Interpretation of natural language instructions

Translating sentences by using a grammar

Martin Agfjord

University of Gothenburg
Computer Science and Engineering

Outline

Introduction & problem description

- An alternative user interface

Introduction & problem description

- An alternative user interface
- Translation

Introduction & problem description

- An alternative user interface
- Translation
- Delimitation
  - Intranet of a software development company

Introduction & problem description

- An alternative user interface
- Translation
- Delimitation
    - Intranet of a software development company
    - Customers, People and Projects exists

Introduction & problem description

- An alternative user interface
- Translation
- Delimitation
  - Intranet of a software development company
  - Customers, People and Projects exists
  - Limited amount of instructions

Interface definition

Sufficient for novice users

`people who know Java`

Interface definition

Sufficient for novice users

`people who know Java`

Sufficient for expert users

`people java`

Solution

Solution

- Precise translation

Solution

- Precise translation
- Need mapping from natural language to query language

Solution

- Precise translation
- Need mapping from natural language to query language
  - Use a grammar

Translation with a grammar

- Structured rules for strings

Translation with a grammar

- Structured rules for strings
- Example: Is/Are rule

Translation with a grammar

- Structured rules for strings
- Example: Is/Are rule
  - The students are here
  - The student is here

Translation with a grammar

- Structured rules for strings
- Example: Is/Are rule
    - The students are here
    - The student is here
- Swedish
    - Studenterna är här
    - Studenten är här

Translation with a grammar

- Structured rules for strings
- Example: Is/Are rule
  - The students are here
  - The student is here
- Swedish
  - Studenterna är här
  - Studenten är här

Translation with a grammar

- Structured rules for strings
- Example: Is/Are rule
  - The students are here
  - The student is here
- Swedish
  - Studenterna är här
  - Studenten är här
- Use the rules to extract the semantics of a sentence

Translation with a grammar

- Structured rules for strings
- Example: Is/Are rule
  - The students are here
  - The student is here
- Swedish
  - Studenterna är här
  - Studenten är här
- Use the rules to extract the semantics of a sentence
  - Studenterna är här
    `===> ((Student Def Plural) Is Here)`

Translation with a grammar

- Structured rules for strings
- Example: Is/Are rule
  - The students are here
  - The student is here
- Swedish
  - Studenterna är här
  - Studenten är här
- Use the rules to extract the semantics of a sentence
  - Studenterna är här
    `===> ((Student Def Plural) Is Here)`
- Use the rules again to produce a sentence

Translation with a grammar

- Structured rules for strings
- Example: Is/Are rule
  - The students are here
  - The student is here
- Swedish
  - Studenterna är här
  - Studenten är här
- Use the rules to extract the semantics of a sentence
  - Studenterna är här
    `===> ((Student Def Plural) Is Here)`
- Use the rules again to produce a sentence
  - `((Student Def Plural) Is Here)`
    ===> The students are here

Translation with a grammar

- Structured rules for strings
- Example: Is/Are rule
  - The students are here
  - The student is here
- Swedish
  - Studenterna är här
  - Studenten är här
- Use the rules to extract the semantics of a sentence
  - Studenterna är här
    `===> ((Student Def Plural) Is Here)`
- Use the rules again to produce a sentence
  - `((Student Def Plural) Is Here)`
    `===>` The students are here

How can we build a grammar to translate sentences?

We will use Grammatical Framework (GF)

Introducing Grammatical Framework (GF)

- Open source development platform for natural languages

Introducing Grammatical Framework (GF)

- Open source development platform for natural languages
    - Functional programming language

Introducing Grammatical Framework (GF)

- Open source development platform for natural languages
  - Functional programming language
  - Designed for creating natural language grammars

Introducing Grammatical Framework (GF)

- Open source development platform for natural languages
  - Functional programming language
  - Designed for creating natural language grammars
- Separates abstract and concrete syntax

Introducing Grammatical Framework (GF)

- Open source development platform for natural languages
    - Functional programming language
    - Designed for creating natural language grammars
- Separates abstract and concrete syntax
    - Abstract syntax represents the semantics of a sentence

Introducing Grammatical Framework (GF)

- Open source development platform for natural languages
  - Functional programming language
  - Designed for creating natural language grammars
- Separates abstract and concrete syntax
  - Abstract syntax represents the semantics of a sentence

```
((Student Def Plural) Is Here)
```

Introducing Grammatical Framework (GF)

- Open source development platform for natural languages
  - Functional programming language
  - Designed for creating natural language grammars
- Separates abstract and concrete syntax
  - Abstract syntax represents the semantics of a sentence

    `((Student Def Plural) Is Here)`
  - Concrete syntax represents the semantics as a string

Introducing Grammatical Framework (GF)

- Open source development platform for natural languages
  - Functional programming language
  - Designed for creating natural language grammars
- Separates abstract and concrete syntax
  - Abstract syntax represents the semantics of a sentence

    `((Student Def Plural) Is Here)`
  - Concrete syntax represents the semantics as a string

    The students are here

Introducing Grammatical Framework (GF)

- Open source development platform for natural languages
  - Functional programming language
  - Designed for creating natural language grammars
- Separates abstract and concrete syntax
  - Abstract syntax represents the semantics of a sentence

    `((Student Def Plural) Is Here)`
  - Concrete syntax represents the semantics as a string

    The students are here

Same technique used by programming languages

- Programmer writes source code in concrete syntax

Introducing Grammatical Framework (GF)

- Open source development platform for natural languages
  - Functional programming language
  - Designed for creating natural language grammars
- Separates abstract and concrete syntax
  - Abstract syntax represents the semantics of a sentence

    `((Student Def Plural) Is Here)`
  - Concrete syntax represents the semantics as a string

    The students are here

Same technique used by programming languages

- Programmer writes source code in concrete syntax
- Compiler translates concrete syntax to abstract syntax

Introducing Grammatical Framework (GF)

- Open source development platform for natural languages
  - Functional programming language
  - Designed for creating natural language grammars
- Separates abstract and concrete syntax
  - Abstract syntax represents the semantics of a sentence

    `((Student Def Plural) Is Here)`
  - Concrete syntax represents the semantics as a string

    The students are here

Same technique used by programming languages

- Programmer writes source code in concrete syntax
- Compiler translates concrete syntax to abstract syntax
- The rest of the compiler manipulates the abstract syntax

A simple example

Abstract syntax

`Instruction People (Know Java)`

```
    Instruction
   /           \
People        Know
                |
              Java
```

A simple example

Abstract syntax

```
Instruction People (Know Java)
```

```
    Instruction
   /           \
People         Know
                |
               Java
```

Concrete syntaxes

```
people who know Java                          -- English
personer som kan Java                         -- Swedish
q=object_type : Person AND expertise : Java   -- Solr
```

GF implementation: Abstract syntax

GF implementation: Abstract syntax

```
abstract Instrucs = {

  cat
    Instruction ;
    Subject ;
    Relation ;
    Object ;
```

GF implementation: Abstract syntax

```
abstract Instrucs = {

  cat
    Instruction ;
    Subject ;
    Relation ;
    Object ;

 fun
    MkInstruction : Subject -> Relation -> Instruction ;
    People : Subject ;
    Know : Object -> Relation ;
    Java : Object
}
```

GF implementation: Abstract syntax

```
abstract Instrucs = {

  cat
    Instruction ;                       MkInstruction
    Subject ;                          /             \
    Relation ;                      People           Know
    Object ;                                          |
                                                     Java
 fun
    MkInstruction : Subject -> Relation -> Instruction ;
    People : Subject ;
    Know : Object -> Relation ;
    Java : Object
}
```

GF implementation: English concrete syntax

GF implementation: English concrete syntax

```
concrete InstrucsEng of Instrucs = {

  lincat
    Instruction = Str ;
    Subject = Str ;
    Relation = Str ;
    Object = Str ;
```

GF implementation: English concrete syntax

```
concrete InstrucsEng of Instrucs = {

  lincat
    Instruction = Str ;
    Subject = Str ;
    Relation = Str ;
    Object = Str ;

  lin
    MkInstruction subject relation =
                   subject ++ "who" ++ relation ;
    People = "people" ;
    Know object = "know" ++ object ;
    Java = "Java" ;
}
```

GF implementation: English concrete syntax

```
concrete InstrucsEng of Instrucs = {

  lincat
    Instruction = Str ;          MkInstruction
    Subject = Str ;             /              \
    Relation = Str ;       People              Know
    Object = Str ;                               |
                                               Java
  lin
    MkInstruction subject relation =
                   subject ++ "who" ++ relation ;
    People = "people" ;
    Know object = "know" ++ object ;
    Java = "Java" ;
}
```

GF implementation: English concrete syntax

```
concrete InstrucsEng of Instrucs = {

  lincat
    Instruction = Str ;          MkInstruction
    Subject = Str ;            /              \
    Relation = Str ;       People             Know
    Object = Str ;                              |
                                              Java
  lin
    MkInstruction "people" relation =
                   "people" ++ "who" ++ relation ;
    People = "people" ;
    Know object = "know" ++ object ;
    Java = "Java" ;
}
```

GF implementation: English concrete syntax

```
concrete InstrucsEng of Instrucs = {

  lincat
    Instruction = Str ;
    Subject = Str ;
    Relation = Str ;
    Object = Str ;

  lin
    MkInstruction "people" relation =
                     "people who" ++ relation ;
    People = "people" ;
    Know object = "know" ++ object ;
    Java = "Java" ;
}
```

```
          MkInstruction
         /             \
    People             Know
                         |
                       Java
```

GF implementation: English concrete syntax

```
concrete InstrucsEng of Instrucs = {

  lincat
    Instruction = Str ;
    Subject = Str ;
    Relation = Str ;
    Object = Str ;

  lin
    MkInstruction "people" relation =
                    "people who" ++ relation ;
    People = "people" ;
    Know "Java" = "know" ++ "Java" ;
    Java = "Java" ;
}
```

```
        MkInstruction
       /             \
  People             Know
                       |
                     Java
```

GF implementation: English concrete syntax

```
concrete InstrucsEng of Instrucs = {

  lincat
    Instruction = Str ;          MkInstruction
    Subject = Str ;             /             \
    Relation = Str ;       People            Know
    Object = Str ;                              |
                                              Java
  lin
    MkInstruction "people" relation =
                  "people who" ++ relation ;
    People = "people" ;
    Know "Java" = "know Java" ;
    Java = "Java" ;
}
```

GF implementation: English concrete syntax

```
concrete InstrucsEng of Instrucs = {

  lincat
    Instruction = Str ;
    Subject = Str ;
    Relation = Str ;
    Object = Str ;

  lin
    MkInstruction "people" "know Java" =
                   "people who" ++ "know Java" ;
    People = "people" ;
    Know "Java" = "know Java" ;
    Java = "Java" ;
}
```

```
        MkInstruction
       /             \
  People             Know
                       |
                     Java
```

GF implementation: English concrete syntax

```
concrete InstrucsEng of Instrucs = {

  lincat
    Instruction = Str ;          MkInstruction
    Subject = Str ;             /             \
    Relation = Str ;       People             Know
    Object = Str ;                              |
                                              Java
  lin
    MkInstruction "people" "know Java" =
                 "people who know Java" ;
    People = "people" ;
    Know "Java" = "know Java" ;
    Java = "Java" ;
}
```

GF implementation: Solr concrete syntax

GF implementation: Solr concrete syntax

```
concrete InstrucsEng of Instrucs = {

  lincat
    Instruction = Str ;
    Subject = Str ;
    Relation = Str ;
    Object = Str ;
```

GF implementation: Solr concrete syntax

```
concrete InstrucsEng of Instrucs = {

  lincat
    Instruction = Str ;
    Subject = Str ;
    Relation = Str ;
    Object = Str ;

  lin
  MkInstruction subject relation =
      "q=" ++ subject ++ "AND" ++ relation ;
  People = "object_type : Person" ;
  Know object = "expertise :" ++ object ;
  Java = "Java" ;
}
```

GF implementation: Solr concrete syntax

```
concrete InstrucsEng of Instrucs = {

  lincat
    Instruction = Str ;
    Subject = Str ;
    Relation = Str ;
    Object = Str ;

  lin
  MkInstruction subject relation =
      "q=" ++ subject ++ "AND" ++ relation ;
  People = "object_type : Person" ;
  Know object = "expertise :" ++ object ;
  Java = "Java" ;
}
```

```
       MkInstruction
      /             \
People               Know
                       |
                     Java
```

GF implementation: Solr concrete syntax

```
concrete InstrucsEng of Instrucs = {

  lincat
    Instruction = Str ;
    Subject = Str ;
    Relation = Str ;
    Object = Str ;

  lin
  MkInstruction "object_type : Person" relation =
      "q=" ++ "object_type : Person" ++ "AND" ++ relation ;
  People = "object_type : Person" ;
  Know object = "expertise :" ++ object ;
  Java = "Java" ;
}
```

```
         MkInstruction
        /             \
   People             Know
                        |
                      Java
```

GF implementation: Solr concrete syntax

```
concrete InstrucsEng of Instrucs = {

  lincat
    Instruction = Str ;          MkInstruction
    Subject = Str ;             /             \
    Relation = Str ;        People             Know
    Object = Str ;                               |
                                               Java
  lin
  MkInstruction "object_type : Person" relation =
      "q= object_type : Person" ++ "AND" ++ relation ;
  People = "object_type : Person" ;
  Know object = "expertise :" ++ object ;
  Java = "Java" ;
}
```

GF implementation: Solr concrete syntax

```
concrete InstrucsEng of Instrucs = {

  lincat
    Instruction = Str ;          MkInstruction
    Subject = Str ;             /             \
    Relation = Str ;        People            Know
    Object = Str ;                              |
                                              Java
  lin
  MkInstruction "object_type : Person" relation =
      "q= object_type : Person AND" ++ relation ;
  People = "object_type : Person" ;
  Know object = "expertise :" ++ object ;
  Java = "Java" ;
}
```

GF implementation: Solr concrete syntax

```
concrete InstrucsEng of Instrucs = {

  lincat
    Instruction = Str ;          MkInstruction
    Subject = Str ;             /             \
    Relation = Str ;       People             Know
    Object = Str ;                              |
                                              Java
  lin
  MkInstruction "object_type : Person" relation =
      "q= object_type : Person AND" ++ relation ;
  People = "object_type : Person" ;
  Know "Java" = "expertise :" ++ "Java" ;
  Java = "Java" ;
}
```

GF implementation: Solr concrete syntax

```
concrete InstrucsEng of Instrucs = {

  lincat
    Instruction = Str ;          MkInstruction
    Subject = Str ;             /             \
    Relation = Str ;       People              Know
    Object = Str ;                               |
                                                Java
  lin
  MkInstruction "object_type : Person" relation =
      "q= object_type : Person AND" ++ relation ;
  People = "object_type : Person" ;
  Know "Java" = "expertise : Java" ;
  Java = "Java" ;
}
```

GF implementation: Solr concrete syntax

```
concrete InstrucsEng of Instrucs = {

  lincat
    Instruction = Str ;          MkInstruction
    Subject = Str ;             /             \
    Relation = Str ;       People             Know
    Object = Str ;                              |
                                              Java
  lin
  MkInstruction "object_type : Person" "expertise : Java"=
      "q= object_type : Person AND" ++ "expertise : Java" ;
  People = "object_type : Person" ;
  Know "Java" = "expertise : Java" ;
  Java = "Java" ;
}
```

GF implementation: Solr concrete syntax

```
concrete InstrucsEng of Instrucs = {

  lincat
    Instruction = Str ;          MkInstruction
    Subject = Str ;             /              \
    Relation = Str ;        People              Know
    Object = Str ;                                |
                                                Java
  lin
  MkInstruction "object_type : Person" "expertise : Java"=
      "q= object_type : Person AND expertise : Java" ;
  People = "object_type : Person" ;
  Know "Java" = "expertise : Java" ;
  Java = "Java" ;
}
```

GF implementation: Translation

GF + Abstract syntax + Concrete syntax =

GF implementation: Translation

GF + Abstract syntax + Concrete syntax =

Parser

```
> parse -lang=InstrucsEng "people who know Java"
MkInstruction People (Know Java)
```

GF implementation: Translation

GF + Abstract syntax + Concrete syntax =

Parser

```
> parse -lang=InstrucsEng "people who know Java"
MkInstruction People (Know Java)
```

Linearizer

```
> linearize -lang=InstrucsSolr
>                  MkInstruction People (Know Java)
"q= object_type : Person AND expertise : Java"
```

GF implementation: Translation

GF + Abstract syntax + Concrete syntax =

Parser

```
> parse -lang=InstrucsEng "people who know Java"
MkInstruction People (Know Java)
```

Linearizer

```
> linearize -lang=InstrucsSolr
>                  MkInstruction People (Know Java)
"q= object_type : Person AND expertise : Java"
```

Generator

```
> generate_trees
MkInstruction People (Know Java)
```

GF implementation: Resource Grammar Library

GF implementation: Resource Grammar Library

- Contains linguistic descriptions for natural languages

GF implementation: Resource Grammar Library

- Contains linguistic descriptions for natural languages
  - Types for nouns, verbs, adjectives, noun phrases, verb phrases, relative sentences, phrases...

GF implementation: Resource Grammar Library

- Contains linguistic descriptions for natural languages
  - Types for nouns, verbs, adjectives, noun phrases, verb phrases, relative sentences, phrases...
- Developer does not need to know linguistics

GF implementation: Resource Grammar Library

- Contains linguistic descriptions for natural languages
  - Types for nouns, verbs, adjectives, noun phrases, verb phrases, relative sentences, phrases...
- Developer does not need to know linguistics
  - Example: 'Yesterday I ate an apple'

GF implementation: Resource Grammar Library

- Contains linguistic descriptions for natural languages
  - Types for nouns, verbs, adjectives, noun phrases, verb phrases, relative sentences, phrases...
- Developer does not need to know linguistics
  - Example: 'Yesterday I ate an apple'
  - Direct translation to Swedish: 'Igår jag åt ett äpple'

GF implementation: Resource Grammar Library

- Contains linguistic descriptions for natural languages
    - Types for nouns, verbs, adjectives, noun phrases, verb phrases, relative sentences, phrases...
- Developer does not need to know linguistics
    - Example: 'Yesterday I ate an apple'
    - Direct translation to Swedish: 'Igår jag åt ett äpple'
    - Correct translation to Swedish: 'Igår åt jag ett äpple'

GF implementation: Resource Grammar Library

- Contains linguistic descriptions for natural languages
  - Types for nouns, verbs, adjectives, noun phrases, verb phrases, relative sentences, phrases...
- Developer does not need to know linguistics
  - Example: 'Yesterday I ate an apple'
  - Direct translation to Swedish: 'Igår jag åt ett äpple'
  - Correct translation to Swedish: 'Igår åt jag ett äpple'
- Only need to know the *domain*

Resource Grammar Library: English concrete syntax

Resource Grammar Library: English concrete syntax

```
concrete InstrucsEng of Instrucs =
  lincat
    Instruction = Utt ;
    Subject = N ;
    Relation = RS ;
    Object = NP ;
```

Resource Grammar Library: English concrete syntax

```
concrete InstrucsEng of Instrucs =
  lincat
    Instruction = Utt ;
    Subject = N ;
    Relation = RS ;
    Object = NP ;

  lin
   MkInstruction subject relation =
           mkUtt (mkNP aPl_Det (mkCN subject relation)) ;
   People = mkN "person" "people" ;
   Know object = mkRS' (mkVP (mkV2 (mkV "know")) object) ;
   Java = mkNP (mkPN "Java") ;
  oper
     mkRS' : VP -> RS = \vp -> mkRS (mkRCl which_RP vp) ;
```

Resource Grammar Library: Swedish concrete syntax

```
concrete InstrucsEng of Instrucs =
  lincat
    Instruction = Utt ;
    Subject = N ;
    Relation = RS ;
    Object = NP ;

  lin
   MkInstruction subject relation =
           mkUtt (mkNP aPl_Det (mkCN subject relation)) ;
   People = mkN "person" "personer" ;
   Know object = mkRS' (mkVP (mkV2 (mkV "kan") object)) ;
   Java = mkNP (mkPN "Java") ;
  oper
     mkRS' : VP -> RS = \vp -> mkRS (mkRCl which_RP vp) ;
```

Extending the grammar

Extending the grammar: All programming languages

- Extend the grammar to support more programming languages

Extending the grammar: All programming languages

- Extend the grammar to support more programming languages
- Arbitrary names instead of hard coded functions

```
fun
  Java : Object ;
lin
  Java = "Java" ;
```

Extending the grammar: All programming languages

- Extend the grammar to support more programming languages
- Arbitrary names instead of hard coded functions

```
fun
  Java : Object ;
lin
  Java = "Java" ;

===>

fun
  MkObject : Symb -> Object ;
lin
  MkObject symb = symb.s ;
```

Extending the grammar: More instructions

- Extend grammar to support more instructions:

Extending the grammar: More instructions

- Extend grammar to support more instructions:

```
people who know Java
people who work in London
people who work with Unicef
customers who use Solr
projects who use Solr
```

Extending the grammar: More instructions

- Extend grammar to support more instructions:

  `people who know Java`

  `people who work in London`

  `people who work with Unicef`

  `customers who use Solr`

  `projects who use Solr`

Extending the grammar: Boolean operators

Extending the grammar: Boolean operators

- Extend grammar to support boolean operators
- `people who know Java or Python`

Extending the grammar: Boolean operators

- Extend grammar to support boolean operators
- `people who know Java or Python`
- `people who know Haskell and work in London`

Extending the grammar: Boolean operators

- Extend grammar to support boolean operators
- `people who know Java or Python`
- `people who know Haskell and work in London`

Suggestion Engine

- Narrow application grammar requires precise input

Suggestion Engine

- Narrow application grammar requires precise input
- Need to help user to find correct instructions

Suggestion Engine

- Narrow application grammar requires precise input
- Need to help user to find correct instructions
- Use suggestions based on partial input

Suggestion Engine

- Narrow application grammar requires precise input
- Need to help user to find correct instructions
- Use suggestions based on partial input
- Extract possible instructions into Solr

Suggestion Engine

- Narrow application grammar requires precise input
- Need to help user to find correct instructions
- Use suggestions based on partial input
- Extract possible instructions into Solr

Problem with arbitrary names

Suggestion Engine

- Narrow application grammar requires precise input
- Need to help user to find correct instructions
- Use suggestions based on partial input
- Extract possible instructions into Solr

Problem with arbitrary names

```
> generate_trees
InstrucExternal Customer (UseExt (MkObject (MkSymb "Foo")))
InstrucInternal People (Know (MkObject (MkSymb "Foo")))
InstrucInternal People (WorkIn (MkObject (MkSymb "Foo")))
InstrucInternal People (WorkWith (MkObject (MkSymb "Foo")))
InstrucResource Project (UseRes (MkObject (MkSymb "Foo")))
```

Suggestion Engine

- Narrow application grammar requires precise input
- Need to help user to find correct instructions
- Use suggestions based on partial input
- Extract possible instructions into Solr

Problem with arbitrary names

```
> generate_trees | linearize -lang=InstrucsEng
"customers who use Foo"
"people who know Foo"
"people who work in Foo"
"people who work with Foo"
"projects which use Foo"
```

Suggestion Engine

- Resolved by introducing name types

```
MkSkill : Symb -> Skill ;
MkOrganization : Symb -> Organization ;
MkModule: Symb -> Module ;
MkLocation : Symb -> Location
```

Suggestion Engine

- Resolved by introducing name types

```
MkSkill : Symb -> Skill ;
MkOrganization : Symb -> Organization ;
MkModule: Symb -> Module ;
MkLocation : Symb -> Location
```

```
> generate_trees
InstrucExternal Customer (UseExt (MkModule (MkSymb "Foo")))
InstrucInternal People (Know (MkSkill (MkSymb "Foo")))
InstrucInternal People (WorkIn (MkLocation (MkSymb "Foo")))
InstrucInternal People (WorkWith (MkOrganization (MkSymb "Foo")))
InstrucResource Project (UseRes (MkModule (MkSymb "Foo")))
```

Suggestion Engine

- Resolved by introducing name types

```
MkSkill : Symb -> Skill ;
MkOrganization : Symb -> Organization ;
MkModule: Symb -> Module ;
MkLocation : Symb -> Location
```

```
> generate_trees (post processed)
InstrucExternal Customer (UseExt (MkModule (MkSymb "Module0")))
InstrucInternal People (Know (MkSkill (MkSymb "Skill0")))
InstrucInternal People (WorkIn (MkLocation (MkSymb "Location0")))
InstrucInternal People (WorkWith (MkOrganization (MkSymb "Organi..)))
InstrucResource Project (UseRes (MkModule (MkSymb "Module0")))
```

Suggestion Engine

- Resolved by introducing name types

```
MkSkill : Symb -> Skill ;
MkOrganization : Symb -> Organization ;
MkModule: Symb -> Module ;
MkLocation : Symb -> Location
```

```
> generate_trees | linearize -lang=InstrucsEng
"customers who use Module0"
"people who know Skill0"
"people who work in Location0"
"people who work with Organization0"
"projects which use Module0"
```

Suggestion Engine: Pseudocode of algorithm

Suggestion Engine: Pseudocode of algorithm

```
suggestions(sentence) {
    // sentence = "anyone that ever knew java"
    // names[] = {Java}
    names[] = extractNames(sentence);
```

Suggestion Engine: Pseudocode of algorithm

```
suggestions(sentence) {
    // sentence = "anyone that ever knew java"
    // names[] = {Java}
    names[] = extractNames(sentence);

    // "anyone that ever knew java" ===>
    // "anyone that ever knew Skill0"
    sentence = replaceNamesWithTypes(sentence, names);
```

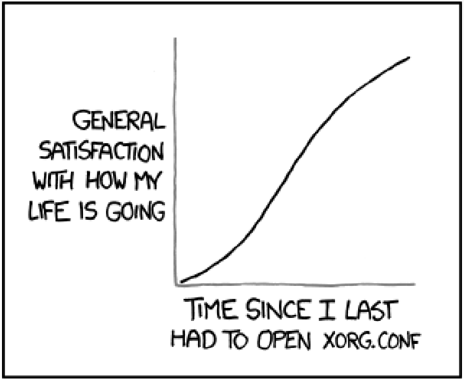
Suggestion Engine: Pseudocode of algorithm

```
suggestions(sentence) {
    // sentence = "anyone that ever knew java"
    // names[] = {Java}
    names[] = extractNames(sentence);

    // "anyone that ever knew java" ===>
    // "anyone that ever knew Skill0"
    sentence = replaceNamesWithTypes(sentence, names);

    // suggestions = { "people who know Skill0", ... }
    suggestions[] = findSentences(sentence);
```

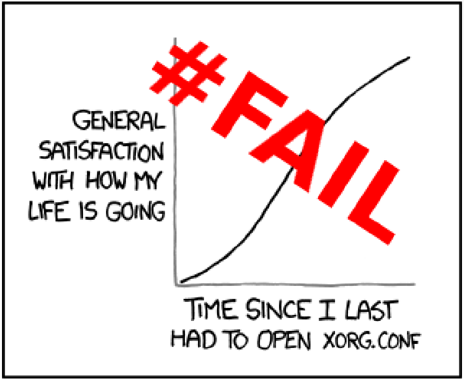
Suggestion Engine: Pseudocode of algorithm

```
suggestions(sentence) {
    // sentence = "anyone that ever knew java"
    // names[] = {Java}
    names[] = extractNames(sentence);

    // "anyone that ever knew java" ===>
    // "anyone that ever knew Skill0"
    sentence = replaceNamesWithTypes(sentence, names);

    // suggestions = { "people who know Skill0", ... }
    suggestions[] = findSentences(sentence);

    for each suggestion in suggestions {
    // "people who know Skill0" ===>
    // "people who know Java"
      suggestion = restoreNames(names, suggestion);
    }
    return suggestions;
}
```

Results


GENERAL SATISFACTION WITH HOW MY LIFE IS GOING
TIME SINCE I LAST HAD TO OPEN XORG.CONF

Results


GENERAL SATISFACTION WITH HOW MY LIFE IS GOING
#FAIL
TIME SINCE I LAST HAD TO OPEN XORG.CONF

Conclusion

Conclusion

- Application sufficient for novice and expert users

Conclusion

- Application sufficient for novice and expert users
- Resource Grammar Library not worth the effort

Conclusion

- Application sufficient for novice and expert users
- Resource Grammar Library not worth the effort
  - Not possible to express a few sentences

Conclusion

- Application sufficient for novice and expert users
- Resource Grammar Library not worth the effort
  - Not possible to express a few sentences
  - For example: `people which knows Java`

Conclusion

- Application sufficient for novice and expert users
- Resource Grammar Library not worth the effort
  - Not possible to express a few sentences
  - For example: `people which knows Java`
  - `people who know Java and work in London`

Conclusion

- Application sufficient for novice and expert users
- Resource Grammar Library not worth the effort
    - Not possible to express a few sentences
    - For example: `people which knows Java`
    - `people who know Java and work in London`
    - Compare: `people who know Java and who work in London`

Conclusion

- Application sufficient for novice and expert users
- Resource Grammar Library not worth the effort
  - Not possible to express a few sentences
  - For example: `people which knows Java`
  - `people who know Java and work in London`
  - Compare: `people who know Java and who work in London`
  - Problem with constant `which_RP`

Conclusion

- Application sufficient for novice and expert users
- Resource Grammar Library not worth the effort
  - Not possible to express a few sentences
  - For example: `people which knows Java`
  - `people who know Java and work in London`
  - Compare: `people who know Java and who work in London`
  - Problem with constant `which_RP`
  - Linearizes `projects who use Solr`

Conclusion

- Application sufficient for novice and expert users
- Resource Grammar Library not worth the effort
  - Not possible to express a few sentences
  - For example: `people which knows Java`
  - `people who know Java and work in London`
  - Compare: `people who know Java and who work in London`
  - Problem with constant `which_RP`
  - Linearizes `projects who use Solr`

Future Work

Future Work

- Improvments of suggestions

Future Work

- Improvments of suggestions
  - No suggestions based on empty string

Future Work

- Improvments of suggestions
  - No suggestions based on empty string
  - Add heuristic to auto completion

Future Work

- Improvments of suggestions
  - No suggestions based on empty string
  - Add heuristic to auto completion
  - If invalid instruction, choose most similar suggestion

Future Work

- Improvments of suggestions
  - No suggestions based on empty string
  - Add heuristic to auto completion
  - If invalid instruction, choose most similar suggestion
- Instructions in speech

Future Work

- Improvments of suggestions
  - No suggestions based on empty string
  - Add heuristic to auto completion
  - If invalid instruction, choose most similar suggestion
- Instructions in speech
- Proper handling of ambiguous instructions

Future Work

- Improvments of suggestions
  - No suggestions based on empty string
  - Add heuristic to auto completion
  - If invalid instruction, choose most similar suggestion
- Instructions in speech
- Proper handling of ambiguous instructions
- Use application in other context