A Type Theoretic Approach to Structural Resolution

Peng Fu, Ekaterina Komendantskaya

University of Dundee
School of Computing
Logic programming and Proof

- **k1**: $\text{Connect}(x, y), \text{Connect}(y, z) \Rightarrow \text{Connect}(x, z)$
- **k2**: $\Rightarrow \text{Connect}(N1, N2)$
- **k3**: $\Rightarrow \text{Connect}(N2, N3)$

- **Are there any proof of** $\text{Connect}(x, N3)$ **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)

- \( k_1 : \text{Eq}(x) \Rightarrow \text{Eq}(\text{List}(x)) \)
- \( k_2 : \Rightarrow \text{Eq}(\text{Int}) \)

\( \text{eq} : \text{Eq}(x) \Rightarrow x \rightarrow x \rightarrow \text{Bool} \)

\( \text{test} = \text{eq}\ d\ [1,3]\ [1,2,3] \)

- What is the proof of \( \text{Eq} \ (\text{List} \ \text{Int}) \)?
- \( d = (k_1 \ k_2) \) is a proof of \( \text{Eq} \ (\text{List} \ \text{Int}) \)!
Resolutions

$k : \text{Stream}(y) \Rightarrow \text{Stream}(\text{cons}(x, y))$

Query $\text{Stream}(\text{cons}(x, y))$

- **SLD-resolution:**
  \[
  \{\text{Stream}(\text{cons}(x, y))\} \rightarrow \{\text{Stream}(y)\} \rightarrow \{\text{Stream}(y_2)\} \rightarrow \ldots
  \]

Matcher $\sigma t_1 \equiv t_2$ Unifier $\sigma t_1 \equiv \sigma t_2$
Resolutions

\[ k : \text{Stream}(y) \Rightarrow \text{Stream}(\text{cons}(x, y)) \]

**Query** \( \text{Stream}(\text{cons}(x, y)) \)

- **SLD-resolution:**
  \[ \{ \text{Stream}(\text{cons}(x, y)) \} \rightsquigarrow \{ \text{Stream}(y) \} \rightsquigarrow \{ \text{Stream}(y2) \} \rightsquigarrow \ldots \]

- **Resolution by Term matching:**
  \[ \{ \text{Stream}(\text{cons}(x, y)) \} \rightarrow \{ \text{Stream}(y) \} \]

Matcher \( \sigma t_1 \equiv t_2 \)
Unifier \( \sigma t_1 \equiv \sigma t_2 \)
Resolutions

\[ k : \text{Stream}(y) \Rightarrow \text{Stream}(\text{cons}(x, y)) \]

Query \text{Stream}(\text{cons}(x, y))

- **SLD-resolution:**
  \[
  \{\text{Stream}(\text{cons}(x, y))\} \leadsto \{\text{Stream}(y)\} \leadsto \{\text{Stream}(y2)\} \leadsto \ldots
  \]

- **Resolution by Term matching:**
  \[
  \{\text{Stream}(\text{cons}(x, y))\} \rightarrow \{\text{Stream}(y)\}
  \]

- **Structural Resolution(Combine Matching and Unification):**
  \[
  \{\text{Stream}(\text{cons}(x, y))\} \rightarrow \{\text{Stream}(y)\}
  \leftarrow \{\text{Stream}(\text{cons}(x1, y1))\} \rightarrow \{\text{Stream}(y1)\}
  \leftarrow \{\text{Stream}(\text{cons}(x2, y2))\} \rightarrow \{\text{Stream}(y2)\}
  \leftarrow \{\text{Stream}(\text{cons}(x3, y3))\} \rightarrow \{\text{Stream}(y3)\}\ldots
  \]

Matcher \( \sigma t_1 \equiv t_2 \) Unifier \( \sigma t_1 \equiv \sigma t_2 \)
A Few Definitions

- **Term-matching reduction:**
  \[ \Phi \vdash \{A_1, ..., A_i, ..., A_n\} \rightarrow_\kappa \{A_1, ..., \sigma B_1, ..., \sigma B_m, ..., A_n\}, \text{ if there exists } \kappa : B_1, ..., B_m \Rightarrow C \in \Phi \text{ such that } C \mapsto_\sigma A_i. \]
A Few Definitions

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

- **Unification reduction:**
  \[ \Phi \vdash \{ A_1, ..., A_i, ..., A_n \} \rightsquigarrow_\kappa,\gamma,\gamma' \{ \gamma A_1, ..., \gamma B_1, ..., \gamma B_m, ..., \gamma A_n \}, \text{ if there exists } \kappa : B_1, ..., B_m \Rightarrow C \in \Phi \text{ such that } C \sim_\gamma A_i. \]
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 \( C \leftrightarrow_{\sigma} A_i \).

- **Unification reduction:**
  \[ \Phi \vdash \{A_1, ..., A_i, ..., A_n\} \rightsquigarrow_{\kappa, \gamma, \gamma'} \{\gamma A_1, ..., \gamma B_1, ..., \gamma B_m, ..., \gamma A_n\}, \]  
  if there exists \( \kappa : B_1, ..., B_m \Rightarrow C \in \Phi \) such that \( C \sim_{\gamma} A_i \).

- **Substitutional reduction:**
  \[ \Phi \vdash \{A_1, ..., A_i, ..., A_n\} \hookrightarrow_{\kappa, \gamma, \gamma'} \{\gamma A_1, ..., \gamma A_i, ..., \gamma A_n\}, \]  
  if there exists \( \kappa : B_1, ..., B_m \Rightarrow C \in \Phi \) such that \( C \sim_{\gamma} A_i \).
A Few Definitions

▶ Term-matching reduction:
\[ \Phi \vdash \{A_1, \ldots, A_i, \ldots, A_n\} \rightarrow_\kappa \{A_1, \ldots, \sigma B_1, \ldots, \sigma B_m, \ldots, A_n\} \], if there exists \( \kappa : B_1, \ldots, B_m \Rightarrow C \in \Phi \) such that \( C \leftrightarrow_\sigma A_i \).

▶ Unification reduction:
\[ \Phi \vdash \{A_1, \ldots, A_i, \ldots, A_n\} \leadsto_\kappa,\gamma,\gamma' \{\gamma A_1, \ldots, \gamma B_1, \ldots, \gamma B_m, \ldots, \gamma A_n\} \], if there exists \( \kappa : B_1, \ldots, B_m \Rightarrow C \in \Phi \) such that \( C \sim_\gamma A_i \).

▶ Substitutional reduction:
\[ \Phi \vdash \{A_1, \ldots, A_i, \ldots, A_n\} \hookrightarrow_\kappa,\gamma,\gamma' \{\gamma A_1, \ldots, \gamma A_i, \ldots, \gamma A_n\} \], if there exists \( \kappa : B_1, \ldots, B_m \Rightarrow C \in \Phi \) such that \( C \sim_\gamma A_i \).

▶ LP-TM: \( (\Phi, \rightarrow) \)
LP-Unif: \( (\Phi, \leadsto) \)
LP-Struct: \( (\Phi, \hookrightarrow^\mu \cdot \hookrightarrow^1) \)
Question 1. What is the relation between LP-Unif and LP-Struct?

- Again, the graph example

\[k_1 : \text{Connect}(x, y), \text{Connect}(y, z) \Rightarrow \text{Connect}(x, z)\]
\[k_2 : \Rightarrow \text{Connect}(N_1, N_2)\]
\[k_3 : \Rightarrow \text{Connect}(N_2, N_3)\]

- Connect\((N_1, N_3)\) in LP-Unif has a finite path.

- For LP-Struct:

\[
\{\text{Connect}(N_1, N_3)\} \rightarrow_{k_1, [N_1/x, N_3/z]} \{\text{Connect}(N_1, y), \text{Connect}(y, N_3)\} \rightarrow_{k_1, [N_1/x, y_1/z]} \\
\{\text{Connect}(N_1, y_1), \text{Connect}(y_1, y), \text{Connect}(y, N_3)\} \rightarrow_{k_1 \ldots}
\]
A Notion of Productivity

- We say a logic program is *productive* if $\rightarrow$ is terminating
- Inspired by productivity in FP
- Allow finite observation, e.g. Stream
  \[
  \{\text{Stream}(\text{cons}(x,y))\} \rightarrow \{\text{Stream}(y)\} \\
  \leftarrow \{\text{Stream}(\text{cons}(x_1,y_1))\} \rightarrow \{\text{Stream}(y_1)\} \\
  \leftarrow \{\text{Stream}(\text{cons}(x_2,y_2))\} \rightarrow \{\text{Stream}(y_2)\} \\
  \leftarrow \{\text{Stream}(\text{cons}(x_3,y_3))\} \rightarrow \{\text{Stream}(y_3)\} ...
  \]
- There are nonproductive programs, e.g. Connect
Question 2. What is the relation between LP-Unif and LP-Struct, given $\rightarrow$ is terminating?

- $k_1 : \Rightarrow P(c)$
- $k_2 : Q(x) \Rightarrow P(x)$

- **LP-Unif:** $P(x) \leadsto \emptyset$

- **LP-Struct:** $P(x) \rightarrow Q(x)$
Realizability Transformation

\[ \begin{align*}
  k_1 & : \text{Connect}(x, y, u_1), \text{Connect}(y, z, u_2) \\
       & \Rightarrow \text{Connect}(x, z, k_1(u_1, u_2)) \\
  k_2 & : \Rightarrow \text{Connect}(N_1, N_2, k_2) \\
  k_3 & : \Rightarrow \text{Connect}(N_2, N_3, k_3) \\
\end{align*} \]

LP-Struct:

\[ \begin{align*}
  \{ \text{Connect}(N_1, N_3, u) \} & \leftrightarrow \{ \text{Connect}(N_1, N_3, k_1(u_1, u_2)) \} \rightarrow \\
  \{ \text{Connect}(N_1, y, u_1), \text{Connect}(y, N_3, u_2) \} & \leftrightarrow \\
  \{ \text{Connect}(N_1, N_2, k_2), \text{Connect}(N_2, N_3, u_2) \} & \rightarrow \\
  \{ \text{Connect}(N_2, N_3, u_2) \} & \leftrightarrow \{ \text{Connect}(N_2, N_3, k_3) \} \rightarrow \emptyset \\
  [k_1(k_2, k_3)/u] \\
\]
Realizability Transformation

\[ k_1 : \Rightarrow P(c, k_1) \]
\[ k_2 : Q(x, u_1) \Rightarrow P(x, k_2(u_1)) \]

LP-Struct: \[ P(x, u) \leftrightarrow P(c, k_1) \rightarrow \emptyset \]

**Question 3**: How to justify the realizatibility transformation?
Use a Type System

- Girard’s observation on atomic intuitionistic sequent calculus

\[
\begin{align*}
B \vdash A & \quad \text{axiom} \\
\sigma B \vdash \sigma C & \quad \text{subst} \\
A \vdash D, B, D \vdash C & \quad \text{cut}
\end{align*}
\]

- Is \( \vdash Q \) provable?

- Internalized “\( \vdash \)” as “\( \Rightarrow \)”

\[
\begin{align*}
\frac{(\kappa : \forall x.F) \in \Phi}{\kappa : \forall x.F} & \quad \text{axiom} \\
\frac{e : F}{e : \forall x.F} & \quad \text{gen} \\
\frac{e : \forall x.F}{e : [t/x]F} & \quad \text{inst} \\
\frac{e_1 : A \Rightarrow D, e_2 : B, D \Rightarrow C}{\lambda a. \lambda b. (e_2 b) (e_1 a) : A, B \Rightarrow C} & \quad \text{cut}
\end{align*}
\]
Soundness Results

- **Soundness of LP-Unif**
  If $\Phi \vdash \{A\} \rightsquigarrow^* \emptyset$, 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 $\Phi$. 
Realizability Transformation

Realizability transformation $F$ on normal proofs

- $F(\kappa : A_1, \ldots, A_m \Rightarrow B) := \kappa : A_1[y_1], \ldots, A_m[y_m] \Rightarrow B[f_\kappa(y_1, \ldots, y_m)]$

- $F(\lambda a.n : A_1, \ldots, A_m \Rightarrow B) := \lambda a.n : A_1[y_1], \ldots, A_m[y_m] \Rightarrow B[[n][y/a]]$

For $A \equiv P(x)$, we write $A[y] \equiv P(x, y)$. Similarly, $A[t] \equiv P(x, t)$
Realizability Transformation

- **Preserve Provability**
  Given axioms $\Phi$, if $e : A \Rightarrow B$ holds with $e$ in normal form, then $F(e : A \Rightarrow B)$ holds for axioms $F(\Phi)$

- **Preserve Behavior of LP-Unif**
  $\Phi \vdash \{A\} \leadsto^* \emptyset$ iff $F(\Phi) \vdash \{A[y]\} \leadsto^* \emptyset$

- **Operational Equivalence of LP-Unif and LP-Struct**
  $F(\Phi) \vdash \{A[y]\} \leadsto^* \emptyset$ iff $F(\Phi) \vdash \{A[y]\}(\rightarrow^\mu \cdot \leftarrow^1)^*\emptyset$.

**New! Helps to prove the completeness result**
If there exists a proof $e : \Rightarrow A$ given axioms $\Phi$, then
$\Phi \vdash \{A\} \leadsto^\gamma \emptyset$ for some $\gamma$. 
Summary and Future Work

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