A Specialization Technique for Deriving Deterministic Constraint Logic Programs and Its Application to Pattern Matching

Fabio Fioravanti
Maurizio Proietti

IASI-CNR
Viale Manzoni 30
00185 Rome, Italy

Alberto Pettorossi

DISP - Università di Roma Tor Vergata
Via del Politecnico 1
00133 Rome, Italy

AGP'2002 - Madrid, Spain **September 2002**

Program Specialization :

- program improvement when knowing the context of use

Jones et al.

e.g. sorting when ordered --> identity or reverse

- Partial Evalution technique

Gallagher
Leuschel et al.
De Schreye et al.
Jones et al.
Lloyd-Shepherdson
Glück . . .

Rules + Strategies approach

Burstall-Darlington
Tamaki-Sato

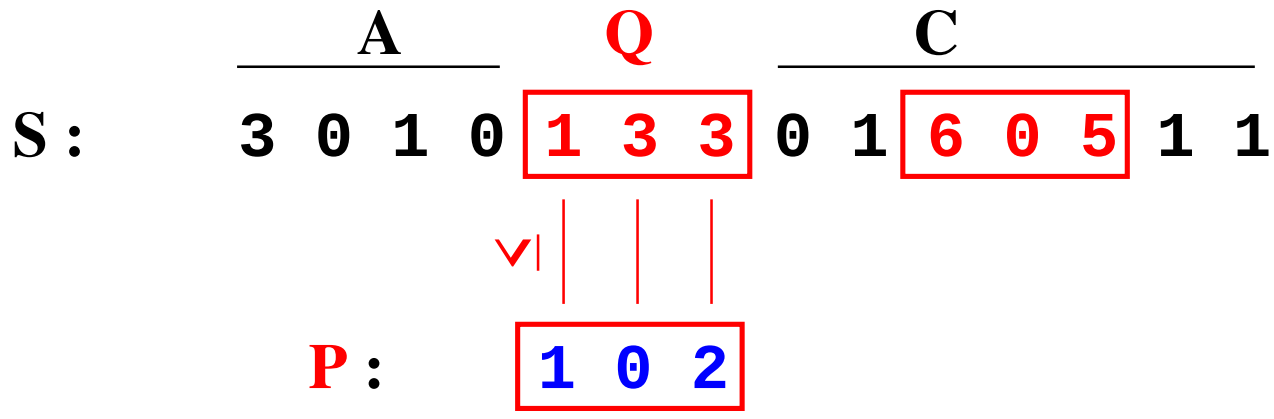
- we make programs more deterministic
by using the clause splitting rule

Leq_Match 1

1. leq_match(P,S) ← append(B,C,S) ∧ append(A,Q,B) ∧ leq(P,Q)

2. append([ ], Ys, Ys) ←
3. append([X|Xs], Ys, [X|Zs]) ← append(Xs, Ys, Zs)

4. leq([ ], [ ]) ←
5. leq([X|Xs], [Y|Ys]) ← X ≤ Y ∧ leq(Xs,Ys)



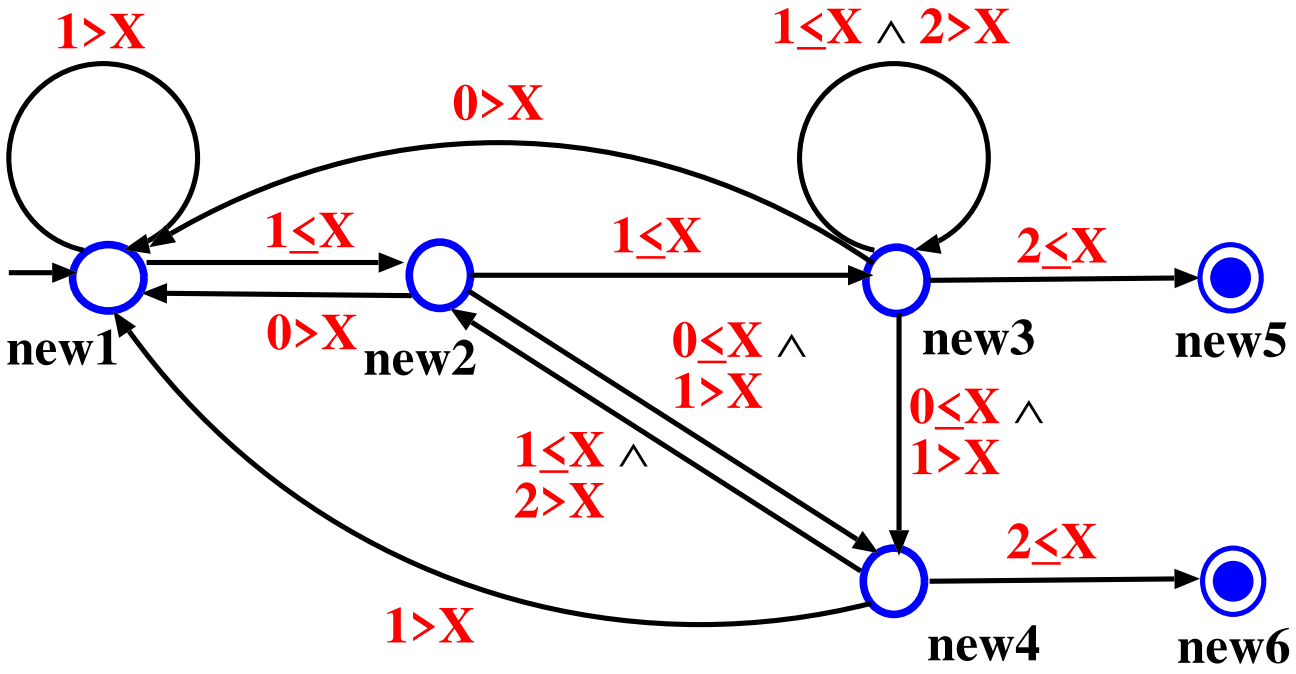
6. $leq\_match_{sp}(S)$ ← leq_match([1,0,2],S)

Leq_Match 2

by specializing w.r.t. the goal leq_match([1,0,2], S)
we get the deterministic program:

9. $leq\_match_{sp}(S) \leftarrow new1(S)$
16. $new1([X|Xs]) \leftarrow 1 \leq X \wedge new2(Xs)$
17. $new1([X|Xs]) \leftarrow 1 > X \wedge new1(Xs)$
$new2([X|Xs]) \leftarrow 1 \leq X \wedge new3(Xs)$
$new2([X|Xs]) \leftarrow 0 \leq X \wedge 1 > X \wedge new4(Xs)$
$new2([X|Xs]) \leftarrow 0 > X \wedge new1(Xs)$
$new3([X|Xs]) \leftarrow 2 \leq X \wedge new5(Xs)$
$new3([X|Xs]) \leftarrow 1 \leq X \wedge 2 > X \wedge new3(Xs)$
$new3([X|Xs]) \leftarrow 0 \leq X \wedge 1 > X \wedge new4(Xs)$
$new3([X|Xs]) \leftarrow 0 > X \wedge new1(Xs)$
$new4([X|Xs]) \leftarrow 2 \leq X \wedge new6(Xs)$
$new4([X|Xs]) \leftarrow 1 \leq X \wedge 2 > X \wedge new2(Xs)$
$new4([X|Xs]) \leftarrow 1 > X \wedge new1(Xs)$
$new5(Xs) \leftarrow$
$new6(Xs) \leftarrow$

Leq_Match 3



mutual exclusive clauses

--> deterministic automaton

Basic Ideas 1

- **increase of efficiency by increase of determinism**

Basic Ideas 2

- generation of mutually exclusive clauses :

clauses : $H_1 \leftarrow c_1 \wedge G_1$, $H_2 \leftarrow c_2 \wedge G_2$ goal : A

if both $c_1 \wedge A = H_1$ are satisfiable
$c_2 \wedge A = H_2$

then split :

$H_1 \leftarrow c_1 \wedge G_1$
$H_2 \leftarrow c_1 \wedge c_2 \wedge G_2$
$H_2 \leftarrow \neg c_1 \wedge c_2 \wedge G_2$

$\Rightarrow$

mutually exclusive packets

$H_1 \leftarrow c_1 \wedge G_1$ — multiple folding → $\gamma 1$
$H_2 \leftarrow c_1 \wedge c_2 \wedge G_2$

$H_2 \leftarrow \neg c_1 \wedge c_2 \wedge G_2$ → $\gamma 2$

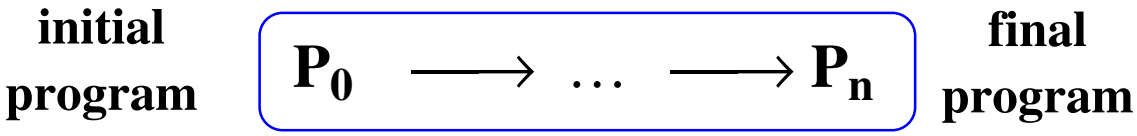
Basic Ideas 3

- **mutual exclusive clauses imply determinism**

  if variables are sufficiently instantiated by c_1 and c_2 :

  mode analysis needed.

Rules + Strategies for Specialization



$$lm(P_0, D) = \ldots = lm(P_n, D)$$

Each rule transforms a program P_i into a new program P_{i+1}

Each rule application preserves the least *D*-model.

Least *D*-model

- An interpretation *D* for the constraints
  - *true* is interpreted as the whole carrier
  - *false* is interpreted as the empty set
  - = is interpreted as the identity over the carrier
  - each *n*-ary constraint is interpreted as a subset of D^n
- A *D*-model of a CLP program P is an interpretation *I* which extends *D* such that
  - each *n*-ary predicate is assigned a subset of D^n
  - $I \models \forall(\mathrm{P})$
- The least *D*-model of a CLP program P, denoted $lm(\mathrm{P}, D)$, is the minimal *D*-model.

Transformation Rules

R1. Definition
R2. Unfolding
R3. Folding
R4. Clause Removal
R5. Constraint Replacement
R6-R7. Clause Fusion and Clause Splitting

extension of rules of :
- **Bensaou-Guessarian**
- **Etalle-Gabbrielli**
- **Gergatsoulis-Katzouraki**
- **Tamaki-Sato**

R1. Definition

Introduce the clauses

$$\delta_1 : \quad newp(X_1, ..., X_h) \leftarrow c_1 \wedge G_1$$
$$\ldots$$
$$\delta_m : \quad newp(X_1, ..., X_h) \leftarrow c_m \wedge G_m$$

where

- **newp is a new predicate symbol**
- $\{X_1, ..., X_h\} \subseteq FV(c_1 \wedge G_1, ..., c_m \wedge G_m)$
- **the predicates occurring in $G_1, ..., G_m$ occur also in P_0**

$Defs_n$: all definitions introduced until P_n

R2. Unfolding

By unfolding $\gamma: \; H \leftarrow c \wedge A \wedge G$

w.r.t. A we get the clauses

$$\Gamma: \; \{ \; H \leftarrow \overbrace{c \wedge A = H_1 \wedge c_1}^{\text{satisfiable}} \wedge G_1 \wedge G$$

for every clause $H_1 \leftarrow c_1 \wedge G_1$ in P_k }

$$P_{k+1} = (P_k - \{\gamma\}) \cup \Gamma$$

R3. Folding

Given the clauses

$$\gamma_1 : \; H \leftarrow c \wedge c_1\vartheta \wedge G_1\vartheta \wedge G$$
$$\ldots$$
$$\gamma_m : \; H \leftarrow c \wedge c_m\vartheta \wedge G_m\vartheta \wedge G$$

and the definitions

$$\delta_1 : \; newp(X_1, ..., X_h) \leftarrow c_1 \wedge G_1$$
$$\ldots$$
$$\delta_m : \; newp(X_1, ..., X_h) \leftarrow c_m \wedge G_m$$

by folding $\gamma_1, ..., \gamma_m$ we get the clause

$$\eta : \; H \leftarrow c \wedge newp(X_1, ..., X_h)\,\vartheta \wedge G$$

$$P_{k+1} = (P_k - \{\gamma_1, ..., \gamma_m\}) \cup \{\eta\}$$

R4. Clause Removal

- **unsatisfiable constraint**

$D \models \neg\exists(c)$

γ : $H \leftarrow c \wedge G$

- **subsumed clause**

$D \models \forall(c_1 \rightarrow \exists Z c_2)$

with $Z = FV(c_2) - vars(H, G_2)$

γ : $(H \leftarrow c_1 \wedge G_1 \wedge G_2)\vartheta$

δ : $H \leftarrow c_2 \wedge G_2$

R5. Constraint Replacement

A clause $\gamma_1 : \; H \leftarrow c_1 \wedge G$

A new clause $\gamma_2 : \; H \leftarrow c_2 \wedge G$

if $D \models \forall(\exists Y c_1 \leftrightarrow \exists Z c_2)$

with $Y = FV(c_1) - vars(H,G)$ and $Z = FV(c_2) - vars(H,G)$

R6-R7. Clause Splitting and Clause Fusion

One clause

γ : $H \leftarrow (c \vee d) \wedge G$

splitting ↓ ↑ fusion

Two clauses

γ_1 : $H \leftarrow c \wedge G$

γ_2 : $H \leftarrow d \wedge G$

In particular:

true $\wedge$ ...

↓ splitting

$c \wedge$ ...

$\neg c \wedge$ ...

Correctness of the Rules

Let $P_0, ..., P_n$ be transformation sequence.

If whenever clauses $\gamma_1, ..., \gamma_m$ are folded

using the definitions $\delta_1, ..., \delta_m$

either every $\delta_1, ..., \delta_m$ has been unfolded during $P_0, ..., P_n$

or the head predicate of $\delta_1, ..., \delta_m$ does not depends on itself in P_n

then $lm(P_0 \cup Defs_n, D) = lm(P_n, D)$

Operational Semantics

$$c \wedge A \wedge G \;\mapsto_P\; \overbrace{c \wedge A = H_1 \wedge c_1}^{\text{satisfiable}} \wedge G_1 \wedge G$$

iff $H_1 \leftarrow c_1 \wedge G_1$ is a clause in P

solve : a computable function for simplifying constraints

- soundness :

  if $solve(c_1, X) = c_2$ then $D \quad \forall X (\exists Y c_1 \leftrightarrow \exists Z c_2)$

  where $Y = FV(c_1) - X$ and $Z = FV(c_2) - X$

- completeness :

  $solve(c, \emptyset) = true$ iff c is satisfiable, i.e., $D \quad \exists(c)$

  $solve(c, \emptyset) = false$ iff c is unsatisfiable, i.e., $D \quad \neg\exists(c)$

Modes and Determinism 1

- a term t is grounded by a constraint c iff
  for every variable X in t
  there exists at most one value of X s.t. c is satisfiable.

- a mode for a predicate p : p(..., +, ..., ?, ...)
  i j
  + iff the i-th argument is grounded by some c
  ? iff no information of the j-th argument

  an input variable is a variable in a term with +

- a constraint atom $c \wedge p(...)$ satisfies a mode for p
  iff every term in p(...) with + is grounded by c.

Modes and Determinism 2

Given a program P and
a mode M, i.e., a mode for each predicate in P

- program P **satisfies** mode M iff
  for each $c_0 \wedge p(...)$ which satisfies M and
  for each $c \wedge A \wedge G$ such that $c_0 \wedge p(...)$ $\quad c \overset{*}{\underset{P}{\mapsto}} A \wedge G$
  $\underline{c \wedge A}$ satisfies M

$\Downarrow$ **Theorem 1**

- program P is **deterministic** w.r.t. mode M iff
  for each $c_0 \wedge p(...)$ which satisfies M and
  for each $c \wedge A \wedge G$ such that $c_0 \wedge p(...)$ $\quad c \overset{*}{\underset{P}{\mapsto}} A \wedge G,$
  there exists **at most one** clause $H_1 \leftarrow c_1 \wedge G_1$ in P
  such that $\underline{c \wedge A} = H_1 \wedge c_1$ is satisfiable

Modes and Determinism 3

- Given the clauses γ_1 : $p(t_1, \ldots, t_h, u_1, \ldots, u_k) \leftarrow c_1 \wedge G_1$

  γ_2 : $p(v_1, \ldots, v_h, w_1, \ldots, w_k) \leftarrow c_2 \wedge G_2$

  and a mode M $p(+, \ldots, +, ?, \ldots, ?)$

  γ_1 and γ_2 are **mutual exclusive** w.r.t. mode M

  iff D $\models \neg\exists(t_1=v_1 \wedge \ldots \wedge t_h=v_h \wedge c_1 \wedge c_2)$

Theorem 1
If program P which **satisfies** mode M and
the clauses of P are **pairwise mutual exclusive**
then P is **deterministic** w.r.t. M.

Determinization Strategy 1

a program P, a mode M, a constrained atom $c \wedge p(t_1, ..., t_h)$

initial definition δ_{sp} : $p_{sp}(X_1, ..., X_r) \leftarrow c \wedge p(t_1, ..., t_h)$

$Defs := \{\delta_{sp}\}$; $Cls := \{\delta_{sp}\}$; $P_{sp} := \{\}$; $M_{sp} := \{p_{sp}(+, ?, ...)\}$;

if X_1 input variable w.r.t M

while $Cls \neq \{\}$ **do**

(1) **Unfold-Simplify**:
clauses in Cls are unfolded and simplified --> UnfCls

(2) **Partition** :
clauses in UnfCls are partitioned in mutually exclusive packets --> PartCls

(3) **Define-Fold** :
packets in PartCls are folded introducing new definition NewDefs and updating M_{sp} --> FldCls
clauses with constraints only in their bodies --> CFacts

(4) $Defs := Defs \cup NewDefs$; $Cls := NewDefs$; $P_{sp} := P_{sp} \cup CFacts \cup FldCls$

Determinization Strategy 2

program P, mode M.

Given $H \leftarrow c \wedge A \wedge G$ in P, A is a consumer atom iff
for every clause $H_1 \leftarrow c_1 \wedge G_1$ in P one of the following holds:
- G_1 is the empty goal
- $c \wedge A = H_1 \wedge c_1$ is unsatisfiable
- $D \models \forall(c \rightarrow \exists Y(A = H_1))$ with Y are the free variables in $A = H_1$ which are not input variables.

consumer atom

$$H \leftarrow X>3 \wedge \underset{+\ ?}{p(X,Y)} \wedge G$$

a clause in P : $p(4,Y_1) \leftarrow Y_1>0 \wedge q(Y_1)$

because $D \models X>3 \rightarrow \exists Y\, Y_1\, (p(X,Y)=p(4,Y_1))$

Determinization Strategy 3

(1) Unfold-Simplify:

- unfold the leftmost consumer atom

- replace $H \leftarrow c \wedge G$ by $H \leftarrow \mathit{solve}(c, \mathrm{vars}(H,G)) \wedge G$
- eliminate subsumed clauses
- eliminate clauses with unsatisfiable c

(2) Partition :

- generate by clause splitting, constraint replacement, clause removal, and clause fusion a set of packets, each is of the form :

$$H \leftarrow c \wedge d_1 \wedge G_1$$
$$\ldots$$
$$H \leftarrow c \wedge d_m \wedge G_m$$

– clauses belonging to two different packets are mutually esclusive w.r.t. M

Determinization Strategy 4

(3) Define-Fold :

- fold every packet :

$$H \leftarrow c \wedge d_1 \wedge G_1$$
$$\ldots$$
$$H \leftarrow c \wedge d_m \wedge G_m$$

by (possibly) introducing the definition :

$$newp(X_1, ..., X_r) \leftarrow d_1 \wedge G_1$$
$$\ldots$$
$$newp(X_1, ..., X_r) \leftarrow d_m \wedge G_m$$

thereby deriving a single clause:

$$H \leftarrow c \wedge newp(X_1, ..., X_r)$$

Correctness of the Determinization Strategy

Given a program P and a mode M for P. Assume :

- P satisfies mode M

- the Determinization Strategy with clause

δ_{sp} : $p_{sp}(X_1, ..., X_r) \leftarrow c \wedge p(t_1, ..., t_h)$

returns P_{sp} with mode M_{sp}

Then **Correctness**

- for all $\vartheta = \{X_1/d_1, ..., X_r/d_r\}$ with $d_1, ..., d_r$ ground

$lm(P, D) \models (c \wedge p(t_1, ..., t_h))\vartheta$ **iff** $lm(P_{sp}, D) \models (c \wedge p_{sp}(X_1, ..., X_r))\vartheta$

- P_{sp} is deterministic w.r.t. mode M_{sp} **Determinism**

Specializing Leq_Match 1

leq_match(P,S) ← append(B,C,S) ∧ append(A,Q,B) ∧ leq(P,Q)
append([ ], Ys, Ys) ←
append([X|Xs], Ys, [X|Zs]) ← append(Xs, Ys, Zs)
leq([ ], [ ]) ←
leq([X|Xs], [Y|Ys]) ← X ≤ Y ∧ leq(Xs,Ys)

mode M = {leq_match(+,+), append(?,?, +), leq(+,+)}

6. $leq\_match_{sp}(S) \leftarrow leq\_match([1,0,2],S)$ $M_{sp} = \{leq\_match_{sp}(+)\}$

Unfold-Simplify clause 6 :

7. $leq\_match_{sp}(S) \leftarrow append(B,C,S) \wedge append(A,Q,B) \wedge leq([1,0,2],Q)$

This clause is a packet in itself. Define :

8. new1(S) ← append(B,C,S) ∧ append(A,Q,B) ∧ leq([1,0,2],Q)

Fold clause 7: . . .

Specializing Leq_Match 2

Fold clause 7 :

9. $\text{leq\_match}_{sp}(S) \leftarrow \text{new1}(S)$

Defs := {6, 8}; Cls := {8}; P_{sp} := {9}; M_{sp} := {leq_match_{sp}(+), new1(+)};

Unfold-Simplify clause 8 :

10. $\text{new1}([X|Xs]) \leftarrow 1 \leq X \wedge \text{append}(Q,C,Xs) \wedge \text{leq}([0,2],Q)$
11. $\text{new1}([X|Xs]) \leftarrow \text{append}(B,C,Xs) \wedge \text{append}(A,Q,B) \wedge \text{leq}([1,0,2],Q)$

Clauses 10 and 11 are not mutually exclusive w.r.t. M_{sp}. Clause Splitting of clause 11 :

12. $\text{new1}([X|Xs]) \leftarrow 1 \leq X \wedge \text{append}(B,C,Xs) \wedge \text{append}(A,Q,B) \wedge \text{leq}([1,0,2],Q)$
13. $\text{new1}([X|Xs]) \leftarrow X > 1 \wedge \text{append}(B,C,Xs) \wedge \text{append}(A,Q,B) \wedge \text{leq}([1,0,2],Q)$

Two packets of clauses : {10, 12} and {13}.

Specializing Leq_Match 3

Packet {10, 12} :

10. new1([X|Xs]) ← 1≤X ∧ append(Q,C,Xs) ∧ leq([0,2],Q)
12. new1([X|Xs]) ← 1≤X ∧ append(B,C,Xs) ∧ append(A,Q,B) ∧ leq([1,0,2],Q)

Folding packet {10, 12} using Definition {14, 15} :

14. new2(Xs) ← append(Q,C,Xs) ∧ leq([0,2],Q)
15. new2(Xs) ← append(B,C,Xs) ∧ append(A,Q,B) ∧ leq([1,0,2],Q)

We get :

16. new1([X|Xs]) ← 1≤X ∧ new2(Xs)

Folding packet {13} using Definition 8 we get :

17. new1([X|Xs]) ← X>1 ∧ new1(Xs)

Defs := {6,8,14,15}; Cls := {14,15};

P_{sp} := {9,16,17}; M_{sp} := {leq_match_{sp}(+), new1(+), new2(+)};

Leq_Match 2

by specializing w.r.t. the goal leq_match([1,0,2], S)
we get the deterministic program:

9. $leq\_match_{sp}(S) \leftarrow new1(S)$
16. $new1([X|Xs]) \leftarrow 1 \leq X \wedge new2(Xs)$
17. $new1([X|Xs]) \leftarrow 1 > X \wedge new1(Xs)$

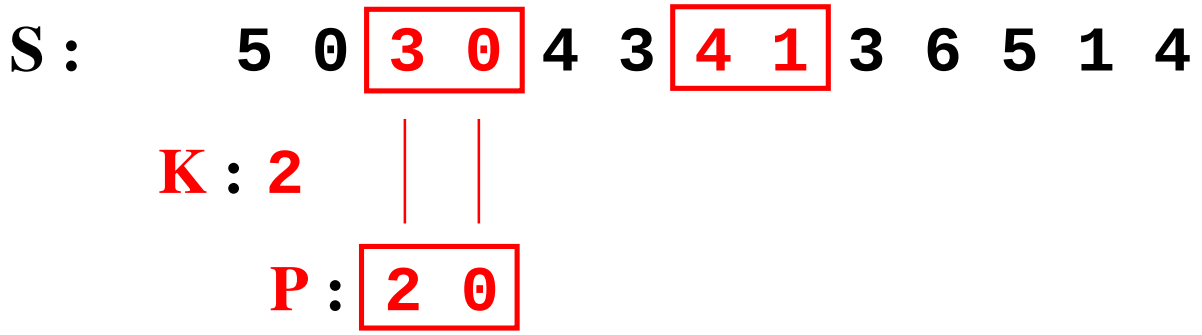
$new2([X|Xs]) \leftarrow 1 \leq X \wedge new3(Xs)$
$new2([X|Xs]) \leftarrow 0 \leq X \wedge 1 > X \wedge new4(Xs)$
$new2([X|Xs]) \leftarrow 0 > X \wedge new1(Xs)$
$new3([X|Xs]) \leftarrow 2 \leq X \wedge new5(Xs)$
$new3([X|Xs]) \leftarrow 1 \leq X \wedge 2 > X \wedge new3(Xs)$
$new3([X|Xs]) \leftarrow 0 \leq X \wedge 1 > X \wedge new4(Xs)$
$new3([X|Xs]) \leftarrow 0 > X \wedge new1(Xs)$
$new4([X|Xs]) \leftarrow 2 \leq X \wedge new6(Xs)$
$new4([X|Xs]) \leftarrow 1 \leq X \wedge 2 > X \wedge new2(Xs)$
$new4([X|Xs]) \leftarrow 1 > X \wedge new1(Xs)$
$new5(Xs) \leftarrow$
$new6(Xs) \leftarrow$

Near_Match 1

near_match(P,K,S) ← app(B,C,S) ∧ app(A,Q,B) ∧ near(P,K,Q)

app([ ], Ys, Ys) ←
app([X|Xs], Ys, [X|Zs]) ← app(Xs, Ys, Zs)

near([ ],K,[ ]) ←
near([X|Xs], K, [Y|Ys]) ← |X-Y| ≤ K ∧ leq(Xs,Ys)

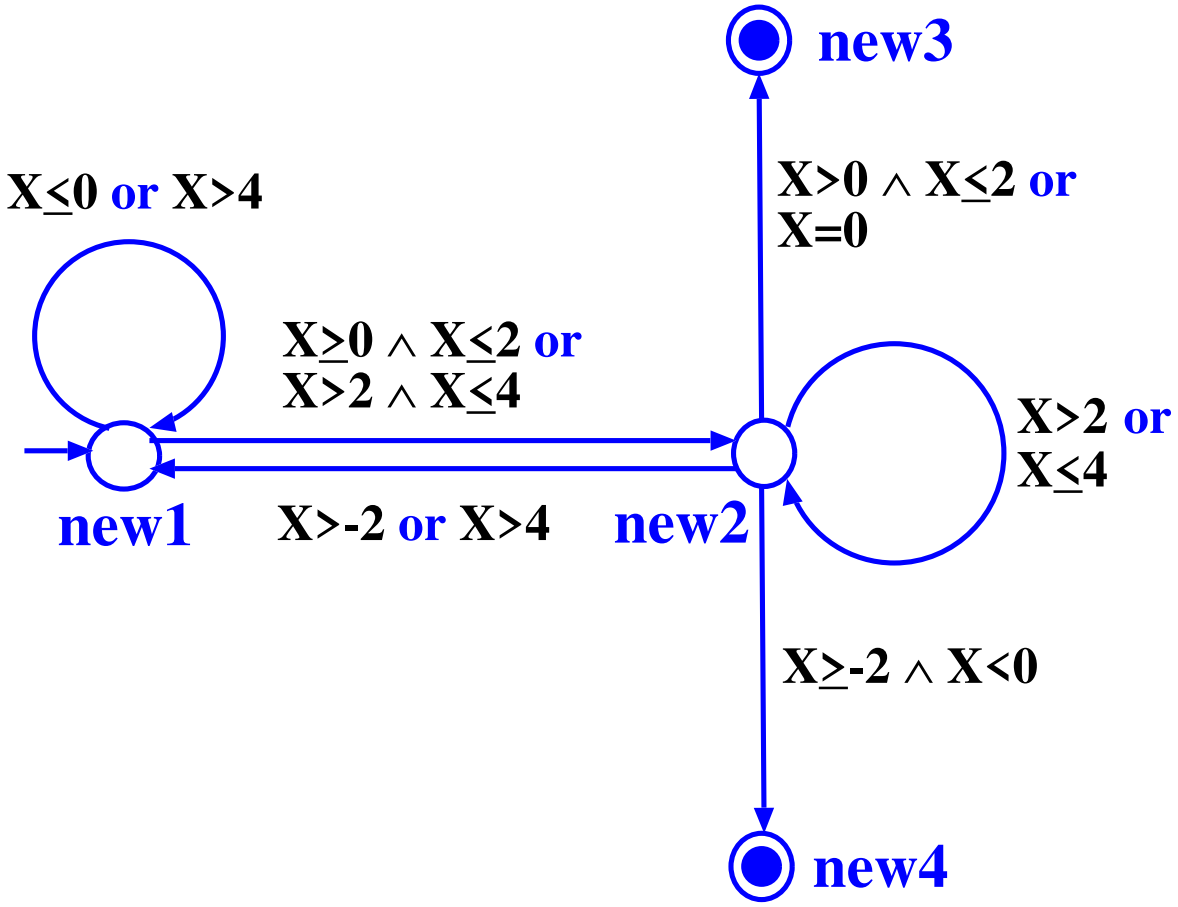


Near_Match 2

by specializing near_match w.r.t. the goal near_match([2,0], 2, S) we get the deterministic program:

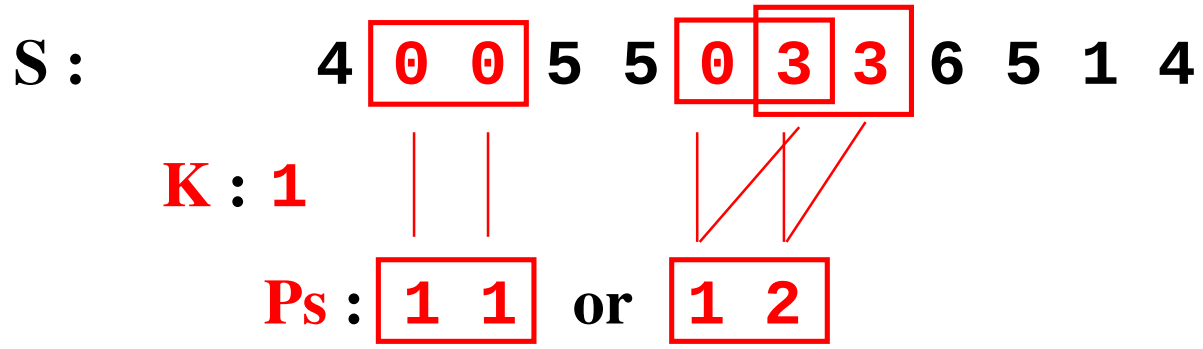
$near\_match_{sp}(S) \leftarrow new1(S)$
$new1([X|Xs]) \leftarrow X \geq 0 \wedge X \leq 2 \wedge new2(Xs)$
$new1([X|Xs]) \leftarrow X \leq 0 \wedge new1(Xs)$
$new1([X|Xs]) \leftarrow X > 2 \wedge X \leq 4 \wedge new2(Xs)$
$new1([X|Xs]) \leftarrow X > 4 \wedge new1(Xs)$
$new2([X|Xs]) \leftarrow X > 2 \wedge X \leq 4 \wedge new2(Xs)$
$new2([X|Xs]) \leftarrow X > 0 \wedge X \leq 2 \wedge new3(Xs)$
$new2([X|Xs]) \leftarrow X = 0 \wedge new3(Xs)$
$new2([X|Xs]) \leftarrow X \geq -2 \wedge X < 0 \wedge new4(Xs)$
$new2([X|Xs]) \leftarrow X < -2 \wedge new1(Xs)$
$new2([X|Xs]) \leftarrow X > 4 \wedge new1(Xs)$
$new3(Xs) \leftarrow$
$new4(Xs) \leftarrow$

Near_Match 3


new3
X≤0 or X>4
X>0 ∧ X≤2 or
X=0
X≥0 ∧ X≤2 or
X>2 ∧ X≤4
X>2 or
X≤4
new1
X>-2 or X>4
new2
X≥-2 ∧ X<0
new4

Near_multimatch

near_mmatch([P|Ps], K, S) ← near_match(P, K, S)
near_mmatch([P|Ps], K, S) ← near_mmatch(Ps, K, S)



Interval_Regular_Expressions 1

in_language(E,S) ← string(S) ∧ accepts(E,S)

accepts(lt(X), [Y]) ← Y<X
accepts(gt(X), [Y]) ← Y>X
accepts(int(X_1, X_2),[Y]) ← X_1<Y ∧ Y<X_2

less than
greater than
within interval

accepts(E_1·E_2, S) ← app(S_1,S_2,S) ∧ accepts(E_1, S_1) ∧ accepts(E_2, S_2)
accepts(E_1+E_2, S) ← accepts(E_1,S)
accepts(E_1+E_2, S) ← accepts(E_2,S)

accepts(E*, [ ]) ←
accepts(E*, S) ← ne_app(S_1,S_2,S) ∧ accepts(E, S_1) ∧ accepts(E*,S_2)

append for nonempty lists

string([ ]) ←
string([X|Xs]) ← string(Xs)

Interval_Regular_Expressions 2

E : int(0,2)* + int(1,3)*

- S1 : 1 2 2 1 2 2 2 1 3 1

in_language(E, S1) holds

- S2 : 1 2 2 1 2 2 2 4 0 1

in_language(E, S2) does not hold

E : gt(1) · gt(1)* + gt(1) · gt(1)* · lt(0)

- S3 : 3 2 4 3 3 2 2 3 3 2

in_language(E, S3) holds

- S4 : 3 2 4 3 3 2 2 3 3 1

in_language(E, S4) does not hold

Speedups

Program	Constrained Atom for Specialization	Speedup
Leq_Match	leq_match([1,0,2], S)	2.7
Near_Match	near_match([2,0], 2, S)	37-56
Near_Match	near_match([2,0,4], 2, S)	46
Near_Mmatch	near_mmatch([[1,1],[1,2]], 1, S)	45
Int_Reg_Expr	in_language(int(0,2)*+int(1,3)*, S)	1.2-7
Int_Reg_Expr	in_language((>1)(>1)*+(>1)(>1)*(<0), S) . .	1.3-320

Future work

- Experiment with other classes of algorithms : scheduling
- Adding a generalization technique to get termination
  for general classes of programs

MAP system available at

`http://www.iasi.rm.cnr.it/~fioravan`