PrettyCLP: a Light Java Implementation for Teaching CLP

Alessio Stalla[1] Davide Zanucco[2]
Agostino Dovier[2] Viviana Mascardi[1]

DISI - Univ. of Genova,
alessiostalla@gmail.com,mascardi@disi.unige.it

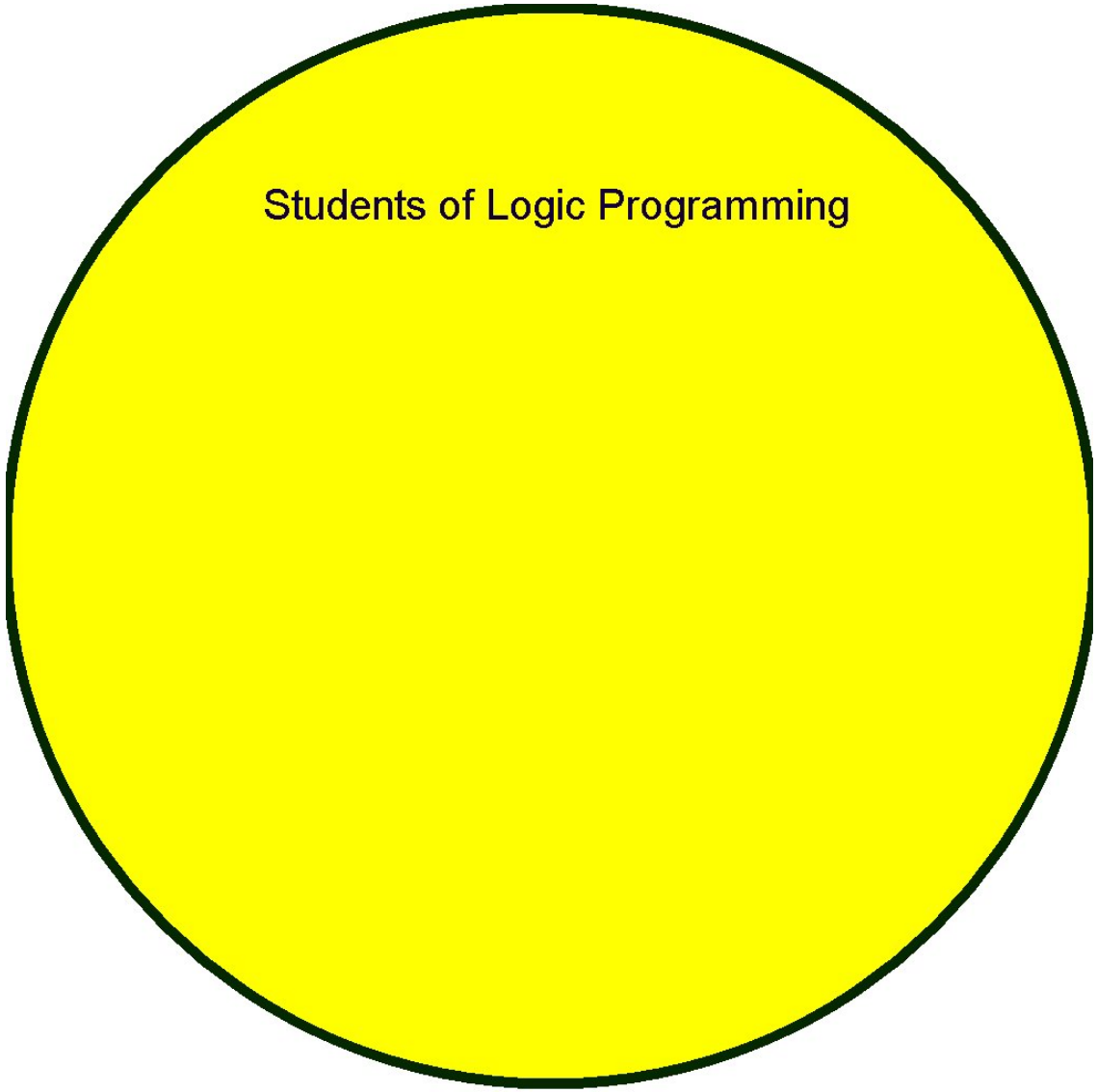
DIMI - Univ. of Udine,
zanucco.davide@spes.uniud.it,agostino.dovier@uniud.it

Pescara, September 2nd, 2011

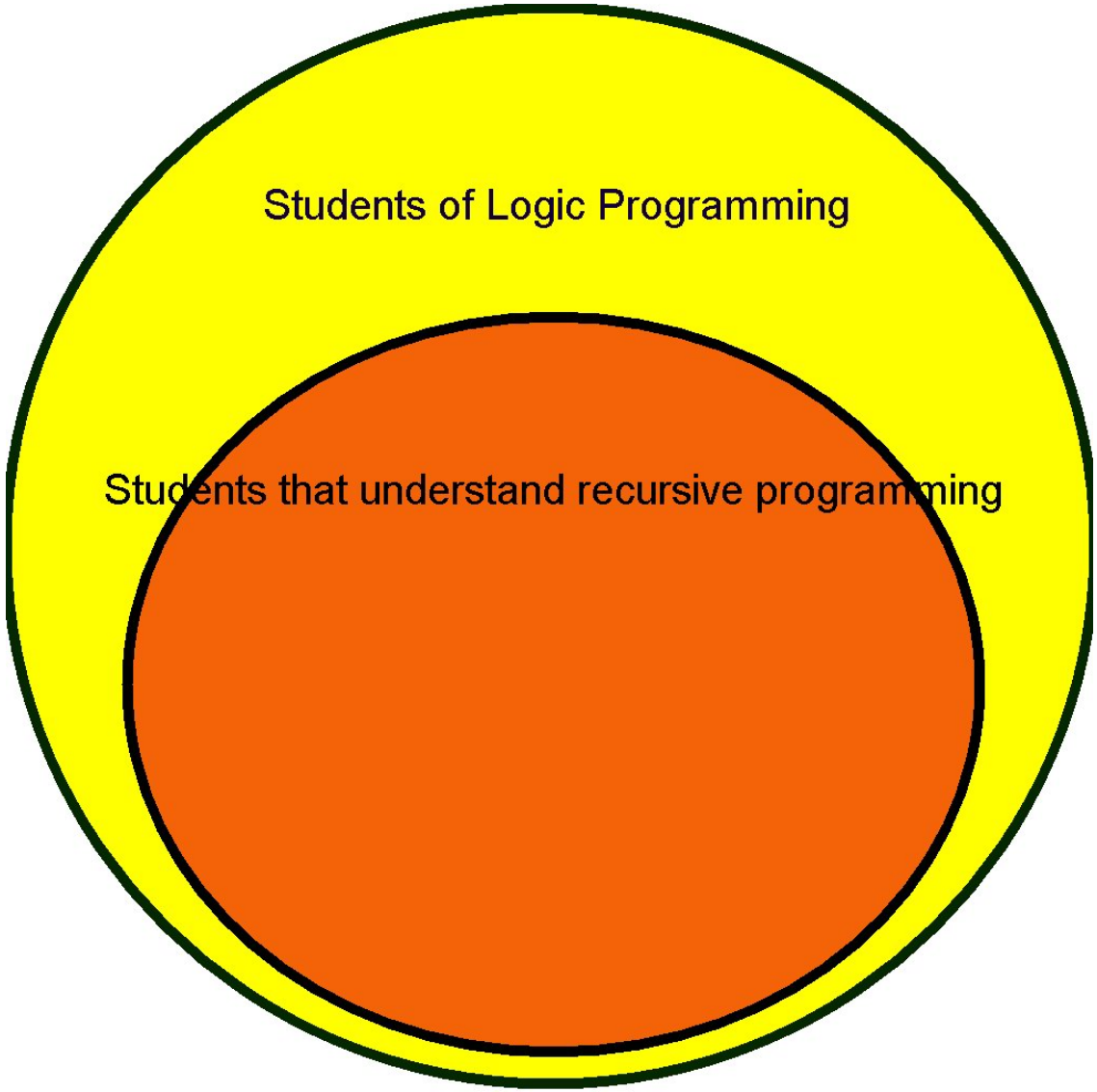
Summary

1. Introduction
2. Pretty Prolog
3. CLP(FD) basics
4. Demo
5. Conclusions

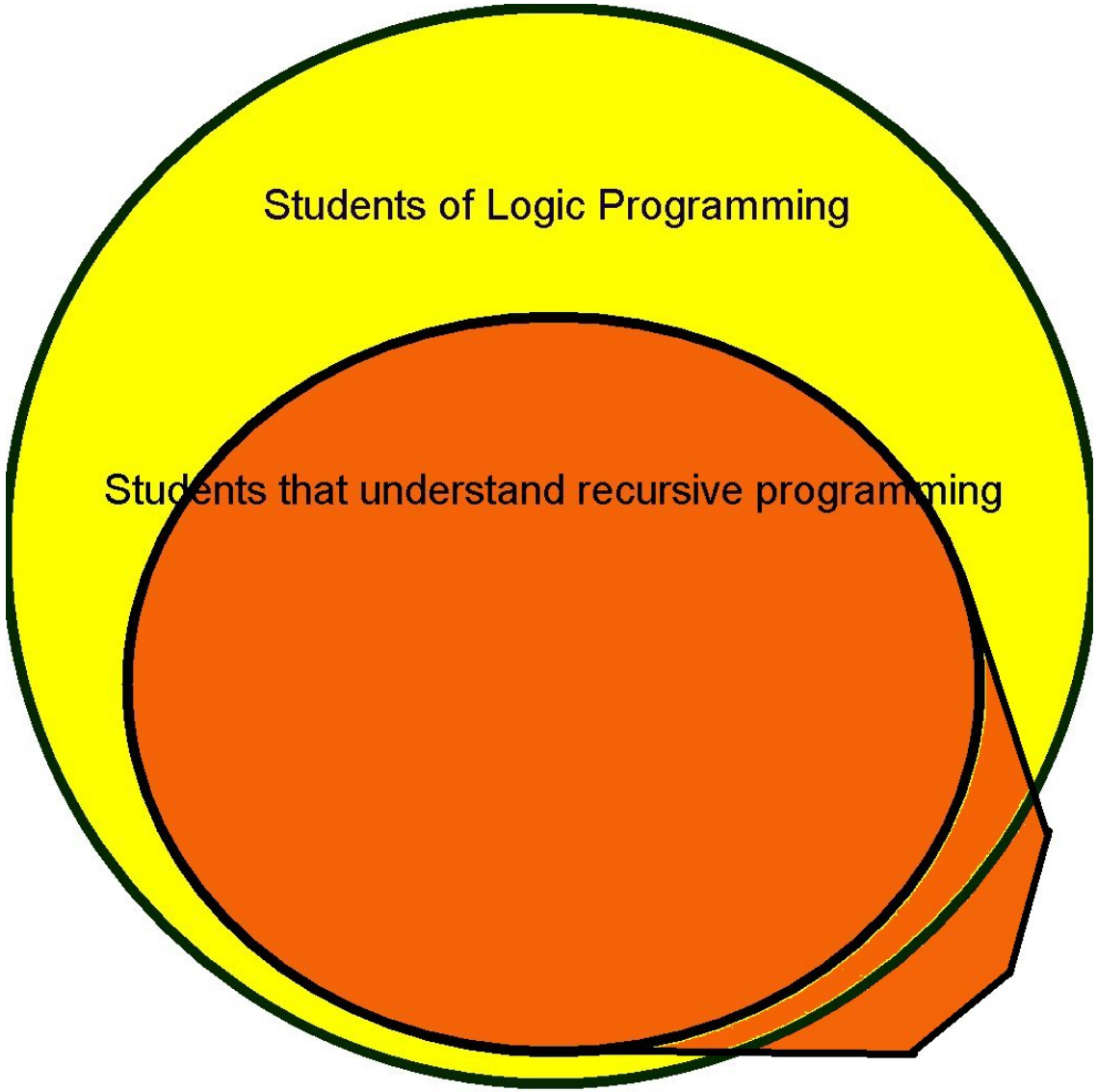
Introduction



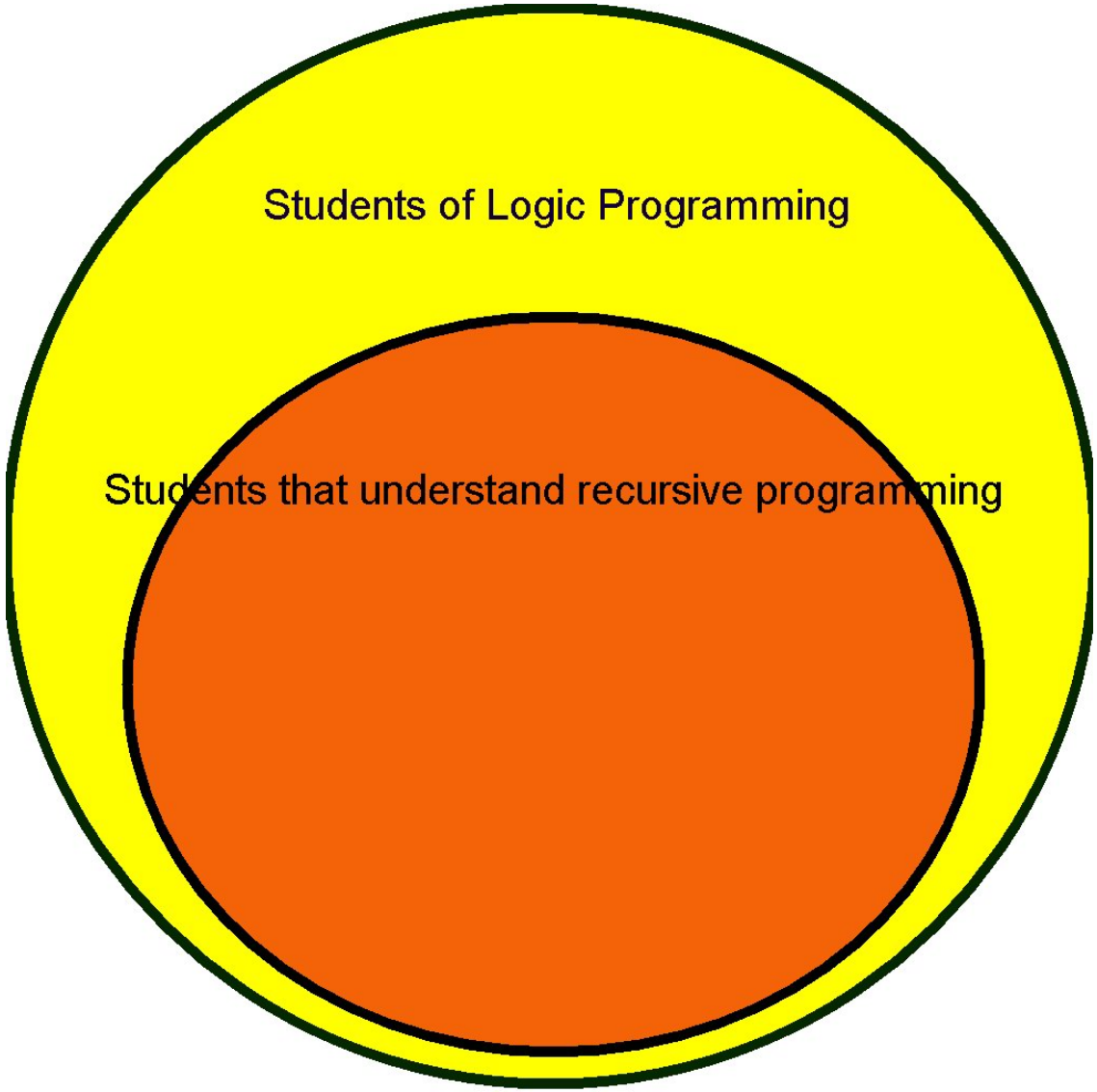
Introduction



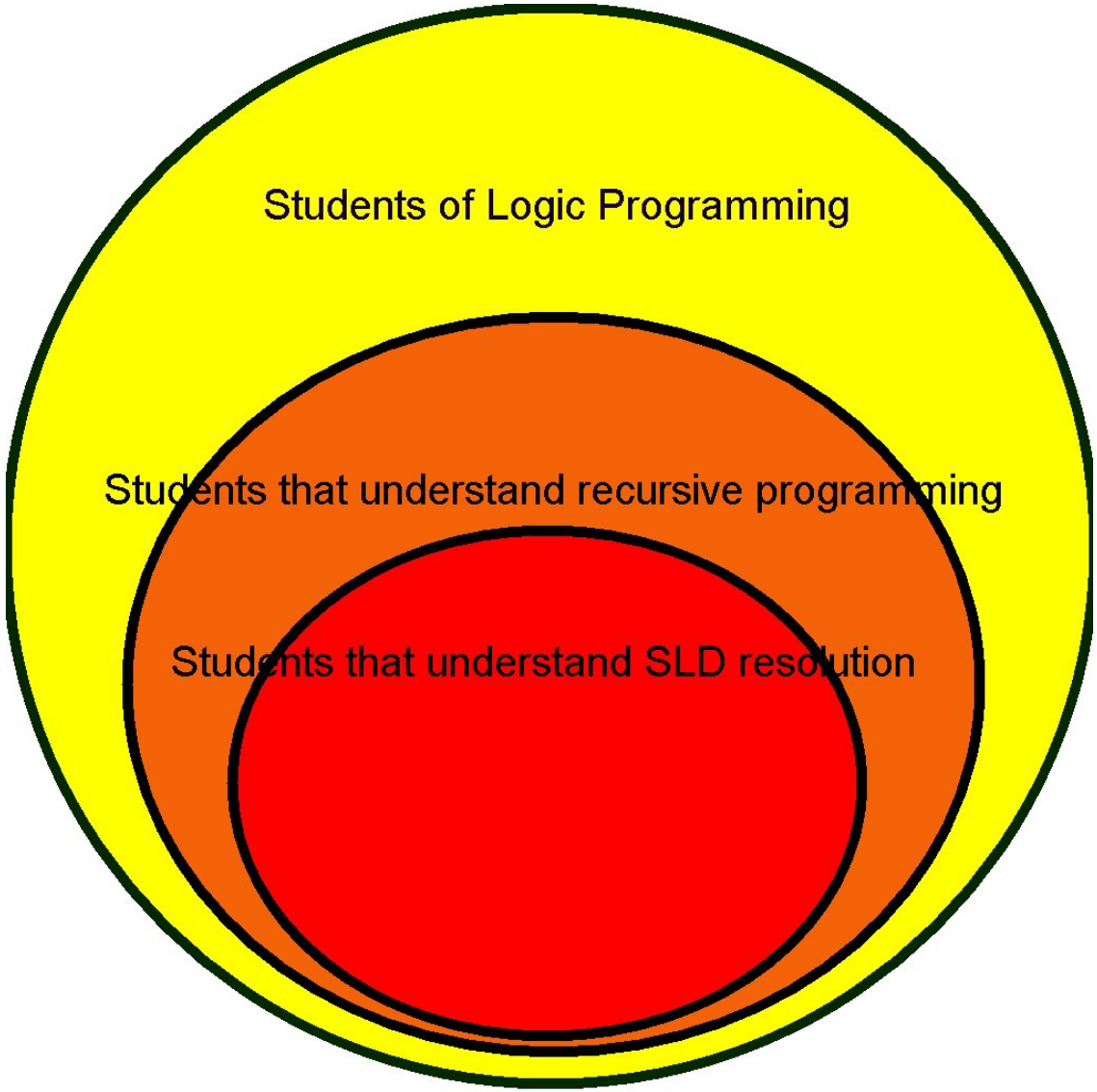
Introduction



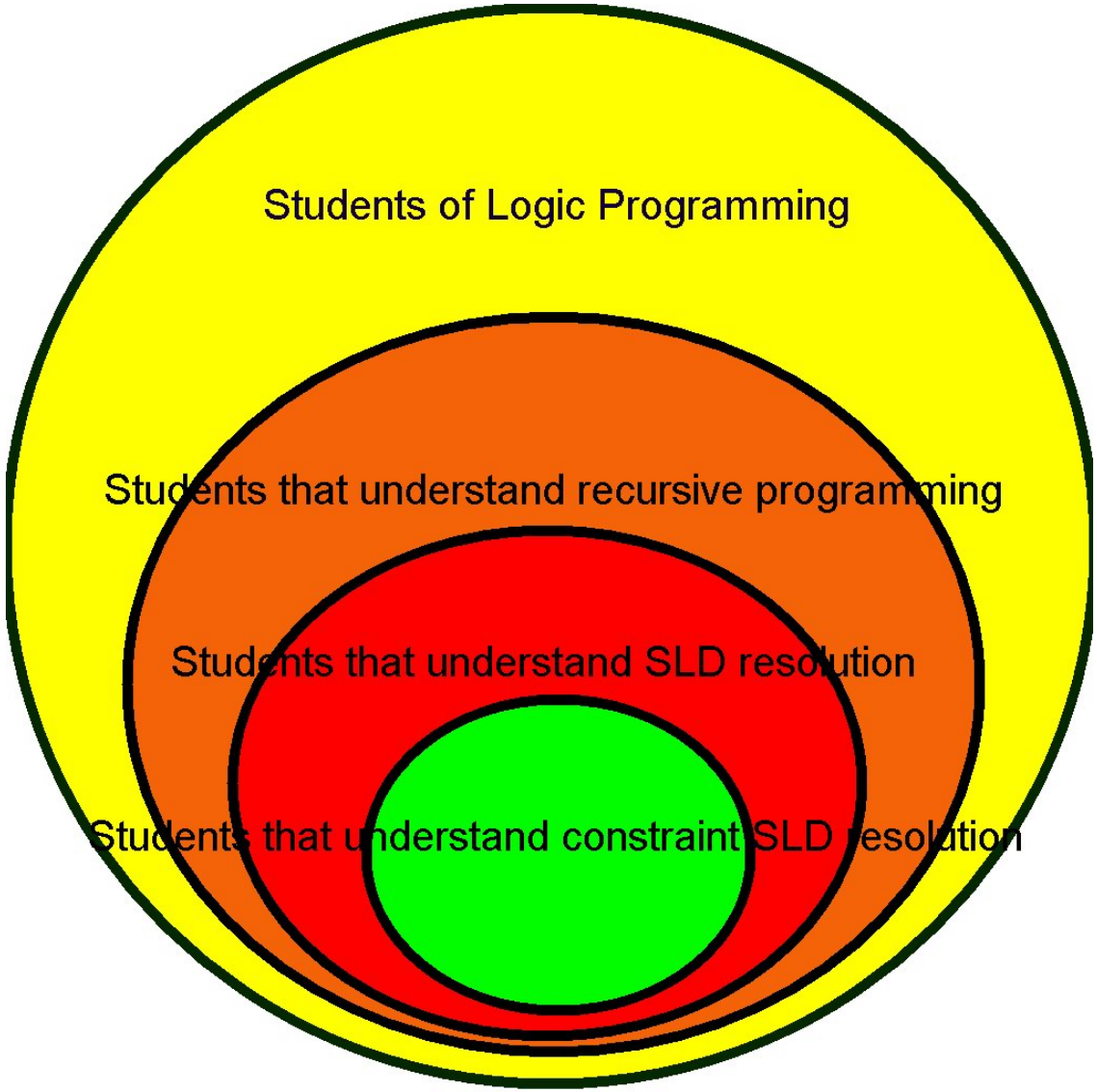
Introduction



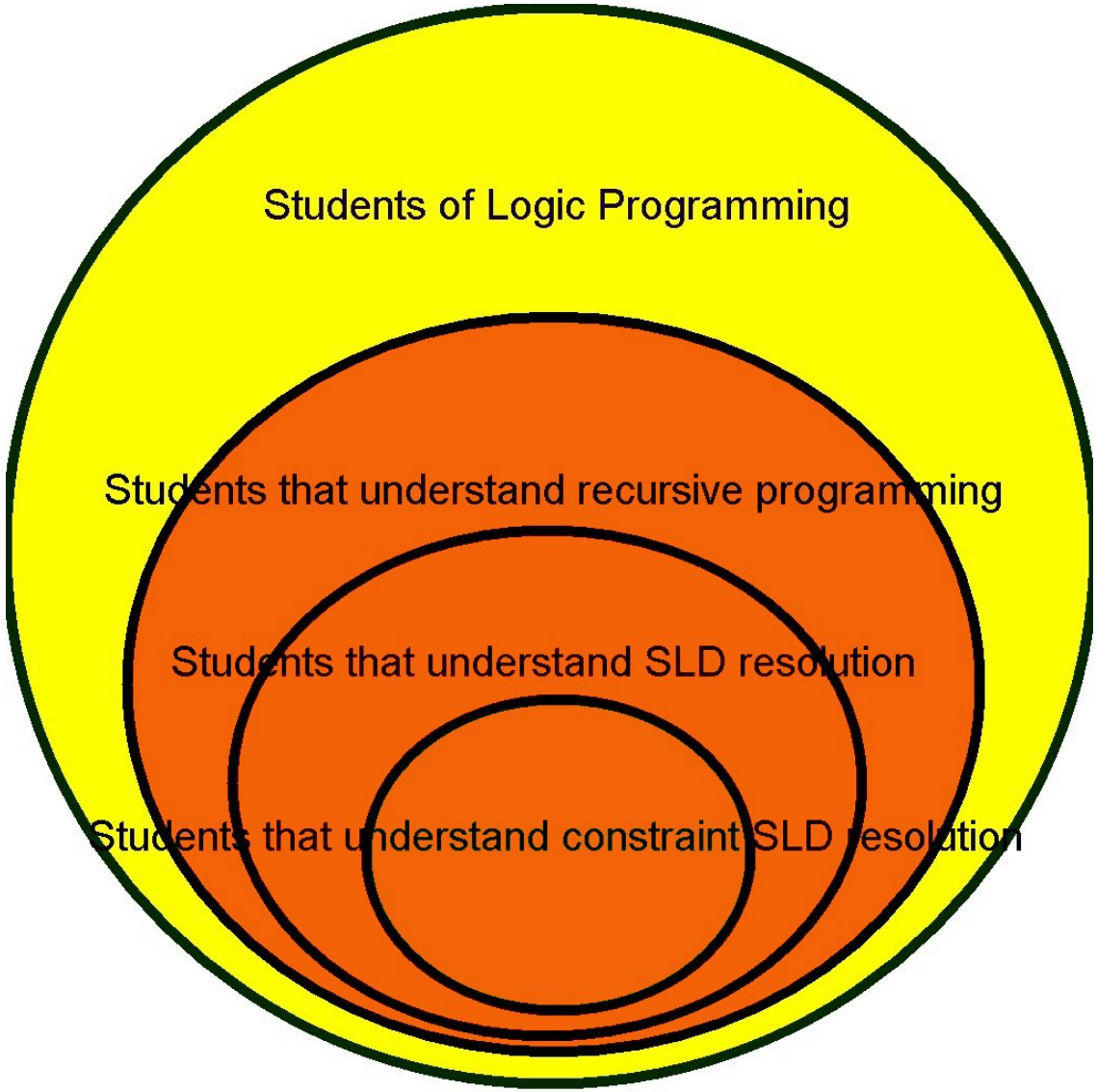
Introduction



Introduction

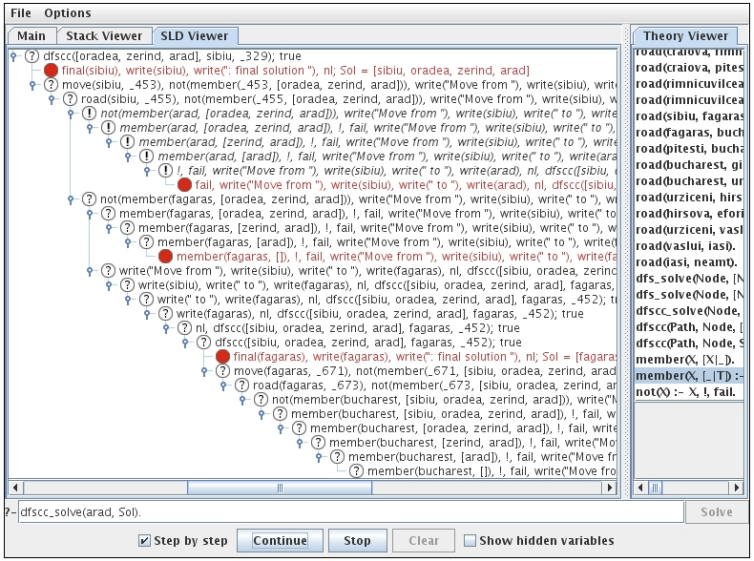


Introduction



Pretty Prolog

- PrettyProlog was developed by a team of the University of Genova, for providing concrete answers to demands raised by Prolog novices [Stalla, Mascardi, Martelli: CILC09].
- Main goal: help to understand the construction and the visit strategy of the SLD tree.
- It was developed from scratch, without reusing existing Prolog implementations, and designed to be simple, modular, and easily expandable.



Pretty Prolog

- PrettyProlog was developed by a team of the University of Genova, for providing concrete answers to demands raised by Prolog novices [Stalla, Mascardi, Martelli: CILC09].
- Main goal: help to understand the construction and the visit strategy of the SLD tree.
- It was developed from scratch, without reusing existing Prolog implementations, and designed to be simple, modular, and easily expandable.

- In this work we have in fact expanded Pretty Prolog to deal with Constraint Logic Programming on finite domains
- Propagation algorithms have been implemented. Differences between arc/bounds consistency are made (as much as possible) explicit

CLP basics

Syntax

- A first-order language $\langle \Pi, \mathcal{F}, \mathcal{V} \rangle$ is partitioned in the two sets Π_C and Π_P ($\Pi = \Pi_C \cup \Pi_P$ and $\Pi_C \cap \Pi_P = \emptyset$)
- Π_P is the set of program defined predicate symbols
- Π_C is the set of constraints predicate symbols (assumed to contain "$=$").
- Similarly, $\mathcal{F}$ is partitioned into $\mathcal{F}_C$ and $\mathcal{F}_P$.

CLP(FD) basics

Syntax

- We focus on CLP on finite domains CLP(FD):
- $\mathcal{F}_C$ contains the binary arithmetic function symbols $+, -, *, /, \texttt{mod}$ etc. as well as a constant symbol for any integer number
- Π_C contains $\leq, <$, etc.
- `false` is assumed to be a special predicate in Π_P which has no rules defining it.
- `domain` and `labeling` are assumed to be in Π_C

CLP(FD) basics

Syntax

- Usual notions for program/constraint atom/literals.
- A *primitive constraint* is a constraint atom or its negation and a *constraint* is a conjunction of primitive constraints (`true` denotes the empty conjunction)
- A *goal CLP* is of the form $\leftarrow \bar{B}$, where $\bar{B}$ is a conjunction of program atoms and primitive constraints.
- A *CLP rule* is of the form $A \leftarrow \bar{B}$ where A is a program atom and $\leftarrow \bar{B}$ is a CLP goal.
- A *CLP program* is a set of CLP rules.

CLP(FD) basics

Syntax

- Usual notions for program/constraint atom/literals.
- A *primitive constraint* is a constraint atom or its negation and a *constraint* is a conjunction of primitive constraints (`true` denotes the empty conjunction)
- A *goal CLP* is of the form $\leftarrow \bar{B}$, where $\bar{B}$ is a conjunction of program atoms and primitive constraints.
- A *CLP rule* is of the form $A \leftarrow \bar{B}$ where A is a program atom and $\leftarrow \bar{B}$ is a CLP goal.
- A *CLP program* is a set of CLP rules.
- Operationally, conjunctions and sets are just lists

CLP(FD) basics

Semantics of a constraint

- Semantically, a constraint C on n variables $X_1, \ldots, X_n$ with domains $D_1, \ldots, D_n$, respectively, is a relation on $D_1 \times \cdots \times D_n$.
- In CLP(FD) D_i's are finite subsets of $\mathbb{Z}$
- A *solution* for C is a mapping $[X_1/d_1, \ldots, X_n/d_n]$ such that $\langle d_1, \ldots, d_n \rangle \in C$.
- If there are no solutions, then C is inconsistent.

CLP(FD) basics

Semantics of a constraint

- Semantically, a constraint C on n variables $X_1, \ldots, X_n$ with domains $D_1, \ldots, D_n$, respectively, is a relation on $D_1 \times \cdots \times D_n$.
- In CLP(FD) D_i's are finite subsets of $\mathbb{Z}$
- A *solution* for C is a mapping $[X_1/d_1, \ldots, X_n/d_n]$ such that $\langle d_1, \ldots, d_n \rangle \in C$.
- If there are no solutions, then C is inconsistent.
- The intended semantics of $\mathcal{F}_C$ symbols (in this case, the arithmetical functions on integer numbers) is fulfilled. This allows to evaluate, e.g., $X + 1 \leq Y * Y - 5 * Z$

CLP(FD) basics

Operational semantics

- It is parametric on the function `solve` that given a constraint C should detect whether C is satisfiable (consistent) in the constraint domain.
- During its computation, `solve`(C) might rewrite C to an equivalent simplified constraint.
- In practice, for complexity reasons, `solve` is an incomplete procedure, in the sense that instead of verifying consistency of the (entire) constraints, acts locally in each primitive constraint, removing some values in domains that cannot belong to any solution until a local property is satisfied.
- This phase is also called Constraint propagation

CLP(FD) basics

Why `solve` is incomplete?

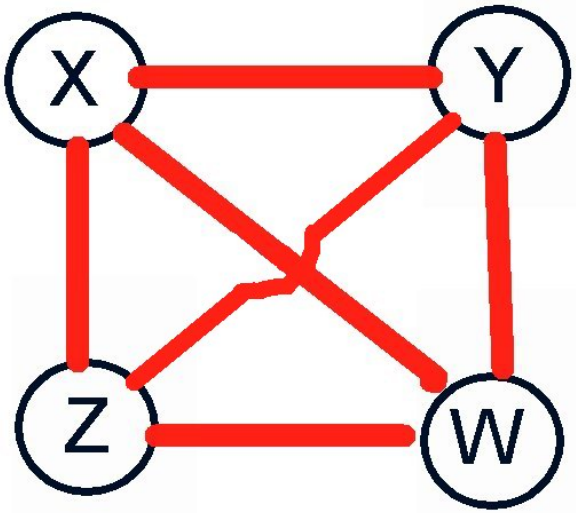
- Let us consider the constraint $X \neq Y, X \neq W, X \neq Z, Y \neq W, Y \neq Z, W \neq Z$, where $D_X = D_Y = D_Z = D_W = \{0, 1, 2\}$.
- Although it is inconsistent, default options in Prolog implementations are such that it is left unaltered by `solve` and, therefore, inconsistency is not detected.

CLP(FD) basics

Why `solve` is incomplete?

- Let us consider the constraint $X \neq Y, X \neq W, X \neq Z, Y \neq W, Y \neq Z, W \neq Z$, where $D_X = D_Y = D_Z = D_W = \{0, 1, 2\}$.
- Although it is inconsistent, default options in Prolog implementations are such that it is left unaltered by `solve` and, therefore, inconsistency is not detected.

It is the encoding of the 3-coloring problem of a graph
Checking consistency of this class of constraints is therefore NP-hard and a fast propagation algorithm can not check it (unless P=NP).



CLP(FD) basics

State

- A *state* is a pair $\langle G \mid C \rangle$ where G is a CLP goal and C is a constraint (also known as the *constraint store*).
- A state $\langle G \mid C \rangle$ is said to be:
  - *successful* if $G = \texttt{true}$ and $\texttt{solve}(C) \neq \texttt{false}$.
  - *failing* if either $\texttt{solve}(C) = \texttt{false}$ or there are no clauses in P with the same predicate of the head of the selected atom in G.
  - *unsolved* if $G \neq \texttt{true}$ and it is not failing.

CLP(FD) basics

Derivation Step

Let $\langle G_1 \mid C_1 \rangle$ be an unsolved state, where $G_1 = \leftarrow L_1, \ldots, L_m$, and P a program. A *CLP-derivation step* $\langle G_1 \mid C_1 \rangle \Rightarrow \langle G_2 \mid C_2 \rangle$ is defined as follows:

- Let L_i be the selected literal in G_1 (let us assume it is L_1—as it happens in the implementation).
- Then $\langle G_2 \mid C_2 \rangle$ is obtained from S and P in one of the following ways:
  - L_1 is a primitive constraint, $C_2 = L_1 \wedge C_1$. If $\texttt{solve}(C_2) = \texttt{false}$, then $G_2 = \leftarrow \texttt{false}$, otherwise $G_2 = \leftarrow L_2, \ldots, L_n$.
  - If $L_1 = p(t_1, \ldots, t_n)$ is a program atom, and $p(s_1, \ldots, s_n) \leftarrow \bar{B}$ is a renaming of a clause of P then $G_2 = \leftarrow t_1 = s_1, \ldots, t_n = s_n, \bar{B}, L_2, \ldots, L_n$ and $C_2 = C_1$.

CLP(FD) basics

Derivation Step

Let $\langle G_1 \mid C_1 \rangle$ be an unsolved state, where $G_1 = \leftarrow L_1, \ldots, L_m$, and P a program. A *CLP-derivation step* $\langle G_1 \mid C_1 \rangle \Rightarrow \langle G_2 \mid C_2 \rangle$ is defined as follows:

- Let L_i be the selected literal in G_1 (let us assume it is L_1—as it happens in the implementation).
- Then $\langle G_2 \mid C_2 \rangle$ is obtained from S and P in one of the following ways:
  - L_1 is a primitive constraint, $C_2 = L_1 \wedge C_1$. If $\texttt{solve}(C_2) = \texttt{false}$, then $G_2 = \leftarrow \texttt{false}$, otherwise $G_2 = \leftarrow L_2, \ldots, L_n$.
  - If $L_1 = p(t_1, \ldots, t_n)$ is a program atom, and $p(s_1, \ldots, s_n) \leftarrow \bar{B}$ is a renaming of a clause of P then $G_2 = \leftarrow t_1 = s_1, \ldots, t_n = s_n, \bar{B}, L_2, \ldots, L_n$ and $C_2 = C_1$.

This is not what it is implemented by the various Prolog systems!!!

CLP(FD) basics

State

- A *derivation* for a state S_0 in P is a maximal sequence of derivations such that $S_0 \Rightarrow S_1 \Rightarrow \cdots$. A derivation for a goal G is a derivation for the state $\langle G \mid \texttt{true} \rangle$.
- A finite *derivation* $S_0 \Rightarrow \cdots \Rightarrow S_n$ is said *successful* (resp. *failing*) if S_n is a successful (resp., failing) state.
- In the case of a successful derivation the computed answer is the projection of the constraint store of S_n on the variables in S_0.
- A simplification, based on `solve`, is usually employed to make the output readable.

CLP(FD) basics

Domain

- The semantics of `domain`$([V_1, \ldots, V_n], a, b)$ is that of assigning the domain $D_{V_i} = \{a, a+1, \ldots, b\}$ to V_i's or to update D_{V_i} with $D_{V_i} \cap \{a, a+1, \ldots, b\}$
- If $D_{V_i} = \emptyset$ then `false`

CLP(FD) basics

Labeling

- The semantics of `labeling`($[V_1, \ldots, V_n]$) is to look for an instantiation for $V_1, \ldots, V_n$ that "satisfy" the current state.
- If there are none of them then `false`

CLP(FD) basics

Arc Consistency

A primitive constraint c on the variables $X_1, \ldots, X_n$ is *arc consistent* (hyper arc consistent if $n > 2$) if:

for all $i \in \{1, \ldots, n\}$
  for all $d_i \in D_i$
    exist $d_1 \in D_1$
      ...
        exist $d_{i-1} \in D_{i-1}$
          exist $d_{i+1} \in D_{i+1}$
            ...
              exist $d_n \in D_n$ s.t.
                $[X_1/d_1, \ldots, X_n/d_n]$ is a solution of c.

CLP(FD) basics

Bounds Consistency

We use the same notion used by major Prolog implementations, a.k.a. *interval consistency*.

A primitive constraint c on the variables $X_1, \ldots, X_n$ is *bounds consistent* (hyper bounds consistent if $n > 2$) if:

for all $i \in \{1, \ldots, n\}$

for all $d_i \in \{\min D_i, \max D_i\}$ (the two interval bounds)

exist $d_1 \in \min D_1 .. \max D_1$ (not necessarily in D_1 !!!)

$\ldots$

exist $d_{i-1} \in \min D_{i-1} .. \max D_{i-1}$

exist $d_{i+1} \in \min D_{i+1} .. \max D_{i+1}$

$\ldots$

exist $d_n \in \min D_n .. \max D_n$ s.t.

$[X_1/d_1, \ldots, X_n/d_n]$ is a solution of c.

CLP(FD) basics

Bounds Consistency

We use the same notion used by major Prolog implementations, a.k.a. *interval consistency*.

A primitive constraint c on the variables $X_1, \ldots, X_n$ is *bounds consistent* (hyper bounds consistent if $n > 2$) if:

for all $i \in \{1, \ldots, n\}$
  for all $d_i \in \{\min D_i, \max D_i\}$ (the two interval bounds)
    exist $d_1 \in \min D_1 .. \max D_1$ (not necessarily in D_1 !!!)
      ...
        exist $d_{i-1} \in \min D_{i-1} .. \max D_{i-1}$
          exist $d_{i+1} \in \min D_{i+1} .. \max D_{i+1}$
            ...
              exist $d_n \in \min D_n .. \max D_n$ s.t.
                $[X_1/d_1, \ldots, X_n/d_n]$ is a solution of c.

The constraint $2X = Y$ where: $D_X = \{0, 1\}, D_Y = \{0, 1, 2\}$ is bounds consistent but not arc consistent.

System Demo

. . . Speriamo funzioni . . .

Conclusions

- We have proved that it is possible to produce Tesi di laurea triennali with some usefulness (Alessio Stalla, 2009; Davide Zanucco, 2010)

Conclusions

- We have proved that it is possible to produce Tesi di laurea triennali with some usefulness (Alessio Stalla, 2009; Davide Zanucco, 2010)
- We have extended Pretty Prolog so as to deal with constraints on finite domains
- It is a wip. Anyway, the .jar is available on the web. Java sources are available on demand
- We hope someone of you will find it interesting and use it in your courses (we'll be glad to add features on demand—e.g. the choice of the granularity of viewpoints)