Refined Definitional Trees and Prolog Implementations of Narrowing

Pascual Julián Iranzo

Pascual.Julian@uclm.es.

Universidad de Castilla – La Mancha

Departamento de Informática

Ciudad Real (España)

Outline of the work

1.- Introduction and Preliminaries.

Outline of the work

1.- Introduction and Preliminaries.

2.- A Refined Representation of Definitional Trees.

Outline of the work

1.- Introduction and Preliminaries.

2.- A Refined Representation of Definitional Trees.

3.- Improving Narrowing Implementations into Prolog.

Outline of the work

1.- Introduction and Preliminaries.

2.- A Refined Representation of Definitional Trees.

3.- Improving Narrowing Implementations into Prolog.

4.- Experiments.

Outline of the work

1.- Introduction and Preliminaries.

2.- A Refined Representation of Definitional Trees.

3.- Improving Narrowing Implementations into Prolog.

4.- Experiments.

5.- Discussion and Conclusions.

Outline of the work

1.- Introduction and Preliminaries.

2.- A Refined Representation of Definitional Trees.

3.- Improving Narrowing Implementations into Prolog.

4.- Experiments.

5.- Discussion and Conclusions.

6.- Future Work.

Outline of the work

1.- Introduction and Preliminaries.
2.- A Refined Representation of Definitional Trees.
3.- Improving Narrowing Implementations into Prolog.
4.- Experiments.
5.- Discussion and Conclusions.
6.- Future Work.

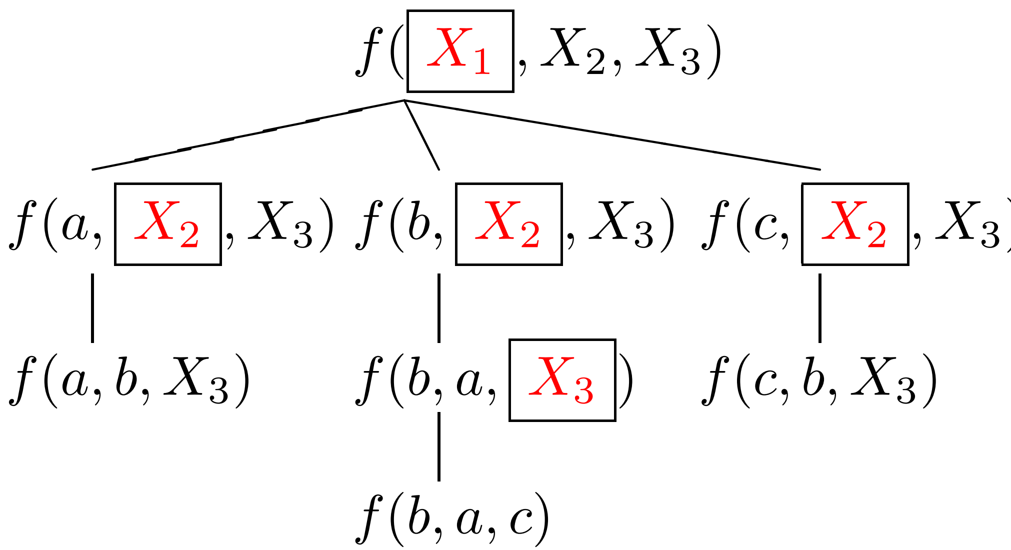
Introduction and Preliminaries: Definitional trees

- Needed Narrowing (NN) is the standard operational mechanism of functional logic languages.
- The definition of NN makes use of the notion of a definitional tree.
- A Definitional tree is a structure which contains all the information about the program rules defining a function and guides the computation.

Introduction and Preliminaries: Definitional trees

- **Example**:

$$R_1 : f(a, b, X) \to r_1,$$
$$R_2 : f(b, a, c) \to r_2,$$
$$R_3 : f(c, b, X) \to r_3.$$



Definitional tree of f.

Introduction and Preliminaries: Narrowing Implementations into Prolog.

- A great effort has been done to provide these languages with high level implementations of NN into Prolog:
  - Rodríguez–Artalejo et al. [PLILP'1993]

Introduction and Preliminaries: Narrowing Implementations into Prolog.

- A great effort has been done to provide these languages with high level implementations of NN into Prolog:
  - Rodríguez–Artalejo et al. [PLILP'1993]
  - Hanus [LOPSTR'1995]

Introduction and Preliminaries: Narrowing Implementations into Prolog.

- A great effort has been done to provide these languages with high level implementations of NN into Prolog:
  - Rodríguez–Artalejo et al. [PLILP'1993]
  - Hanus [LOPSTR'1995]
  - Antoy and Hanus [FroCos'2000]

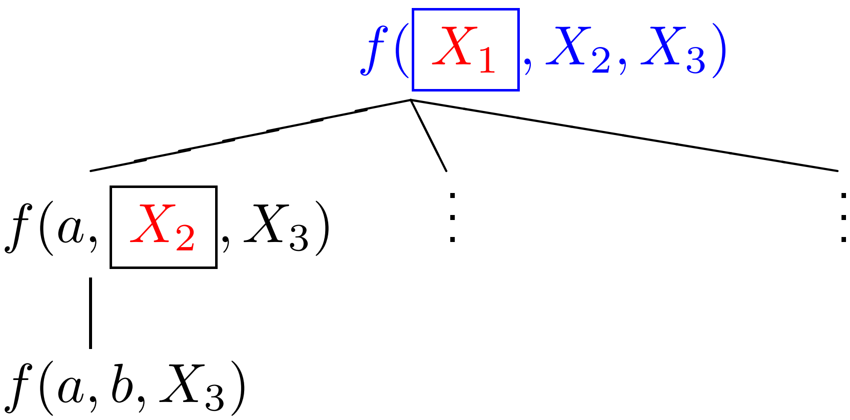
Introduction and Preliminaries: Narrowing Implementations into Prolog.

- These 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.

Introduction and Preliminaries: Narrowing Implementations into Prolog.

- **Example**:

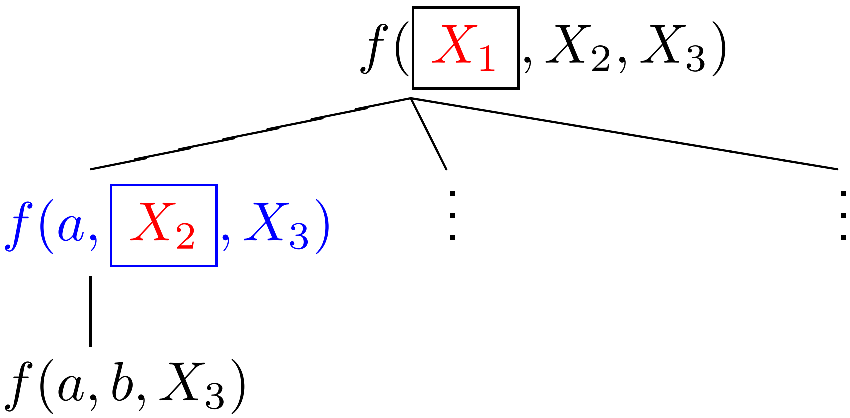


Definitional tree of f.

```
% Clause for the root node
f(X1, X2, X3, H) :-
hnf(X1, HX1),
f_1(HX1, X2, X3, H).
```

Introduction and Preliminaries:
Narrowing Implementations into Prolog.

- **Example:**

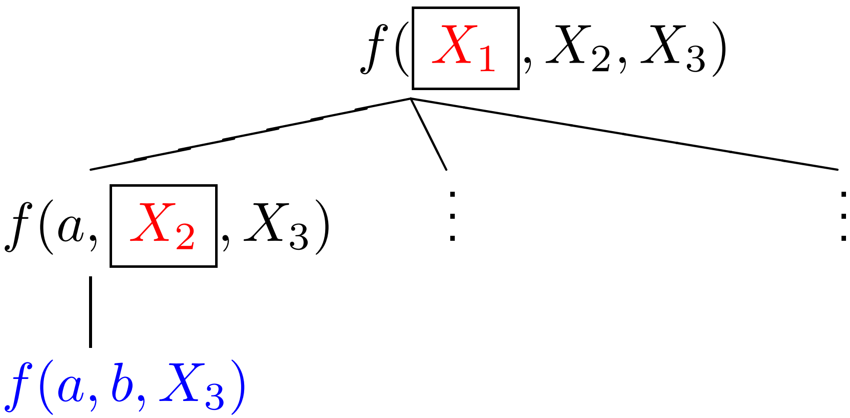


Definitional tree of f.

```
% Clause for the intermediate
% node:
f_1(a, X2, X3, H):-
hnf(X2, HX2),
f_1_a_2(HX2, X3, H).
```

Introducction and Preliminaries: Narrowing Implementations into Prolog.

- **Example**:



% Clause for the leaf node:

f_1_a_2(b, X3, H):- hnf(r1, H).

Definitional tree of f.

Introducction and Preliminaries:
Narrowing Implementations into Prolog.

- After visiting the whole definitional tree we obtain:

```
f(X1,X2,X3,H) :- hnf(X1,HX1), f_1(HX1,X2,X3,H).

f_1(a,X2,X3,H):- hnf(X2,HX2), f_1_a_2(HX2,X3,H).
f_1_a_2(b,X3,H):- hnf(r1,H).
f_1(b,X2,X3,H):- hnf(X2,HX2), f_1_b_2(HX2,X3,H).
f_1_b_2(a,X3,H):- hnf(X3,HX3), f_1_b_2_a_3(HX3,H).
f_1_b_2_a_3(c,H):- hnf(r2,H).
f_1(c,X2,X3,H):- hnf(X2,HX2), f_1_c_2(HX2,X3,H).
f_1_c_2(b,X3,H):- hnf(r3,H).
```

Introduction and Preliminaries: Aim of the Work

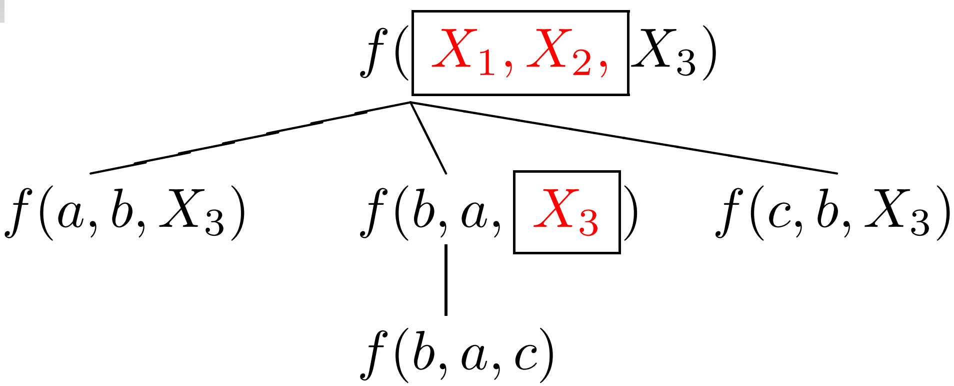
- Definitional trees play a central role in NN implementations (into Prolog).
- Improvements in their representation will be worthwhile.
- **Goal**: to study a refined representation of definitional trees that may introduce improvements in the quality of the Prolog code.

A Refined Representation of Definitional Trees.

- It is noteworthy that the function f has two definitional trees:
  - the one just depicted in the fourth slide;
  - a second one obtained by exploiting position 2 of the generic pattern $f(X_1, X_2, X_3)$.
- The generic pattern $f(X_1, X_2, X_3)$ has two inductive positions.

Refined Definitional Trees and Prolog Implementations of Narrowing

A Refined Representation of Definitional Trees.



Refined definitional tree of f.

- We can take advantage of this situation if we "simultaneously" exploit both positions.

- The new representation cuts the number of nodes from eight to five nodes.

A Refined Representation of Definitional Trees.

- The **main idea of the refinement**: when a pattern has several inductive positions, exploit them altogether.
- We expect the following advantages:
  - **Theoretical**: Determinism in the selection of definitional trees.
  - **Practical**: gains in memory allocation and (maybe) in execution time.

Building Refined Definitional Trees.

- We need a criterion to detect inductive positions.
- We use the concept of uniformly demanded position: Rodríguez–Artalejo et al. [PLILP'1993].
- A variable position of a pattern is uniformly demanded iff a constructor symbol appears at the corresponding position of each lhs subsumed by the pattern.

Building Refined Definitional Trees.

- **Example:**

$$f(X_1, X_2, X_3)$$

$$R_1 : f(a, b, X) \rightarrow r_1,$$

$$R_2 : f(b, a, c) \rightarrow r_2,$$

$$R_3 : f(c, b, X) \rightarrow r_3.$$

Position 1 of the pattern $f(X_1, X_2, X_3)$ is demanded by rule R_1.

Building Refined Definitional Trees.

- **Example**:

$$f(X_1, X_2, X_3)$$

$$R_1 : f(a, b, X) \rightarrow r_1,$$

$$R_2 : f(b, a, c) \rightarrow r_2,$$

$$R_3 : f(c, b, X) \rightarrow r_3.$$

Position 1 of the pattern $f(X_1, X_2, X_3)$ is uniformly demanded, since it is demanded by all rules.

Building Refined Definitional Trees.

- **Example**:

$$f(X_1, X_2, X_3)$$

$$R_1 : f(a, b, X) \rightarrow r_1,$$

$$R_2 : f(b, a, c) \rightarrow r_2,$$

$$R_3 : f(c, b, X) \rightarrow r_3.$$

Position 3 of the pattern $f(X_1, X_2, X_3)$ is demanded by rule R_2.

Building Refined Definitional Trees.

- **Example**:

$f(X_1, X_2, X_3)$

$R_1 : f(a, b, X) \to r_1,$

$R_2 : f(b, a, c) \to r_2,$

$R_3 : f(c, b, X) \to r_3.$

Position 3 of the pattern $f(X_1, X_2, X_3)$ is NOT uniformly demanded, since it is NOT demanded by all rules.

Building Refined Definitional Trees.

- **Proposition**:
  - $\mathcal{R}$ an inductively sequential TRS
  - π be the pattern of a branch node of a definitional tree of a function defined by $\mathcal{R}$.

> If o is an inductive position of π then o is uniformly demanded by $\mathcal{R}_\pi$.

- This proposition provides a necessary condition for a position to be inductively sequential.

Building Refined Definitional Trees.

- **Algorithm** (rpdt): Given a pattern,

1. select a tuple of uniformly demanded positions; fix them as inductive positions and generate the child nodes.

2. If the pattern doesn't have uniformly demanded positions and it is a variant of a lhs of a rule, generate a leaf node and go on with its brothers.

3. Otherwise, return a fail condition.

Building Refined Definitional Trees.

- **Example**: The algorithm in action.

$$rpdt(f(X_1, X_2, X_3), \mathcal{R})$$

Building Refined Definitional Trees.

- **Example**: The algorithm in action.

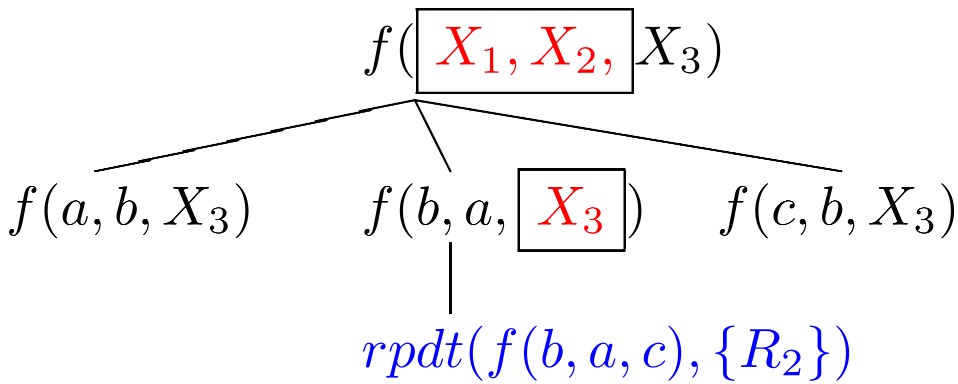
$$f(\boxed{X_1, X_2,}\, X_3)$$

$$rpdt(f(a, b, X_3), \{R_1\}) \quad rpdt(f(b, a, X_3), \{R_2\}) \quad rpdt(f(c, b, X_3), \{R_3\})$$

Refined Definitional Trees and Prolog Implementations of Narrowing

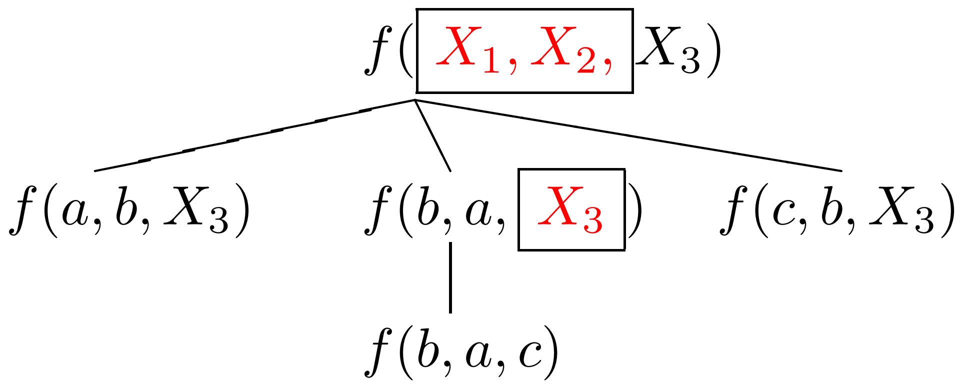
Building Refined Definitional Trees.

- **Example**: The algorithm in action.



Building Refined Definitional Trees.

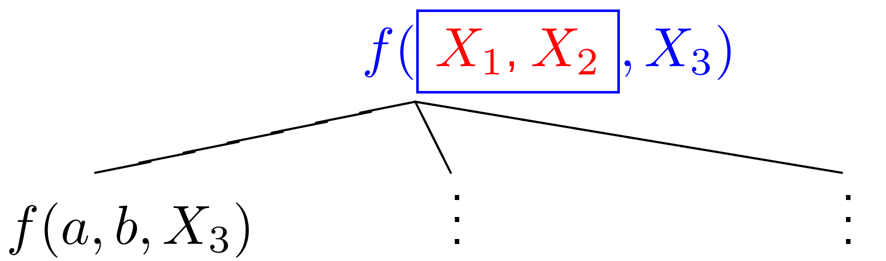
- **Example**: The algorithm in action.



Refined Definitional Trees and Prolog Implementations of Narrowing

Improving Narrowing Implementations into Prolog.

- We can use the new representation of definitional trees to guide the compilation process.
- For our running example, we obtain:

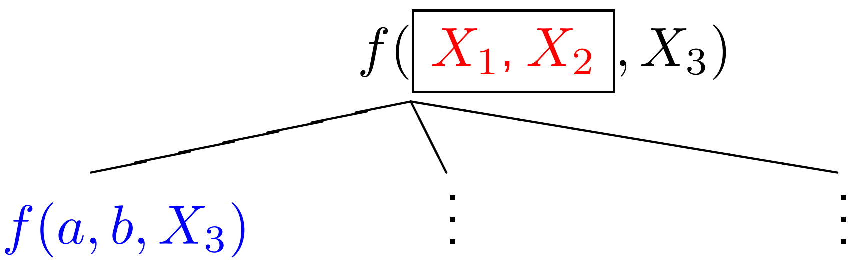


Refi ned defi nitional tree of f.

```
% Clause for the root node
f(X1, X2, X3, H) :-
hnf(X1, HX1), hnf(X2, HX2),
f_1_2(HX1, HX2, X3, H).
```

Improving Narrowing Implementations into Prolog.

- We can use the new representation of definitional trees to guide the compilation process.
- For our running example, we obtain:



Refi ned defi nitional tree of f.

```
% Clause for the root node
f_1_2(a, b, X3, H) :- hnf(r1, H).
```

Improving Narrowing Implementations into Prolog.

- After visiting the whole refined definitional tree, we obtain:

```
f(X1,X2,X3,H) :- hnf(X1,HX1), hnf(X2,HX2), f_1_2(HX1,HX2,X3,H).

f_1_2(a,b,X3,H):- hnf(r1,H).

f_1_2(b,a,X3,H):- hnf(X3,HX3), f_1_2_b_a(HX3,H).

f_1_2_b_a(c,H):- hnf(r2,H).

f_1_2(c,b,X3,H):- hnf(r3,H).
```

- The number of clauses have been reduced.

Experiments.

- We have made some small experiments to verify the effectiveness of our proposal.

Benchmark	Term	Speedup	G. stack Imp.
family	$grandfather(\_,\_)$	19.9 %	0 %
geq	$geq(100000, 99999)$	4.6 %	16.2 %
geq	$geq(99999, 100000)$	4.3 %	16.2 %
xor	$xor(\_,\_)$	18.5 %	0 %
zip	$zip(L1, L2)$	3.6 %	5.5 %
zip3	$zip3(L1, L2, L2)$	4.5 %	10 %
	Average	9.2 %	7.9 %

Discussion and Conclusions: Problems.

- It is difficult to evaluate the impact of the new compilation technique over the whole system.
- There are few opportunities to apply our technique.
- The performance of our translation technique may be in danger when a computation does not terminate or fails.

Discussion and Conclusions: Advantages.

- Determinism in the selection of definitional trees.
- There is some margin for the improvement of execution time and memory allocation.
- Our simple translation technique is able to eliminate some *ad hoc* artifices.
- It can be introduced with a modest programming effort in standard implementations of needed narrowing (e.g. Curry or $\mathcal{TOY}$)

Future Work.

- We want to deal with the problem of failing derivations in order to guarantee no slowdowns.
- We like to study how *clause indexing* relates with our work.
- We aim to investigate how definitional trees may be used as a guide to introduce selective program transformation techniques.