**
- Incorporating GI into MT
- Conclusions

GI: Principles & Practices

- Power of different formal languages
- Learnability of different formal languages
- Learning regular/CF languages
  - Symbolic methods (version space, regular positive/negative inferences, L*, etc.)
  - Numerical methods (connectionist methods)
  - Hybrid methods (statistical/evolutionary methods)

Power of Formal Languages

properly inclusive

Language	Grammar	Machine	Chomsky Hierarchy
Non-computable		??	
Recursively enumerable (RE)	Unrestricted	Turing machines	0
Recursive		Turing machines that always halt	
Context-sensitive	Context-sensitive grammar	Linear-bounded automata	1
Context-free	Context-free grammar	Non-deterministic pushdown automata	2
Regular	Regular expressions	Finite state automata	3

Learnability of Formal Languages

- First define rigorously what constitutes a particular learning scenario – "learnability model"
- Then determine the learnability of each formal language under different models

Learnability Models [Gold 1967]

- A *learnability model* consists of
  I. A **definition of learnability**: what do you mean by saying that a language is learned?
  II. A **method of information presentation:** how does an instructor teach the learner?
  III. A **naming relation** which assigns names to languages: what is the result of the learning?

I. Defining Learnability [Gold 1967]

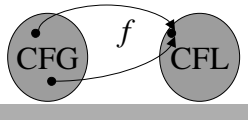
- Identification of a language *in the limit*
  - From some finite time t the guess of language identity stays the same and it is correct ($\exists t\ g_t = g_{t+1} = \ldots = g_\infty$ is correct)
  - Why do we care learnability "in the limit"?
    - If a language cannot be identified in the limit under a model, it's hopeless if we stick to the model
    - Even if a language can be identified in the limit, it might not be practical
- Other learnability definitions: Finite Identification and Fixed-time Identification.

II. Information Presentation [Gold 1967]

- Let i_t be the information presented to the learner at time t
  - Text: i_t is a string in the target language ω and every string in ω must be presented at least once.
  - Informant: each i_t is a string together with a binary signal indicating if the string is in ω.

III. Naming Relations [Gold 1967]

- Every language has it's 'name' in order to be uniquely identified – a name could be a grammar.
- A naming relation f is a mapping between names (N) and languages (Ω); $f: N \times \Omega$
- So the problem of language identification is to find a procedure by which given an unknown language ω the name n such that $f(n)=\omega$
- Two naming relations: *tester* and *generator*.



Results and Implications

Learnability model	Class of languages
Anomalous text →	RE
	Recursive
Informant (all 6 models) →	Primitive recursive
	Context-sensitive
	Context-free
	Regular
	Superfinite (all lngs of finite card. & at least one lng of infinite card.)
Text (5/6 models) →	Finite cardinality languages

Results and Implications

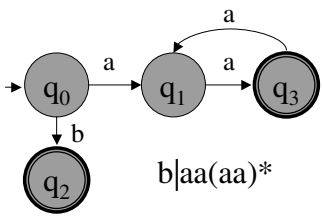
- With pure positive data the model can only learn language with finite cardinality!
- Possible ways out
  - Providing direct/indirect negative evidence (as provided by informants)
  - More constraints on the target classes of languages
  - Probabilistic framework

Roadmap

- Motivations
- GI+MT: Key Questions
- **Principles and Practices of GI**
  - **Language learnability** **[Gold 1967]**
  - **Learning**
    - **Regular languages/finite state automata**
    - **Context-free grammars/push-down automata**
- Incorporating GI into MT
- Conclusions

Learning Finite State Automata

- FSA recognizes *regular languages* – less powerful than CFL.
- Learning is a NP-hard problem [Gold 1978].
- Good for morphological analyses, even good for parsing (by cascading FSAs to simulate CFL).



Deterministic Finite Automata (DFA)

- Representation: Why not use regular expressions/non-deterministic finite automata/production rules?
  - Unique minimal DFA (# of states)
  - Poly-time algorithms for equivalence/subsumption tests and minimizations
  - Easy to understand

Approaches Overview

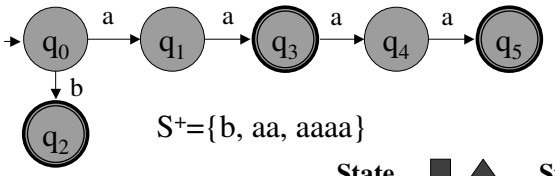
Info presentation \ Learning method	Symbolic	Hybrid	Numeric
Text			Learning Stochastic FSA [Carrasco & Oncina 1994] Learning Hidden Markov Models by Bayesian Model Merging [Carrasco & Oncina 1994]
Informant	Version space [Parekh & Honavar 1996] Regular positive and negative inference [Oncina & Garcia 1992] L* algorithm [Angluin 1987]	Genetic algorithms [Dupont et al 1994]	Recurrent Neural Networks (RNN) [Giles et al 1991, Siegelmann & Giles 1997] Learning RNN with Evolutionary Programming [Angeline et al 1994]

Outline of Symbolic Approaches

- From a set of positive examples S^+ construct the most specific DFA.
  - The most specific DFA consistent to S^+ is a *Prefix Tree Automaton* learned from S^+.
- Generalize/specialize more specific/general DFA by considering a set of negative examples S^-
  - The most general DFA is *universal DFA* consisting of only one accepting state.
  - Obtaining *quotient DFA* by state merging/splitting.

PTA and Quotient DFA

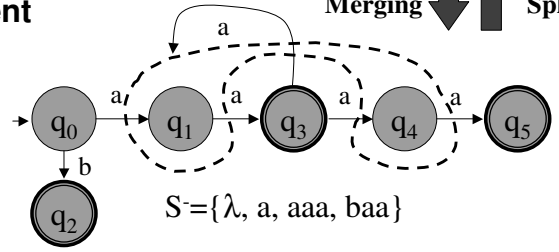
PTA



$S^+=\{b, aa, aaaa\}$

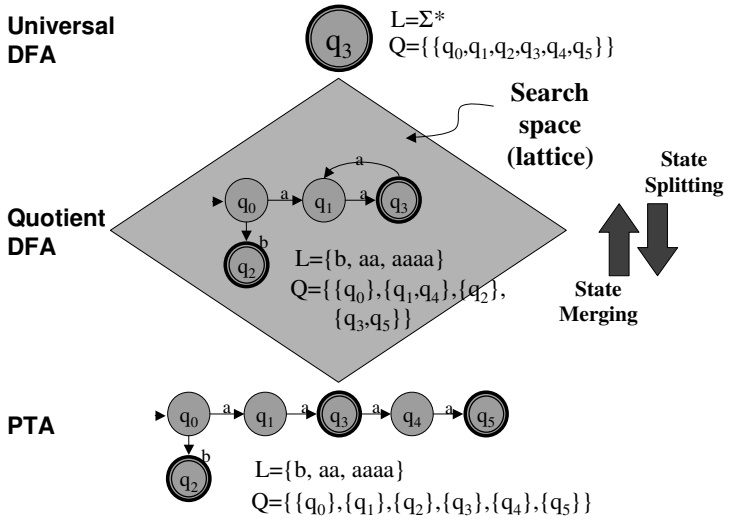
State Merging ⇓ ⇑ **State Splitting**

Quotient DFA



$S^-=\{\lambda, a, aaa, baa\}$

Learning FSA – a Search Problem



The Search Space is…

- .. a lattice with the partial order "more general/specific".
- … is guaranteed to contain the correct target DFA *M* if S^+ is a *structurally complete* set [Dupont et al 1994]
  - For each transition of *M* there exists at least one string in S^+ that covers the transition.
  - For each accepting state of *M* there exists at least one string in S^+ that ends at that state.

Version Space [Parekh & Honavar 1996]

- Information: structurally complete S^+ and informant.
- The algorithm keeps two sets of DFA: *S* (specific) and *G* (general).
- Initially *S*={PTA} and *G*={Universal DFA}
- Upon receiving a negative example the algorithm eliminates DFA in *S* that accepts it and state-splits DFA in *G* so that no DFA in *G* accepts the negative example.
- Convergence is guaranteed.
- Time: exponential in the size of PTA

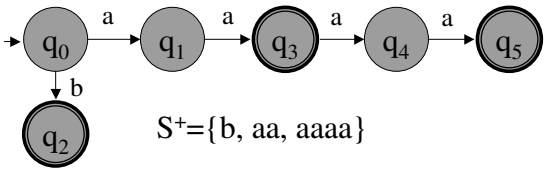
Regular Positive & Negative Inference [Oncina & Garcia 1992]

- Information: *Characteristic* set $S=S^+ \cup S^-$
  - S^+ is structurally complete.
  - S^- prevents any non-equivalent states in PTA from being merged.
- Initially the PTA is constructed from S^+.
- The algorithm attempts to merge states provided the resulting DFA does not accept any string in S^-.
- Convergence is guaranteed.
- Time: $O((|S^+|+|S^-|)\cdot|S^+|^2)$

Regular Positive & Negative Inference

[Oncina & Garcia 1992]

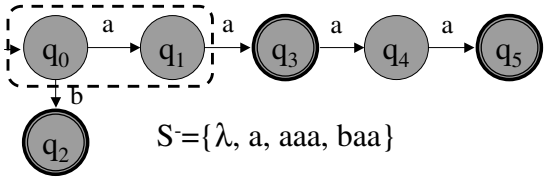
PTA



S^+={b, aa, aaaa}

Merge? NO!

Quotient DFA



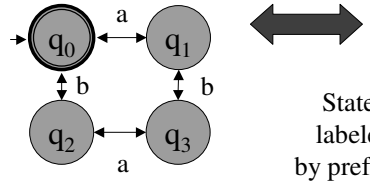
S^-={λ, a, aaa, baa}

L* Algorithm [Angluin 1987]

- Information: *minimally adequate teacher* (informant), which answers two types of questions:
  - *Membership* questions: Is string *s* in the target language?
  - *Conjecture* questions: Is the conjecture DFA the right one? If not give a **counterexample**.
- L* does *not* use a lattice, instead it constructs hypothesis DFA to accommodate the new examples.
- L* does not assume anything on the input.

L* Algorithm [Angluin 1987]

- The algorithm maintains an *observation table T*



L={$a^p b^q$ | *p* and *q* are even numbers}

States labeled by prefixes *s*

suffixes *e* to distinguish two states

	λ	a	b
λ	1	0	0
a	0	1	0
b	0	0	1
ab	0	0	0

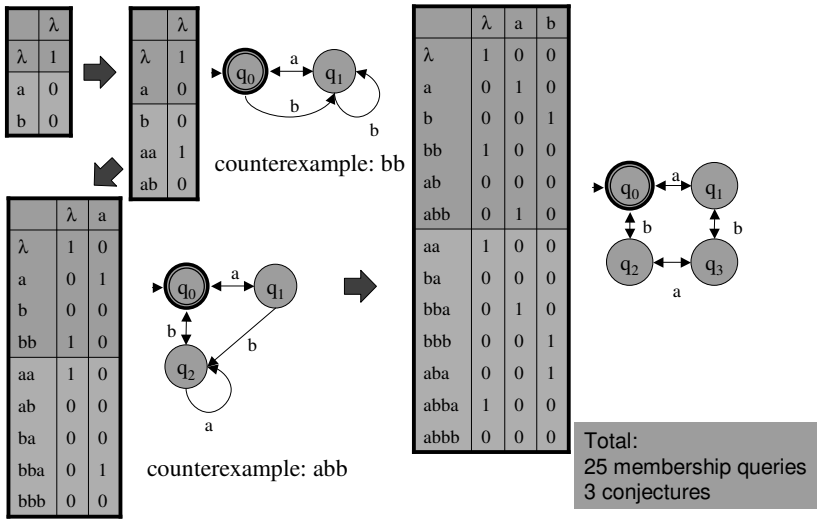
0/1 from membership tests

- But the true observation table is only indirectly available through the teacher.

L* Algorithm [Angluin 1987]

- Algorithm outline
  1. Initialize *T* to contain only one entry [λ, λ] (format: [*row*,*col*])
  2. For each table entry ask **membership question** of string *row*·*col*.
  3. For each state *s*, advance one character *a* to a new state *s*·*a*
     a. If any two states s_1 and s_2 are equivalent but not $s_1 \cdot a \cdot e$ and $s_2 \cdot a \cdot e$, add a new column $a \cdot e$.
     b. If $s \cdot a$ points to a new state add a new row $s \cdot a$.
  4. Otherwise construct DFA from *T* and ask **conjecture question**. If the answer is no for the counterexample *t* (a string) add all prefixes of *t* as new rows of *T*. Go back to Step 2.
  5. Otherwise done.

L* Algorithm [Angluin 1987]



L* Algorithm [Angluin 1987]

- Convergence is guaranteed.
- Time: Polynomial in # of states of the minimal target DFA, and the max length of the counterexamples.
- Interesting questions
  - Randomized oracle as the teacher
  - How to minimize the length of the counterexamples?

Genetic Algorithms [Dupont et al 1994]

- What is Genetic Algorithms (GA)
  - Maintain a population of solutions ('chromosomes')
  - Only well performers reproduce
  - Reproduction involves crossover to combine potentially good "building blocks"
  - Individual solutions can mutate at random
- Why GA?
  - Good for huge and ill-defined search space
  - Optimize upon baseline population – can't hurt

Genetic Algorithms [Dupont et al 1994]

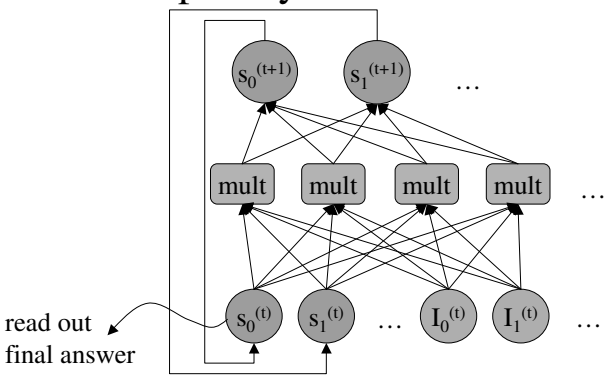
- Information: $S=S^{+} \cup S^{-}$
- Solution encoding: each quotient DFA is a solution
- Fitness metrics: DFA with fewer states and makes fewer errors gets higher scores
- Two GA operations
  - Structural mutation
  - Structural crossover

{{0,1,**3**,5},{2},{4}}
→ {{0,1,5},{2,**3**},{4}}

{{0},**{1,4}**,{2,3,5}} and
{{0},**{1,3}**,{2},{4},{5}}
→ {{0},**{1,3,4}**,{2,5}} and
{{0},**{1,3,4}**,{2},{5}}

Recurrent Neural Networks [Giles et al 1991]

- Information: $S=S^{+} \cup S^{-}$
- RNN implicitly learns a state machine



Recurrent Neural Networks [Giles et al 1991]

- Bound of the RNN size is easily computed [Siegelmann & Giles 1997]
- Gradient descent strategy is used to train the weights in order to minimize the squared error.
- Works have been done to extract FSA from a trained RNN.
- Combined with *Evolutionary Programming* to automatically decide the network topology (GNARL [Angeline et al 1994]).

Roadmap

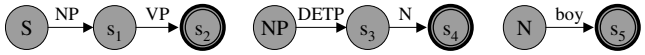
- Motivations
- GI+MT: Key Questions
- **Principles and Practices of GI**
  - **Language learnability [Gold 1967]**
  - **Learning**
    - **Regular languages/finite state automata**
    - **Context-free grammars/push-down automata**
- Incorporating GI into MT
- Conclusions

Learning Context-free Grammars

- You think learning Regular Languages is hard?
  - Given two CF grammars G_1 and G_2 there exists no algorithm to tell if $L(G_1) \subseteq L(G_2)$
  - There exists no algorithm to tell if $L(G_1) \cap L(G_2) = \varnothing$

Representations of CFG

- Chomsky Normal Form (CNF)
  - $A \rightarrow BC$
  - $A \rightarrow a$
- Recursive Transition Networks (RTN)



- Pushdown Automata (PDA)
  - FSA with a stack
  - CFG is equivalent to non-deterministic PDA (NPDA); deterministic PDA is weaker

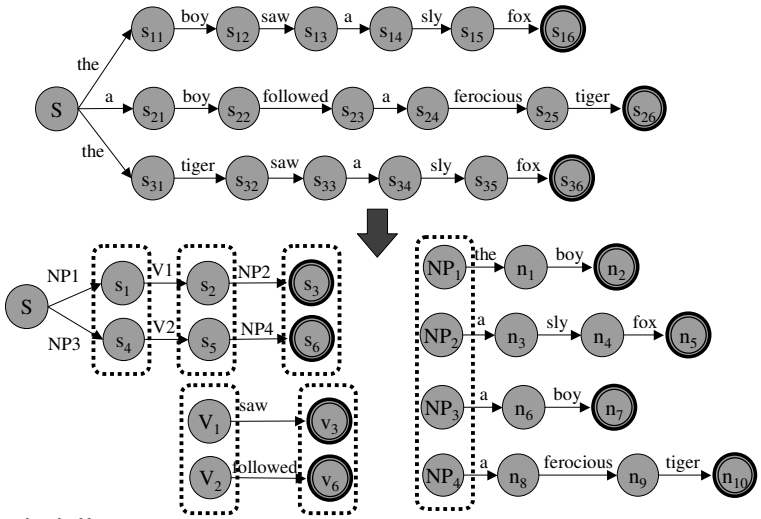
Approaches Outline

Info presentation \ Learning method	Symbolic	Hybrid	Numeric
Text			Learning Stochastic CFG [Stolcke & Omohundro 1994]
Informant	RTN Learning [Langley 1994] Version space [Vanlehn & Ball 1987] [Giordano 1994]	Genetic algorithms [Lankhorst 1995]	Learning PDA using RNN [Das et al 1992]

Learning RTN [Langley 1994]

- Information: $S=S^+ \cup S^-$
- Algorithm Outline
  - Create an initial network which trivially accepts all strings in S^+ (similar to the idea of PTA).
  - Identify recurrent patterns to create sub-nets.
  - Merge networks using S^- or additional knowledge sources.

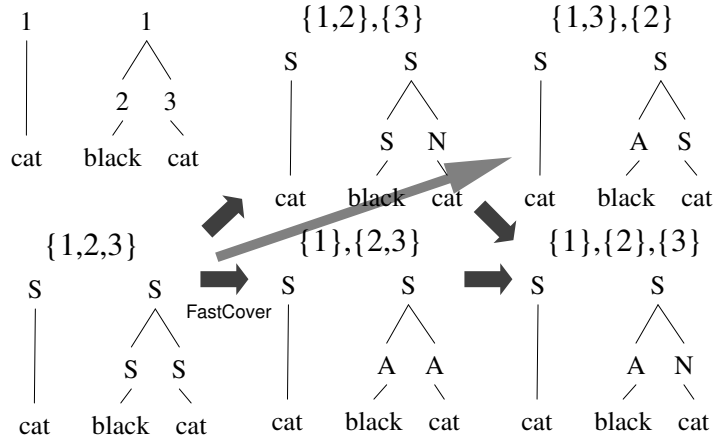
Learning RTN [Langley 1994]



Version Space [Vanlehn & Ball 1987]

- Information: $S=S^+ \cup S^-$, where S^+ contains unlabeled parse trees.
- The *derivational* version space only contains *reduced* grammars – whose proper subsets are not consistent with any positive example.
- The learning problem is reduced to finding the right partitions of the non-terminals.

Version Space [Vanlehn & Ball 1987]



Version Space [Vanlehn & Ball 1987]

- Algorithm outline
  - For each positive example add all possible partitions into the version space.
  - For each negative example prune the grammar which accepts it, and all the grammars which are covered by it.

Genetic Algorithms [Lankhorst 1995]

- Each chromosome encodes a fixed # of transitions of a NPDA.
- The fitness of each chromosome is determined by
  - Training accuracy
  - Correct prefix identification
  - Residual stack size

Roadmap

- Motivations
- GI+MT: Key Questions
- Principles and Practices of GI
- **Incorporating GI into MT**
  - **GI modules + different MT paradigms**
  - **Monolingually trained MT systems**
- Conclusions

GI Modules + Different MT Paradigms

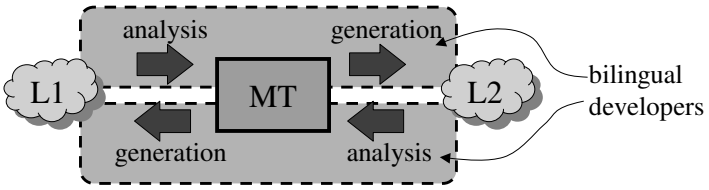
- Summary of GI approaches
  - All approaches except for probabilistic ones require both positive and negative examples.
  - In most of the situation $S^+ \cup S^-$ can be recast into an *informant* or *random-sampling* information source.
  - Most of the CFG learning algorithms require structural positive examples.
  - Most symbolic methods, with noise-free information input, guarantee convergence.
  - Extracting human-understandable knowledge from connectionist approaches is difficult.

GI Modules + Different MT Paradigms

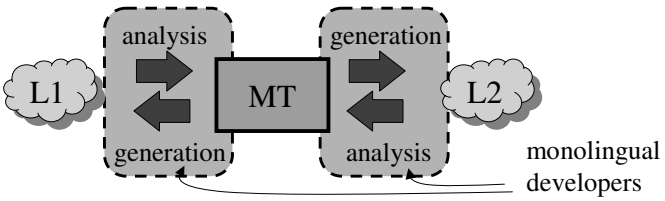
- For all MT paradigms: morphological analysis, segmentation, etc.
- For transfer and interlingua-based MT: learning transformational rules and analysis/generation grammars.
- For corpus-based MT: generalizing matching patterns.

Monolingually Trained MT Systems

- Transfer-based MT

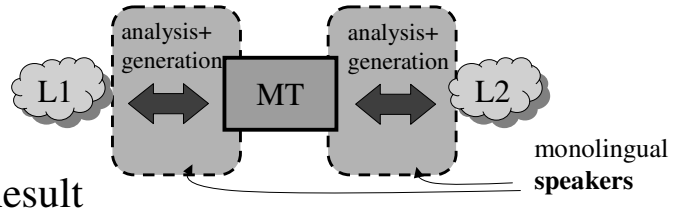


- Interlingua-based MT



Monolingually Trained MT Systems

- GI-enhanced Interlingua-based MT



- Result
  - An MT system which is able to learn from its interactions with different monolingual speakers!
  - Especially good for speech-to-speech MT.

Monolingually Trained MT Systems

- What additional components do we need to build such a system?
  - Bi-directional GI or Example-based Generator component
  - Dialog component

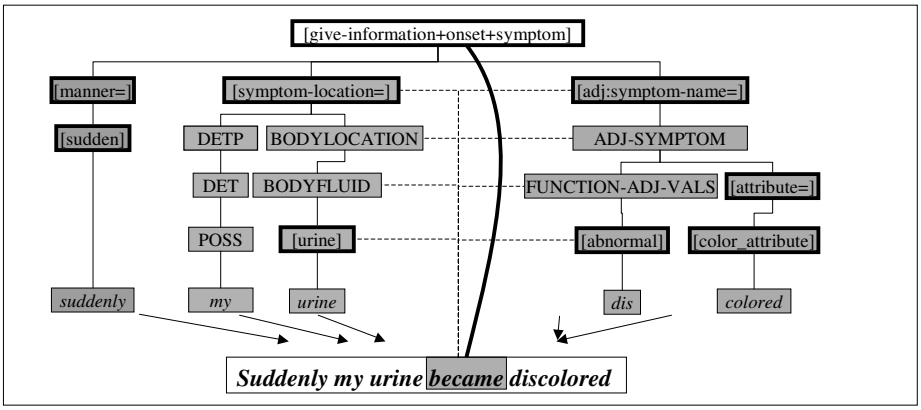
Roadmap

- Motivations
- GI+MT: Key Questions
- Principles and Practices of GI
- Incorporating GI into MT
- **Conclusions**

Conclusions

- GI can be used in many ways to help MT.
- GI is difficult, but interesting.
- GI intersects with theoretical computer science, machine learning, linguistics, cognitive science, etc.
- Much of the works is done on learning formal languages using syntactic guidance. How about taking advantage of domain knowledge?

Conclusions



Hypothesis of attachments:

- Without human intervention – additional knowledge sources required (ontology, statistical models, etc.).
- With human intervention – pose smart questions to the user (e.g. "Is the word *became* usually used to describe the onset of a symptom?) .

Feedback