# A Type Theoretic Approach to Structural Resolution

Peng Fu, Ekaterina Komendantskaya

University of Dundee
School of Computing

# Logic programming and Proof

- `k1 : Connect(x, y), Connect(y, z) => Connect(x, z)`
  `k2 : => Connect(N1, N2)`
  `k3 : => Connect(N2, N3)`
- Are there any proof of `Connect(x, N3)` for some `x`?
  Answer 1: `(k1 k2 k3)` with `[N1/x]`
  Answer 2: `k3` with `[N2/x]`

# Logic programming and Proof

Type Class in Functional Language(e.g. Haskell)

- ```
  k1 : Eq(x) => Eq(List(x))
  k2 : => Eq(Int)

  eq : Eq(x) => x -> x -> Bool

  test = eq d [1,3] [1,2, 3]
  ```
- What is the proof of `Eq(List(Int))`?
- `d = (k1 k2)` is a proof of `Eq(List(Int))`!
  ```

# Resolutions

`k : Stream(y) => Stream(cons(x, y))`

Query `Stream(cons(x,y))`

- SLD-resolution:
  $\{\texttt{Stream(cons(x,y))}\} \rightsquigarrow \{\texttt{Stream(y)}\} \rightsquigarrow$
  $\{\texttt{Stream(y2)}\} \rightsquigarrow \ldots$

Matcher: $\sigma t_1 \equiv t_2$, Unifier: $\sigma t_1 \equiv \sigma t_2$

# Resolutions

`k : Stream(y) => Stream(cons(x, y))`

Query `Stream(cons(x,y))`

- SLD-resolution:
  $\{\texttt{Stream(cons(x,y))}\} \rightsquigarrow \{\texttt{Stream(y)}\} \rightsquigarrow \{\texttt{Stream(y2)}\} \rightsquigarrow \ldots$
- Resolution by Term matching:
  $\{\texttt{Stream(cons(x,y))}\} \rightarrow \{\texttt{Stream(y)}\}$

Matcher: $\sigma t_1 \equiv t_2$, Unifier: $\sigma t_1 \equiv \sigma t_2$

# Resolutions

`k : Stream(y) => Stream(cons(x, y))`

Query `Stream(cons(x,y))`

- SLD-resolution:
  $\{\texttt{Stream(cons(x,y))}\} \rightsquigarrow \{\texttt{Stream(y)}\} \rightsquigarrow$
  $\{\texttt{Stream(y2)}\} \rightsquigarrow \ldots$
- Resolution by Term matching:
  $\{\texttt{Stream(cons(x,y))}\} \rightarrow \{\texttt{Stream(y)}\}$
- Structural Resolution(Matching + Unification):
  $\{\texttt{Stream(cons(x,y))}\} \rightarrow \{\texttt{Stream(y)}\}$
  $\hookrightarrow \{\texttt{Stream(cons(x1,y1))}\} \rightarrow \{\texttt{Stream(y1)}\}$
  $\hookrightarrow \{\texttt{Stream(cons(x2,y2))}\} \rightarrow \{\texttt{Stream(y2)}\}$
  $\hookrightarrow \{\texttt{Stream(cons(x3,y3))}\} \rightarrow \{\texttt{Stream(y3)}\}\ldots$

Matcher: $\sigma t_1 \equiv t_2$, Unifier: $\sigma t_1 \equiv \sigma t_2$

# A Few Definitions

- **Term-matching reduction:**
  $\Phi \vdash \{A_1, ..., A_i, ..., A_n\} \rightarrow_\kappa \{A_1, ..., \sigma B_1, ..., \sigma B_m, ..., A_n\}$, if there exists $\kappa : B_1, ..., B_m \Rightarrow C \in \Phi$ such that $\sigma C \equiv A_i$.

# A Few Definitions

- **Term-matching reduction:**
  $\Phi \vdash \{A_1, ..., A_i, ..., A_n\} \rightarrow_\kappa \{A_1, ..., \sigma B_1, ..., \sigma B_m, ..., A_n\}$, if there exists $\kappa : B_1, ..., B_m \Rightarrow C \in \Phi$ such that $\sigma C \equiv A_i$.
- **Unification reduction:**
  $\Phi \vdash \{A_1, ..., A_i, ..., A_n\} \rightsquigarrow_{\kappa, \gamma \cdot \gamma'} \{\gamma A_1, ..., \gamma B_1, ..., \gamma B_m, ..., \gamma A_n\}$, if there exists $\kappa : B_1, ..., B_m \Rightarrow C \in \Phi$ such that $\gamma C \equiv \gamma A_i$.

# A Few Definitions

- **Term-matching reduction:**
  $\Phi \vdash \{A_1, ..., A_i, ..., A_n\} \to_\kappa \{A_1, ..., \sigma B_1, ..., \sigma B_m, ..., A_n\}$, if there exists $\kappa : B_1, ..., B_m \Rightarrow C \in \Phi$ such that $\sigma C \equiv A_i$.
- **Unification reduction:**
  $\Phi \vdash \{A_1, ..., A_i, ..., A_n\} \rightsquigarrow_{\kappa,\gamma\cdot\gamma'} \{\gamma A_1, ..., \gamma B_1, ..., \gamma B_m, ..., \gamma A_n\}$, if there exists $\kappa : B_1, ..., B_m \Rightarrow C \in \Phi$ such that $\gamma C \equiv \gamma A_i$.
- **Substitutional reduction:**
  $\Phi \vdash \{A_1, ..., A_i, ..., A_n\} \hookrightarrow_{\kappa,\gamma\cdot\gamma'} \{\gamma A_1, ..., \gamma A_i, ..., \gamma A_n\}$, if there exists $\kappa : B_1, ..., B_m \Rightarrow C \in \Phi$ such that $\gamma C \equiv \gamma A_i$.

# A Few Definitions

- **Term-matching reduction:**
  $\Phi \vdash \{A_1, ..., A_i, ..., A_n\} \to_\kappa \{A_1, ..., \sigma B_1, ..., \sigma B_m, ..., A_n\}$, if there exists $\kappa : B_1, ..., B_m \Rightarrow C \in \Phi$ such that $\sigma C \equiv A_i$.
- **Unification reduction:**
  $\Phi \vdash \{A_1, ..., A_i, ..., A_n\} \rightsquigarrow_{\kappa,\gamma\cdot\gamma'} \{\gamma A_1, ..., \gamma B_1, ..., \gamma B_m, ..., \gamma A_n\}$, if there exists $\kappa : B_1, ..., B_m \Rightarrow C \in \Phi$ such that $\gamma C \equiv \gamma A_i$.
- **Substitutional reduction:**
  $\Phi \vdash \{A_1, ..., A_i, ..., A_n\} \hookrightarrow_{\kappa,\gamma\cdot\gamma'} \{\gamma A_1, ..., \gamma A_i, ..., \gamma A_n\}$, if there exists $\kappa : B_1, ..., B_m \Rightarrow C \in \Phi$ such that $\gamma C \equiv \gamma A_i$.
- **LP-TM:** $(\Phi, \to)$
  **LP-Unif:** $(\Phi, \rightsquigarrow)$
  **LP-Struct:** $(\Phi, \to^\mu \cdot \hookrightarrow^1)$

# LP-Unif and LP-Struct

Question 1. What is the relation between LP-Unif and LP-Struct?

- Again, the graph example

```
k1 : Connect(x, y), Connect(y, z) => Connect(x, z)
k2 : => Connect(N1, N2)
k3 : => Connect(N2, N3)
```

- `Connect(N1, N3)` in LP-Unif has a finite path.
- For LP-Struct:

$\{$`Connect(N1, N3)`$\} \rightarrow_{\kappa_1,[N_1/x,N_3/z]}$
$\{$`Connect(N1, y), Connect(y, N3)`$\} \rightarrow_{\kappa_1,[N_1/x,y/z]}$
$\{$`Connect(N1, y1), Connect(y1, y), Connect(y, N3)`$\}$
$\rightarrow_{\kappa_1} \cdots$

# A Notion of Productivity

- We say a logic program is *productive* if $\rightarrow$ is terminating
- Productive programs allow finite observation, e.g. stream
  $\{\texttt{Stream(cons(x,y))}\}\rightarrow\{\texttt{Stream(y)}\}$
  $\hookrightarrow \{\texttt{Stream(cons(x1,y1))}\}\rightarrow\{\texttt{Stream(y1)}\}$
  $\hookrightarrow \{\texttt{Stream(cons(x2,y2))}\}\rightarrow\{\texttt{Stream(y2)}\}$
  $\hookrightarrow \{\texttt{Stream(cons(x3,y3))}\}\rightarrow\{\texttt{Stream(y3)}\}\ldots$
- There are nonproductive programs, e.g. graph

# LP-Unif and LP-Struct

Question 2. Given $\rightarrow$ is terminating, what is the relation between LP-Unif and LP-Struct?

- `k1 :  => P(c)`
  `k2 : Q(x)=> P(x)`
- LP-Unif: `P(x)` $\rightsquigarrow \emptyset$
- LP-Struct: `P(x)` $\rightarrow$ `Q(x)`

# Realizability Transformation

## Reflecting proofs into formulas

```
k1 : Connect(x, y, u1), Connect(y, z, u2)
           => Connect(x, z, k1(u1, u2))
k2 : => Connect(N1, N2, k2)
k3 : => Connect(N2, N3, k3)
```

## LP-Struct:

$\{\texttt{Connect(N1, N3, u)}\} \hookrightarrow \{\texttt{Connect(N1, N3, k1(u1, u2))}\} \rightarrow$

$\{\texttt{Connect(N1, y, u1), Connect(y, N3, u2)}\} \hookrightarrow$

$\{\texttt{Connect(N1, N2, k2), Connect(N2, N3, u2)}\} \rightarrow$

$\{\texttt{Connect(N2, N3, u2)}\} \hookrightarrow \{\texttt{Connect(N2, N3, k3)}\} \rightarrow \emptyset$

## Final Answer: $[\texttt{k1(k2,k3)/u}]$

# Realizability Transformation

```
k1 :  => P(c, k1)
k2 : Q(x, u1)=> P(x, k2(u1))
```

LP-Struct: $\texttt{P(x, u)} \hookrightarrow \texttt{P(c, k1)} \rightarrow \emptyset$

Question 3: How to justify the realizatibility transformation?

# Use a Type System

- Girard's observation on atomic intuitionistic sequent calculus

$$\frac{\underline{A} \vdash D \quad \underline{B}, D \vdash C}{\underline{A}, \underline{B} \vdash C}\ cut \qquad \frac{\underline{B} \vdash C}{\sigma\underline{B} \vdash \sigma C}\ subst \qquad \frac{}{\underline{B} \vdash A}\ axiom$$

- $\vdash Q$ ?
- Internalized "$\vdash$" as "$\Rightarrow$"

$$\frac{(\kappa : \forall \underline{x}.F) \in \Phi}{\kappa : \forall \underline{x}.F}\ axiom \qquad \frac{e : F}{e : \forall \underline{x}.F}\ gen$$

$$\frac{e : \forall \underline{x}.F}{e : [\underline{t}/\underline{x}]F}\ inst \qquad \frac{e_1 : \underline{A} \Rightarrow D \quad e_2 : \underline{B}, D \Rightarrow C}{\lambda \underline{a}.\lambda \underline{b}.(e_2\ \underline{b})\ (e_1\ \underline{a}) : \underline{A}, \underline{B} \Rightarrow C}\ cut$$

# Some Results

- **Soundness of LP-Unif**
  If $\Phi \vdash \{A\} \rightsquigarrow^*_\gamma \emptyset$ , then there exists a proof $e : \Rightarrow \gamma A$ given axioms $\Phi$.
- **Soundness of LP-TM**
  If $\Phi \vdash \{A\} \rightarrow^* \emptyset$ , then there exists a proof $e : \Rightarrow A$ given axioms $\Phi$.

New! **Completeness for LP-Unif**
If there exists a proof $e : \Rightarrow A$ given axioms $\Phi$, then $\Phi \vdash \{A\} \rightsquigarrow^*_\gamma \emptyset$ for some $\gamma$.

# Realizability Transformation

## Realizability transformation $F$ on normal proofs

- $F(\kappa : A_1, ..., A_m \Rightarrow B) :=$
  $\kappa : A_1[y_1], ..., A_m[y_m] \Rightarrow B[f_\kappa(y_1, ..., y_m)]$
- $F(\lambda \underline{a}.n : A_1, ..., A_m \Rightarrow B) :=$
  $\lambda \underline{a}.n : A_1[y_1], ..., A_m[y_m] \Rightarrow B[[\![n]\!]_{[\underline{y}/\underline{a}]}]$

For $A \equiv P(\underline{x})$, we write $A[y] \equiv P(\underline{x}, y)$. Similarly, $A[t] \equiv P(\underline{x}, t)$

# Realizability Transformation

- Preserve Provability
  $\Phi \vdash n : \underline{A} \Rightarrow B$ implies $F(\Phi) \vdash F(n : \underline{A} \Rightarrow B)$
- Preserve Behavior of LP-Unif
  $\Phi \vdash \{A\} \rightsquigarrow^* \emptyset$ iff $F(\Phi) \vdash \{A[y]\} \rightsquigarrow^* \emptyset$
- Operational Equivalence of LP-Unif and LP-Struct
  $F(\Phi) \vdash \{A[y]\} \rightsquigarrow^* \emptyset$ iff $F(\Phi) \vdash \{A[y]\}(\rightarrow^{\mu} \cdot \hookrightarrow^{1})^*\emptyset$.
- Helps to identify productive and non-overlapping programs

# Summary and Future Work

- We define a type system to model LP-TM, LP-Unif and LP-Struct
- We formalize realizability transformation and show it preserves the proof content
- We show that LP-Unif and LP-Struct are operationally equivalent after the tranformation
- Future work: towards analyzing type class inference in Haskell
- Thank you!