ALFA LerNET Project

DEC$^{\mathcal{T}}$ Research Group Presentation

Pascual Julián Iranzo

Pascual.Julian@uclm.es

http://www.inf-cr.uclm.es/www/pulian

General Presentation

- The $DEC^{\mathcal{T}}$ Group was founded in October 2000.
- It has 5 members and 3 research fellows.
- The aims of $DEC^{\mathcal{T}}$ Group are:
  - to develop formal methods for integrating multi-paradigm declarative languages,
  - program transformation,
  - implementation techniques.

Research Experience

- It has published more than 25 journal and international conference papers.
- It has participated in several research projects:

RESEARCH PROJECTS

International	National	Regional
3	3	2

Research Experience

Cooperation with other research groups

U. Complutense de Madrid	Spain	Mario Rodríguez
U. Complutense de Madrid	Spain	Ricardo Peña
U. de Málaga	Spain	Ernesto Pimentel
U. de Medellín	Colombia	Francisco Correa
U. Politécnica de Valencia	Spain	María Alpuente
Portland State University	USA	Sergio Antoy
Università di Udine	Italy	Moreno Falaschi

Conceptual Framework

- Multi-paradigm declarative programming.
- The aim is to integrate the best features of:
  - Logic programming (logical formulas and variables, partial data structures, non-deterministic search),
  - Functional programming (functions, equations, deterministic evaluation, nested terms and evaluation strategies),
  - Other paradigms: e.g., parallelism, fuzzy logic.

Research Framework

- SELF Project: Software Engineering and Lightweight Formalisms.
- We are responsible for two modules:
  - Multi-paradigm declarative languages: advanced implementation techniques,
  - Multi-paradigm declarative languages: transformations and fuzzy logic extensions.

Current Research Lines: Improving high level implementations

- Narrowing is the standard operational mechanism used in the integration of functional and logic languages.
- Narrowing = variable instantiation + rewriting.

Current Research Lines:
Improving high level implementations

- Example:

Program	Computation
$0 + N \;\rightarrow\; N$ $s(M) + N \;\rightarrow\; s(M+N)$	$\boxed{Z + s(0)}$

Current Research Lines: Improving high level implementations

- Example:

Program

$$0 + N \quad \rightarrow \quad N$$

$$s(M) + N \quad \rightarrow \quad s(M + N)$$

Computation

$$\boxed{Z + s(0)}$$

Rule: $0 + N_1 \rightarrow \quad N_1$.

Current Research Lines:
Improving high level implementations

- Example:

Program

$$0 + N \rightarrow N$$
$$s(M) + N \rightarrow s(M + N)$$

Computation

$$\boxed{Z + s(0)}$$

Rule: $0 + N_1 \rightarrow N_1$.

Unifier: $\{Z \mapsto 0, N_1 \mapsto s(0)$

Current Research Lines:
Improving high level implementations

- Example:

Program

$$0 + N \;\; \rightarrow \;\; N$$
$$s(M) + N \;\; \rightarrow \;\; s(M + N)$$

Computation

$$\boxed{0 + s(0)}$$

Rule: $0 + s(0) \rightarrow \; s(0)$.

Unifier: $\{Z \mapsto 0, N_1 \mapsto s(0)$

Current Research Lines:
Improving high level implementations

- Example:

Program

$$0 + N \rightarrow N$$
$$s(M) + N \rightarrow s(M + N)$$

Computation

$$\boxed{s(0)}$$

Answer: $\{Z \mapsto 0\}$

Current Research Lines:
Improving high level implementations

- Narrowing is a nondeterministic procedure.
- Unrestricted narrowing has a huge search space.
- Therefore, many narrowing strategies have been studied during last decade.
- Since laziness is a valuable feature of functional logic languages, lazy narrowing strategies play an important role.

Current Research Lines:
Improving high level implementations

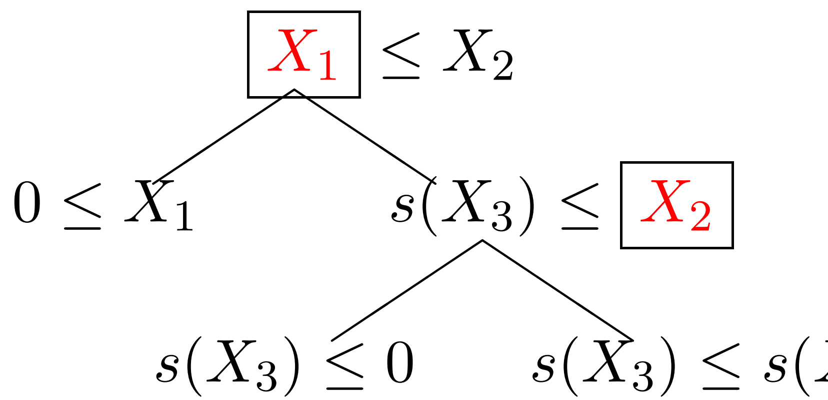
- Needed Narrowing (NN) has been postulated optimal from several points of view.
- NN is the standard operational mechanism of functional logic languages.
- The defi nition of NN makes use of the notion of a defi nitional tree.

Current Research Lines:
Improving high level implementations

- A Defi nitional treeis a structure which contains all the information about the program rules defi ning a function and it guides the computation.

- **Example**:

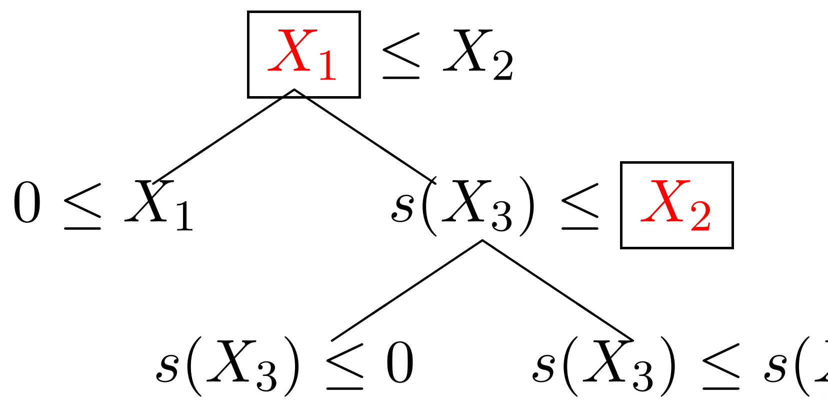
$$
\begin{aligned}
0 \leq N &\rightarrow \quad true \\
s(M) \leq 0 &\rightarrow \quad false \\
s(M) \leq s(N) &\rightarrow \quad M \leq N
\end{aligned}
$$



Current Research Lines:
Improving high level implementations

- A Definitional tree is a structure which contains all the information about the program rules defining a function and it guides the computation.
- **Example**:

$$\boxed{s \leq t}$$



Current Research Lines:
Improving high level implementations

- High level implementations rely on a two-phase transformation procedure that consists of:

1. an algorithm that obtains a representation for the definitional trees associated with a functional logic program;
2. an algorithm that visits the nodes of the definitional trees, generating a Prolog clause for each visited node.

Current Research Lines:
Improving high level implementations

- **Example**:

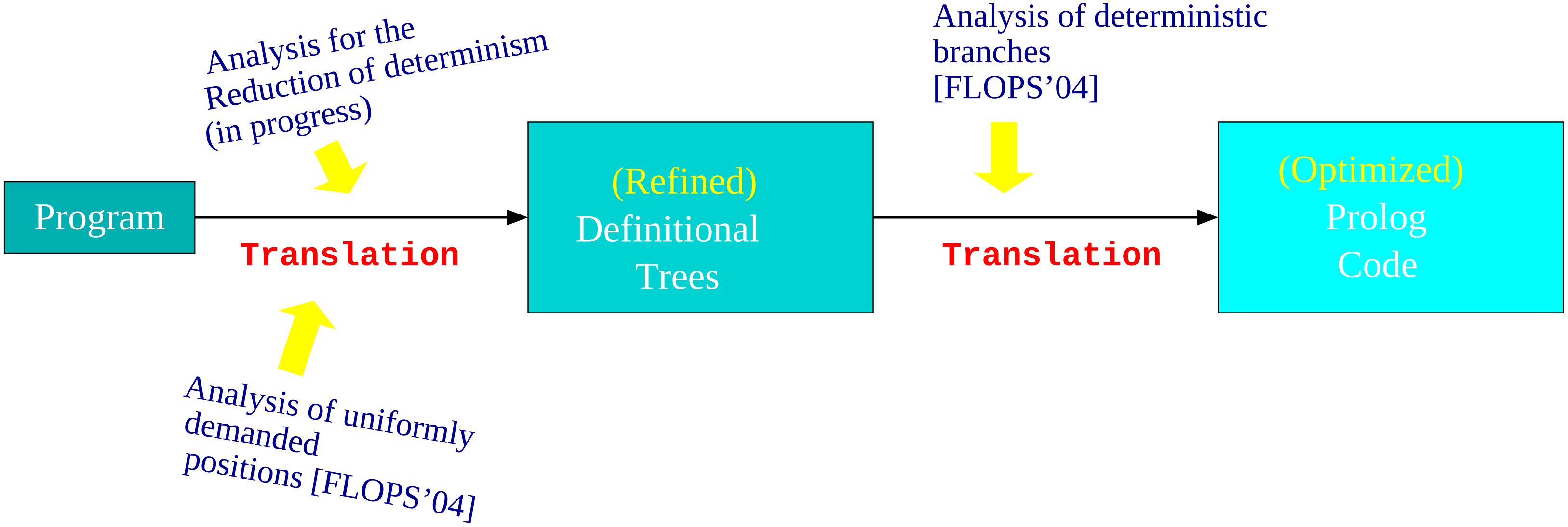
```
% Clause for the root node:
leq(X1,X2,H):- hnf(X1,HX1),leq_1(HX1,X2,H).

% Clauses for the remainder nodes:
leq_1(0,_,true).
leq_1(s(X3),X2,H):- hnf(X2,HX2),leq_1_s_2(HX2,H).
leq_1_s_2(0,false).
leq_1_s_2(s(X4),H):- hnf(leq(X3,X4),H).
```

Current Research Lines:
Improving high level implementations

- Definitional trees play a central role in NN implementations (into Prolog).
- Improvements in their representation and analysis will be worthwhile.
- **Main Goal**: to define implementation techniques that produce an optimal complied Prolog code, by analyzing definitional trees.

Current Research Lines:
Improving high level implementations



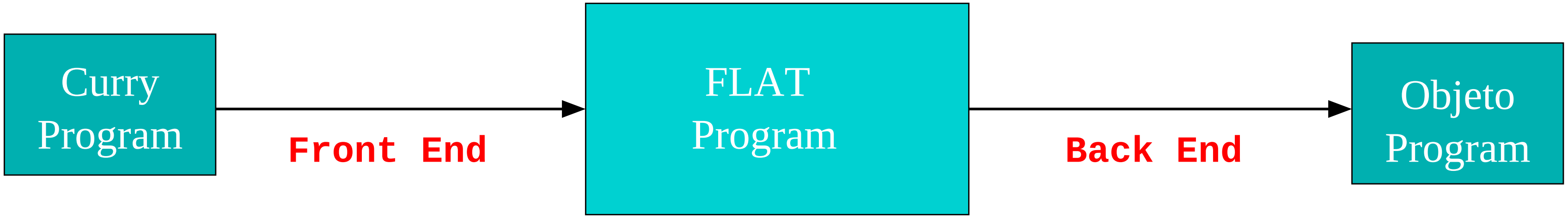
- The **analysis of definitional trees** allow us to improve high level implementations.

Current Research Lines:
Improving high level implementations

- **Next goals**:
  - to incorporate the above mentioned techniques into the **curry2prolog** compiler;
  - to study how to handle the implicit parallelism of the original program.

Current Research Lines:
Compilation by program transformation

- Curry is a modern functional logic programming language.
- Curry programs are translated into an intermediate language before their compilation to object code.



Current Research Lines:
Compilation by program transformation

The FLAT Curry language

$$
\begin{array}{lcl}
\mathcal{R} & ::= & D_1, \ldots, D_m \\
D & ::= & f(x_1, \ldots, x_n) = e \\
e & ::= & x \\
 & & \mid c(e_1, \ldots, e_n) \\
 & & \mid f(e_1, \ldots, e_n) \\
 & & \mid case\ e\ of\ \{p_1 \rightarrow e_1, \ldots, p_n \rightarrow e_n\} \\
 & & \mid fcase\ e\ of\ \{p_1 \rightarrow e_1, \ldots, p_n \rightarrow e_n\} \\
p & ::= & c(x_1, \ldots, x_n)
\end{array}
$$

Current Research Lines: Compilation by program transformation

- **Example**:

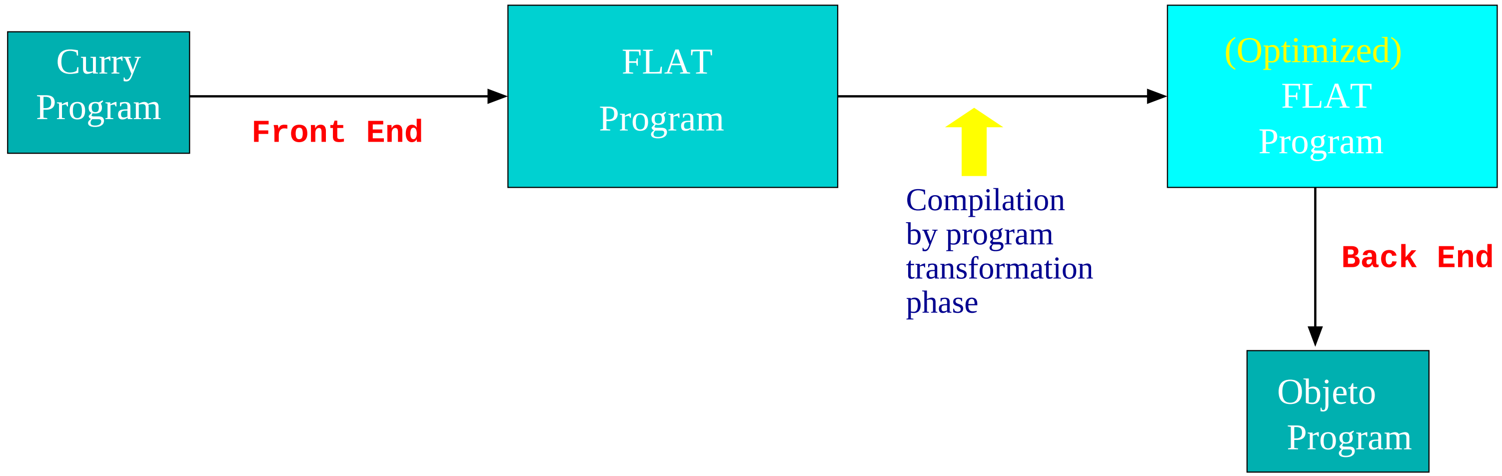
$$
\begin{array}{l}
X_1 \leq X_2 = \quad fcase\ X_1\ of \\
\qquad\qquad\qquad \{0 \to true \\
\qquad\qquad\qquad\ \ s(X_3) \to \quad fcase\ X_2\ of \\
\qquad\qquad\qquad\qquad\qquad\qquad \{0 \to \quad false \\
\qquad\qquad\qquad\qquad\qquad\qquad\ \ s(X_4) \to \quad X_3 \leq X_4\} \\
\qquad\qquad\qquad \}
\end{array}
$$

Current Research Lines:
Compilation by program transformation

- **Compilation by program transformation**: set of source-to-source transformations for optimizing a intermediate level code of a programming language.
- The key idea is to optimize programs written in a core language: Flat Curry.

Current Research Lines: Compilation by program transformation

- The process takes place before code generation at compiling time.



Current Research Lines:
Compilation by program transformation

- In the Functional Programming community, the idea of compilation by program transformation has received great attention.
- **Goal**: to investigate if it is possible to adapt (for Curry programs) techniques developed in the Functional Programming area.
    - Unfolding; case-of-case transformations; algebraic replacements...

Current Research Lines:
Similarity Narrowing

- Fuzzy Logic provides a mathematical background for modeling uncertainty.
- The introduction of fuzzy techniques into declarative languages:
  - may contribute to increase their expressiveness;
  - allows us to deal with a declarative approach to fuzzy system specifi cation.

Current Research Lines: Similarity Narrowing

- Several fuzzy logic programming systems have been developed. However, any effort towards that direction has been done in the fi eld of multi-paradigm declarative languages.
- **Goal**:
  - to integrate fuzzy notions into the framework of multi-paradigm declarative languages.

Current Research Lines: Similarity Narrowing

- There is no common method for introducing fuzzy concepts (into logic programming). We think there exist two major approaches:
  - to replace the syntactic unifi cation mechanism by a fuzzy unifi cation algorithm(that provides a numerical value, called the *unification degree*).
  - to consider programs as fuzzy subsets of rules, where the *truth degree* of each rule is explicitly annotated.

Current Research Lines:
Similarity Narrowing

- As a first approach, we want to adapt the narrowing operational mechanism by replacing the classical unification algorithm.
- Maria Sessa's weak unification algorithm:
  - It is an extension of the Martelli-Montanari unification algorithm with similarity relations.
  - $\{f(t_1, \ldots, t_n) = g(s_1, \ldots, s_n)\}$ unifies if f sim $g > 0$ and $\{t_1 = s_1, \ldots, t_n = s_n)\}$ unifies.

Current Research Lines:
Similarity Narrowing

- Similarity Narrowing = weak unifi cation + rewriting.
- Derivation: $\langle t, \alpha \rangle \overset{\sigma}{\leadsto}{}^{*} \langle s, \gamma \rangle$
- Output: $\langle s, \sigma, \gamma \rangle$
  - s is a value.
  - σ is the (partial) computed answer.
  - γ is the computed similarity degree.

Current Research Lines:
Similarity Narrowing

- Main Goals:
  - to study the viability of this approach;
  - to establish its formal properties;
  - to implement multi-paradigm languages based on similarity narrowing.