PROGETTO COFINANZIATO

SISTEMI FORMALI PER LA SPECIFICA, L'ANALISI, LA VERIFICA, LA SINTESI E LA TRASFORMAZIONE DI SISTEMI SOFTWARE

FILONE 9:

TECNICHE DI TRASFORMAZIONE E SINTESI

FINAL WORKSHOP

VENICE, ITALY

19-21 January 2000

http://strudel.di.unipi.it/p40/tectra.html
http://www.iasi.rm.cnr.it/~adp/MURST40-Levi-98/Filone9.html

UNITA' PARTECIPANTI

Unita' de L'Aquila

Unita' di Pisa

Unita' di Roma Tor Vergata

Unita' di Udine

Unita' di Venezia

PROGRAM TRANSFORMATION

"rules + strategies" approach

$P_0 \xrightarrow{\text{step 1}} P_1 \xrightarrow{\text{step 2}} P_2 \ \ldots \xrightarrow{\text{step k}} P_k$: a transformation sequence

every step is an application of a transformation rule
every rule application is semantics preserving, i.e. correct

need for strategies to get an efficient final program P_k

- Separation of the correctness and efficiency concerns.
- Structuring program derivations.
- Easy correctness proofs of complex transformations.
- Common framework for program proofs and derivations.

A PROGRAM TRANSFORMATION : LIST REVERSAL

INITIAL PROGRAM : $O(n^2)$

1. rev([ ],[ ]) ←
2. rev([H|T],R) ← rev(T,T1), append(T1,[H],R)
3. append([ ],L,L) ←
4. append([H|T],L,[H|TL]) ← append(T,L,TL)

Definition Introduction (in clause 2, [H] is generalized to the variable Acc):

5. rev_acc(L,Acc,R) ← rev(L,L1), append(L1,Acc,R)

Unfolding clause 5 w.r.t. rev(L,L1):

6. rev_acc([ ],Acc,Acc) ←
7. rev_acc([H|T],Acc,R) ← rev(T,T1), append(T1,[H],L1), append(L1,Acc,R)

Goal Replacement (lemma application):

[Lemma: $M(P_0) \models \forall$ T1, H, Acc, R.
$\exists$ L1. append(T1,[H],L1), append(L1,Acc,R) ↔ append(T1,[H|Acc],R)]

8. rev_acc([H|T],Acc,R) ← rev(T,T1), append(T1,[H|Acc],R)

Folding clause 8 using clause 5:

9. rev_acc([H|T],Acc,R) ← rev_acc(T,[H|Acc],R)

Folding clause 2 using clause 5:

10. rev([H|T],R) ← rev_acc(T,[H],R)

FINAL PROGRAM : $O(n)$

1. rev([ ],[ ]) ←
10. rev([H|T],R) ← rev_acc(L,[H],R)
6. rev_acc([ ],Acc,Acc) ←
8. rev_acc([H|T],Acc,R) ← rev_acc(T,[H|Acc],R)

UNFOLD/FOLD TRANSFORMATION RULES

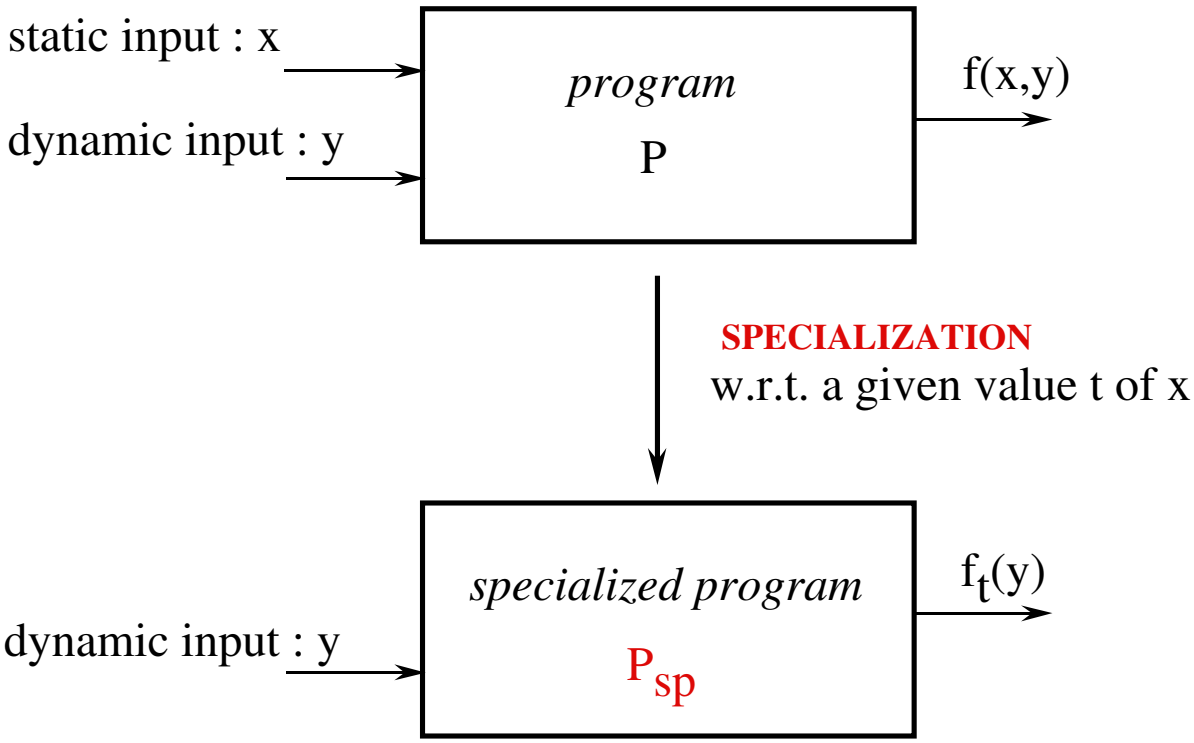
1. *Definition Introduction* DI
2. *Definition Elimination* DE

3. *Unfolding* U
4. *Folding* F

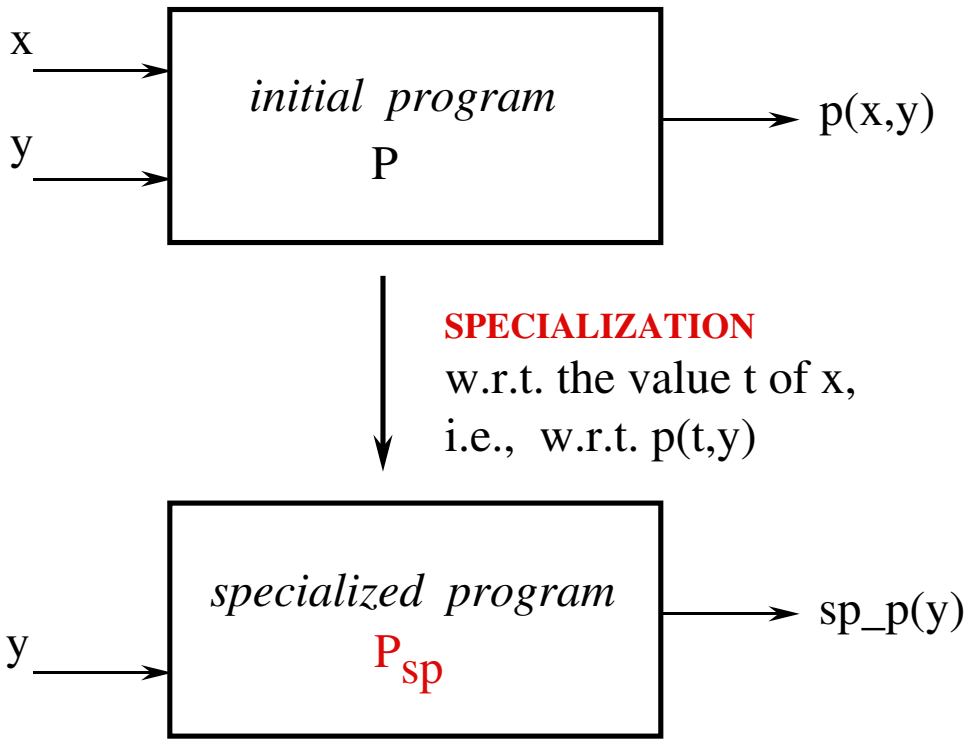
5. *Goal Replacement* GR
6. *Equalities & Tautologies* EQ

collectively called *unfold/fold transformation rules*

PROGRAM SPECIALIZATION



LOGIC PROGRAM SPECIALIZATION



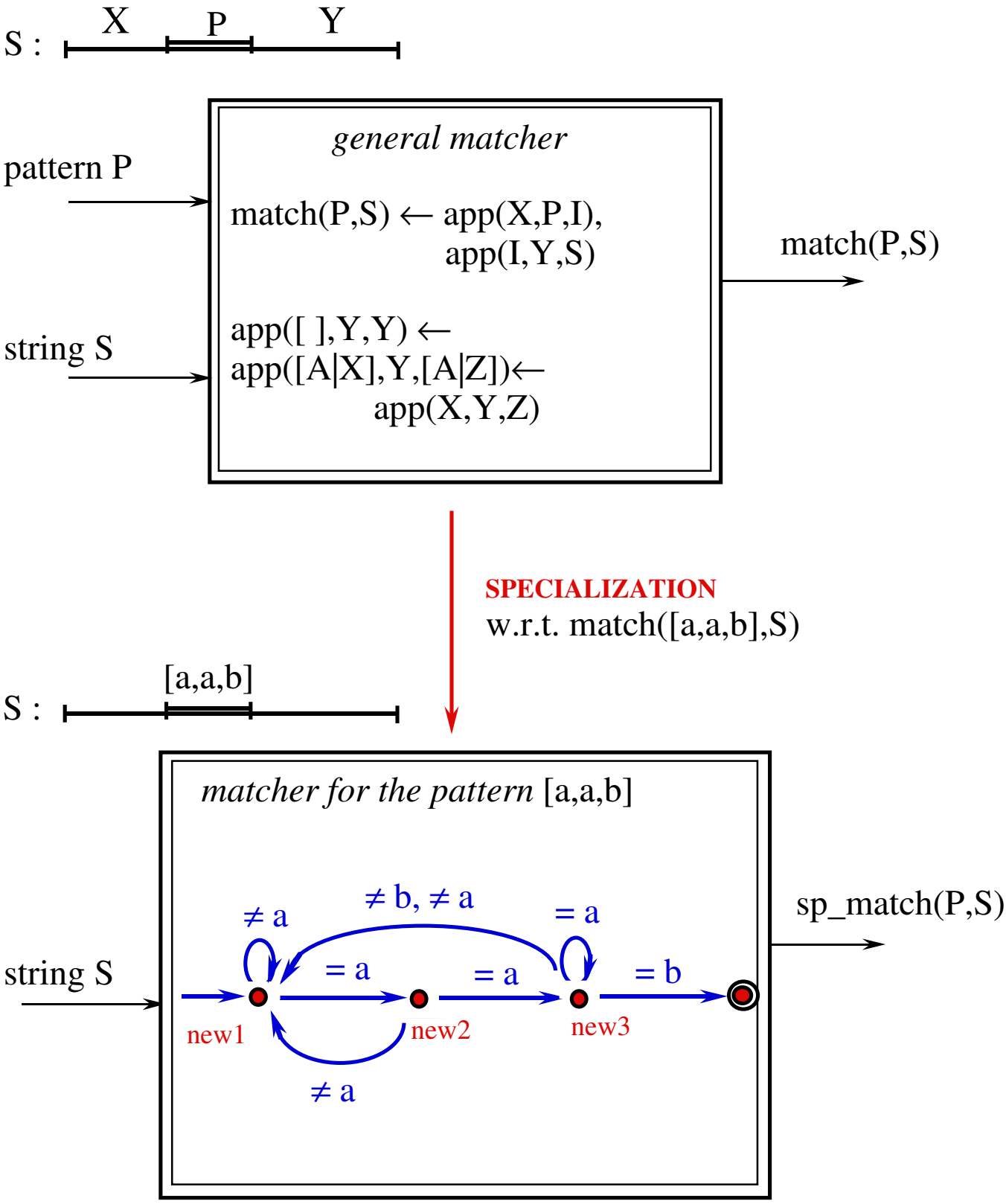
such that

$\forall$ y. $\quad M(P) \models p(t,y) \quad$ *iff* $\quad M(P_{sp}) \models sp_p(y)$

LOGIC PROGRAM SPECIALIZATION is usually achieved via
PARTIAL DEDUCTION [Lloyd-Shepherdson 91]

Survey in:
Gallagher: *Tutorial on Specialization of Logic Programs*, PEPM 93, ACM Press,1993.

SPECIALIZING A MATCHER

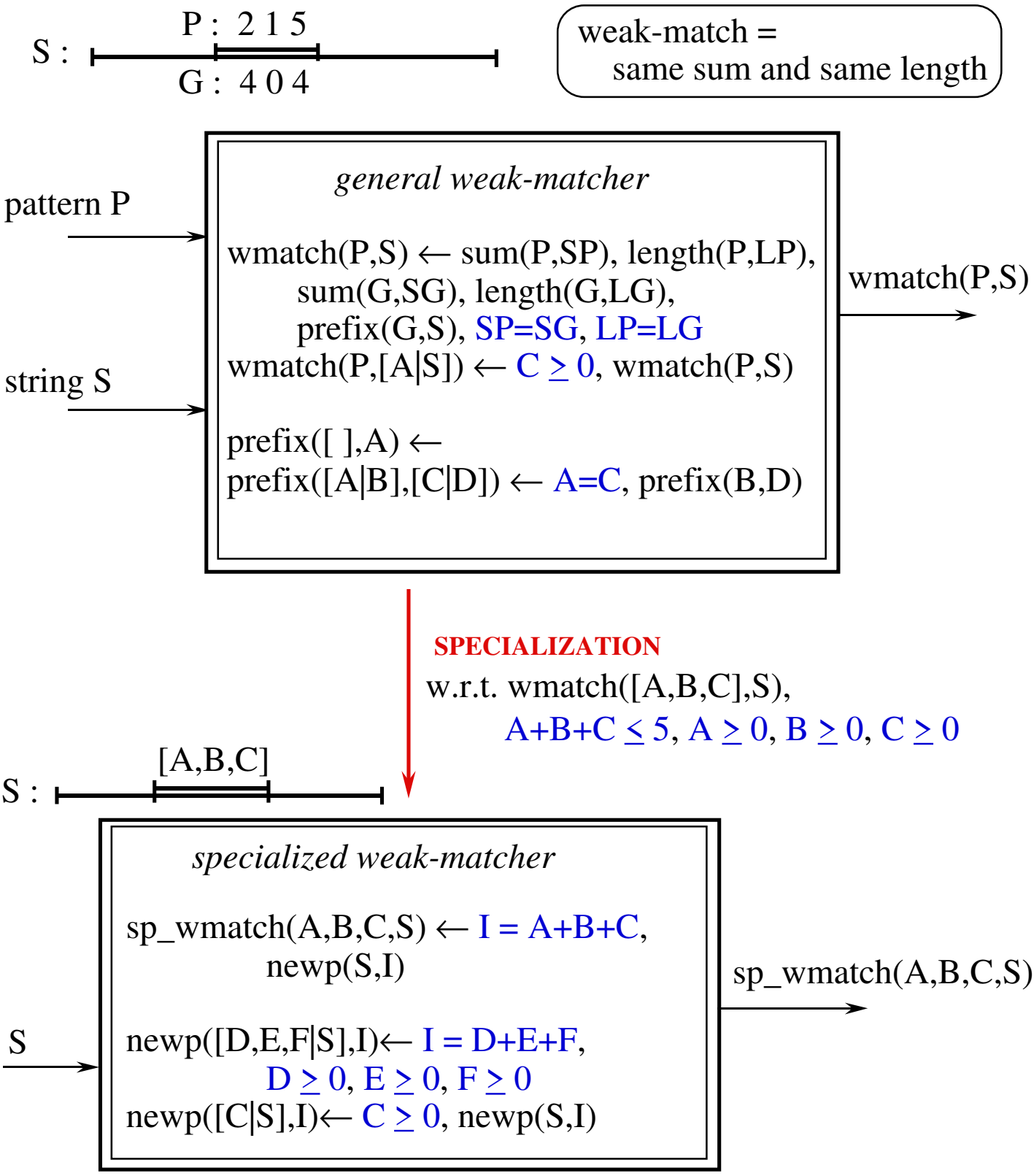


It is a deterministic finite automaton as in the Knuth-Morris-Pratt algorithm.

[see also: Fujita 87, Futamura 87, Glück-Klimov 93, Gallagher 93, etc.]

SPECIALIZING A WEAK-MATCHER

Constraint Logic Programming



PROGRAM SYNTHESIS

Given: a formula φ of the form:

$$\forall X.\ [\exists Y.\ spec(X,Y) \leftrightarrow \exists Z.\ context(X,Z),\ p(X,Z)]$$

and a program P defining spec and context

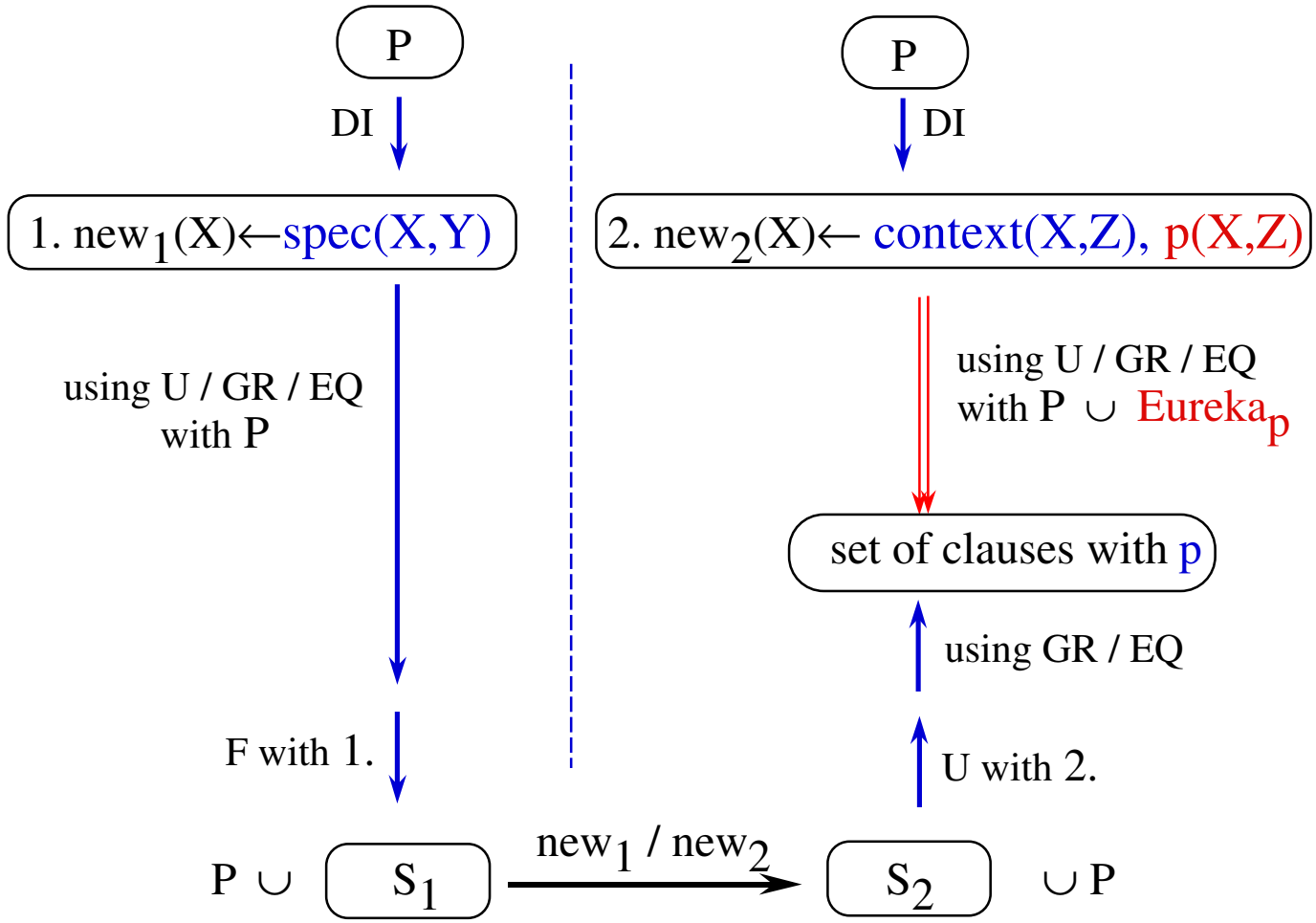
we want to synthezise a program, say $Eureka_p$, defining p, such that:

$$M(P \cup Eureka_p) \models \varphi$$

Synthesis - for avoiding type checking
- for introducing difference lists
- for eliminating left recursion

SYNTHESIS VIA UNFOLD / FOLD PROOFS

Given a definite program P and
a formula φ: ∀X. [∃Y. spec(X,Y) ↔ ∃Z. context(X,Z), p(X,Z)]



U = Unfolding
F = Folding
GR = Goal Replacement
EQ = Equality manipulation

Transformation and Synthesis
= PAPERS =

	Logic Programs	Functional Logic Programs	(Concurrent) Constraint Programs
PROGRAM TRANSFORMATION **620**: *general framework*	**20**: *Review* **30**: *goal replacement* **31-630**: *termination* **32-33**: *queries without failures and queries with success* **619**: *goals as arguments*	**55**: *unfold/fold rules and their correctness* **526**: *transformation system based on narrowing*	**64**: *unfold/fold rules and their correctness for CCP*
PROGRAM SPECIALIZATION	**20**: *Review* **34-35**: *Specializ. wrt call conditions*	**54-57**: *algorithm and system* **328**: *general introduction*	**65**: *usefulness of partial evaluation for CCP* **621**: *rules and strategies for CP*
PROGRAM SYNTHESIS	**21**: *Synthesis via unfold/fold proofs*		

Paper numbers are those of the list in the web page:
http://strudel.di.unipi.it/p40/tectra.html

Transformation and Synthesis
= TEAMS =

	Logic Programs	Functional Logic Programs	(Concurrent) Constraint Programs
PROGRAM TRANSFORMATION Roma 2	Roma 2 Venezia	Udine	L'Aquila Pisa
PROGRAM SPECIALIZATION	Roma 2 Venezia	Udine	Roma 2 Pisa Udine
PROGRAM SYNTHESIS	Roma 2		

[20]

- *Review of rules and strategies* for transforming definite logic programs.

---

[619]

- *Goals as arguments*: a higher-order extension for allowing the use of continuations.

```
flipcheck(X,Y) <- flip(X,Y), check(Y)
flip(l(N), l(N)) <-
flip(t(L,N,R), t(FR,N,FL)) <- flip(L,FL), flip(R,FR)
check(l(N)) <- nat(N)
check(t(L,N,R)) <- nat(N), check(L), check(R)
```

```
flipcheck(X,Y) <- newp(X,Y,G),  G
newp(l(N), l(N), nat(N)) <-
newp(t(L,N,R), t(FR,N,FL), G) <- newp(L,FL,V), newp(R,FR,U),
                                 G = (nat(Y),U,V)
```

---

[620]

- *General framework for proving correctness of transformation rules* when the semantics of the language is given as a set of deductive rules of the form:

$$\frac{\texttt{premise_1} \quad \ldots \quad \texttt{premise_n} \quad \ldots}{\texttt{conclusion}}$$

For instance:

$$\frac{\texttt{2 -> 2} \qquad \texttt{3+1 -> 4}}{\texttt{2+(3+1) -> 6}}$$

It works for functional programs, logic programs, CCS programs, π-calculus, etc.

FUNCTIONAL LOGIC PROGRAMS

Functional Logic Program = { $(t_1 \rightarrow t_2) \Leftarrow C$ }

conditional term rewriting — lhs — rhs — sequence of equations

---

- *Program*:
  1. `even(0) -> true`
  2. `even(s(x)) -> odd(x)`
  3. `odd(x) -> true <= even(x) = false`
  4. `odd(x) -> false <= even(x) = true`

- *Goal*:
  `even(s(y)) = true`

- *Computation via narrowing*:

`even(s(y)) = true`

{2} `odd(y) = true`

{3} `true = true, even(y) = false` ↓ {y/s(x)}

{2} `true = true, odd(x) = false`

{4} `true = true, false = false, even(x) = true` ↓ {x/0}

{1} `true = true, false = false, true = true`

`true, true, true` → `even(s(s(0))) = true`

In general, *any* occurrence of a subterm can be rewritten, not the top one only.

[55, 526] Transformation

- Various unfolding rules.
- Innermost folding rule.
- Transformation based on *needed narrowing*
- Correctness Theorem:
  preservation of the computed answer substitution semantics.
- Study of *composition and tupling strategies*.

[54, 57, 328] Partial Evaluation

- Constructing *resultants*.
- Correctness of partial evaluation based on *independence* and *closedness*
  (as in Lloyd-Shepherdson)
- Specialization algorithm based on *unfolding* and *abstraction*
  (as in Gallagher)
- Local and global control of the partial evaluation
  (as in Leuschel, Martens, De Schreye)

Palindrome example.

The program:

```
palindrome(x) -> true  <=  reverse(x)=x
reverse(x) -> rev(x,nil)
rev(nil,ys) -> ys
rev(x:xs,ys) -> rev(xs,x:ys)
```

specialized wrt the goal:

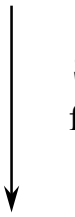
```
palindrome(1:2:x) = y
```

generates the *residual* program:

```
palindrome(1:2:x:xs) -> true  <= rev(xs,x:2:1:nil)=1:2:x:xs
rev(nil,x1:x2:x3:ys) -> x1:x2:x3:ys
rev(x:xs,x1:x2:x3:ys) -> rev(xs,x:x1:x2:x3:ys)
```

LOGIC PROGRAM SPECIALIZATION

- [34, 35] {Pre} program {Post}



SPECIALIZATION wrt an invariant I
for the atom selected during execution

{Pre} $program_{sp}$ {Post}
and
Operational_Sem($program_{sp}$) $\models$ I

- **Example: avoiding type-checking.**

The program:

```
sum(X,0,X) <-
sum(X,s(Y),s(Z)) <- sum(X,Y,Z)
```

Here: `sum(a,s(0),s(a))` succeeds.

specialized wrt: "`sum(u,v,z)` and `z ∈ {0,s(0),...}`"
generates the *residual* program:

```
sum(0,0,0) <-
sum(s(X),0,s(X)) <-
sum(X,s(Y),s(Z)) <- sum(X,Y,Z)
```

Here: `sum(a,s(0),s(a))` fails.

Semantics is not preserved.

- Definition of the operational and fixpoint semantics of the specialized program
- Equivalence of the operational and fixpoint semantics.

TRANSFORMATION OF CONCURRENT CONSTRAINT PROGRAMS

[64]

- Language

  declarations D ::= a sequence of $p(t) \leftarrow A$

  agents $A ::= stop \mid p(t) \mid tell(c) \mid \sum_i ask(c_i) \rightarrow A_i \mid A \parallel A$

  processes $P ::= \langle D, A \rangle$

- Store as set of constraints. Operational Semantics.

- Transformation rules: unfolding
  simplification of guards
  distributivity: $C \parallel (a \rightarrow A) + (b \rightarrow B) = (a \rightarrow C \parallel A) + (b \rightarrow C \parallel B)$
  folding

- Correctness Theorem:
  the operational semantics is preserved during the transformation from the initial program to the final one.

Partial Evaluation of Constraint Programs

[65] *Concurrent Constraint Programs*

- Since *constraints may encode the input information known at compile time*, partial evaluation can be carried out as standard computation.

- Constraints may denote *communications* and *synchronizations*. During partial evaluation one may simplify communications and synchronizations, and avoid unnecessary non-determinism. Efficiency may be greatly improved.

---

[621] *Constraint Programs*

- A new rule (Contextual Constraint Replacement) and two strategies (Context Propagation and Invariant Promotion) for deriving very efficient residual programs.

PRESERVING TERMINATION (1)

[30]

- Often one has to apply goal replacement (GR) for allowing folding:

  e.g., $A, B \rightarrow B, A$

  associativity of append

- Study of GR which preserves c.a.s. semantics and universal termination

---

[31, 630]

- Study of termination of programs:

  1. definition and study of the class of the *well-terminating* logic programs for which under the leftmost selection rule all *well-moded* queries yield finite derivations.

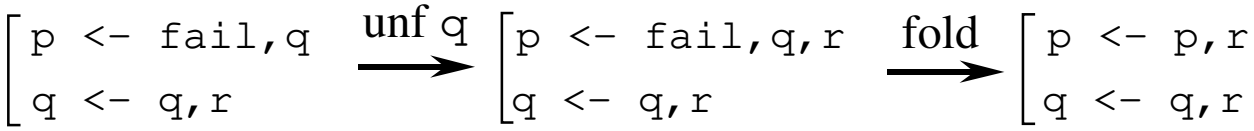
  2. study of conditions which ensure the compositionality of well-termination.

PRESERVING TERMINATION (2)

[32]

- For correct unfold/fold transformations it is important to check that some goals do not produce finitely failing derivations using the leftmost selection rule.

Example.



`<- p` fails `<- p` loops

---

[33]

- Study of conditions on programs and queries which guarantee the presence of at least a successful leftmost derivation.

REFERENCES

1. [57] Albert E., M. Alpuente, M. Falaschi, P. Julian, G. Vidal: *Improving Control in Functional Logic Program Specialization*, Proc.of the Static Analysis Symp. (SAS'98), LNCS 1503, pp.262-277, 1998

2. [55] Alpuente M., M. Falaschi, G. Moreno, G. Vidal: *Safe Folding/Unfolding with Conditional Narrowing*, Proc. of ALP'97, LNCS 1298, pages 1-15, (1997)

3. [328] Alpuente Maria, Moreno Falaschi, German Vidal: *A Unifying View of Functional and Logic Program Specialization*, Computing Surveys, Vol. 30, No. 3es, Pages 9-es, ACM Press, (Sept. 1998).

4. [54] Alpuente Maria, Moreno Falaschi, German Vidal: *Partial Evaluation of Functional Logic Programs*, ACM Transactions on Programming Languages and Systems, 20(4):768-844 (1998)

5. [526] Alpuente Maria, Moreno Falaschi, Gines Moreno, German Vidal: *A Transformation System for Lazy Functional Logic Programs*, In A. Middeldorp and T. Sato, Eds., Proc. of the International Symposium on Functional and Logic Programming, FLOPS '99, Springer, LNCS 1722, pp. 147-162, 1999.

6. [32] Bossi A., N. Cocco: *Programs Without Failures*, N. Fuchs ed., Logic program Synthesis and Transformation, Lecture Notes in Computer Science 1463, Springer-Verlag,1998, 28-48.

7. [30] Bossi A., N. Cocco: *Replacement Can Preserve Termination*, In J. Gallagher ed., Logic Program Synthesis and Transformation, Springer-Verlag, 1997, 104-129.

8. [33] Bossi A., N. Cocco: *Successes in Logic Programs*, In: Logic-Based Program Synthesis and Transformation - LOPSTR'98 selected papers (P. Flener, ed.), Lecture Notes in Computer Science n.1559, Springer-Verlag, 1999, pp. 219-239.

9. [35] Bossi A., S. Rossi: *Call-Correct Specialisation of Logic Programs*, ICTCS'98:Sixth Italian Conference on Theoretical Computer Science, 1998, 335-346.

10. [34] Bossi A., S. Rossi: *Specialising Logic Programs wrt Pre/Post Specifications*, Extended abstract of LOPSTR'98, Dept. of Computer Science, University of Manchester, UK, UMCS-98-6-1, 1998

11 [630] A. Bossi, N. Cocco, S. Etalle: *Transformation of Left Terminating Programs*,Proc. of the Intern. Workshop on Logic-based Program Synthesis and Transformation (LOPSTR'99), Rapporto di Ricerca CS-99-16. Dipartimento di Informatica, Universita' Ca' Foscari di Venezia, pp. 199-206, 1999.

12. [31] Etalle S., A. Bossi, N. Cocco: *Termination of Well-Moded Programs*, Journal of Logic Programming, 38, 1999, 243-257.

13. [65] Etalle S., M. Gabbrielli, *Partial Evaluation of Concurrent Constraint Languages*, ACM Computing Surveys, 1998.

14. [64] Etalle S., M. Gabbrielli and M.C. Meo: *Transformation of CCP programs*, Proc. of CONCUR'98, LNCS 1466. Springer-Verlag,1998.

15. [621] Fioravanti Fabio, Alberto Pettorossi, Maurizio Proietti: *Rules and Strategies for Contextual Specialization of Constraint Logic Programs*, M. Leuschel (ed.) Proceedings of the ICLP'99 Workshop on Optimization and Implementation of Declarative Programming Languages (WOID'99), Las Cruces, New Mexico, USA, Dec.2-3,1999, pp. 1-9.

16. [21] Pettorossi A., M. Proietti: *Synthesis and Transformation of Logic Programs Using Unfold/Fold Proofs*, Journal of Logic Programming, Vol. 41, No. 2&3 (Nov. 1999), pp. 197-230.

17. [20] Pettorossi A., M. Proietti: *Transformation of Logic Programs*, Handbook of Logic in Artificial Intelligence and Logic, Oxford University Press, 1998

18. [619] Pettorossi Alberto , Maurizio Proietti: *Transformation Rules for Logic Programs with Goals as Arguments,* Pre-Proceedings 9th LOPSTR '99 Report University of Venice (Italy) CS-99-16, pages 171-178.

19. [620] Proietti Maurizio, Alberto Pettorossi: *Transforming Inductive Definitions*, D. De Schreye (ed.) Proceedings of the 16th International Conference on Logic Programming (ICLP'99), Las Cruces, New Mexico, USA, Nov. 29 - Dec. 4, 1999, MIT Press, pp. 486-499.