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

FIRST WORKSHOP

ROMA, ITALY

21-23 December 1998

http://strudel.di.unipi.it/p40/tectra.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

- usually a small, local replacement
- many steps to get the final program
- need for strategies to get efficiency improvements

• Separation of the correctness and efficiency concerns.

• Structuring program derivations.

• Easy correctness proofs of complex transformations.

• Common framework for the combination and extension of different techniques.

A PROGRAM TRANSFORMATION : LIST REVERSAL

INITIAL PROGRAM P_0 : $O(n^2)$

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), <u>append(T1,[H],L1), append(L1,Acc,R)</u>

T1 H 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), <u>append(T1,[H|Acc],R)</u>

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 P_6 : $O(n)$

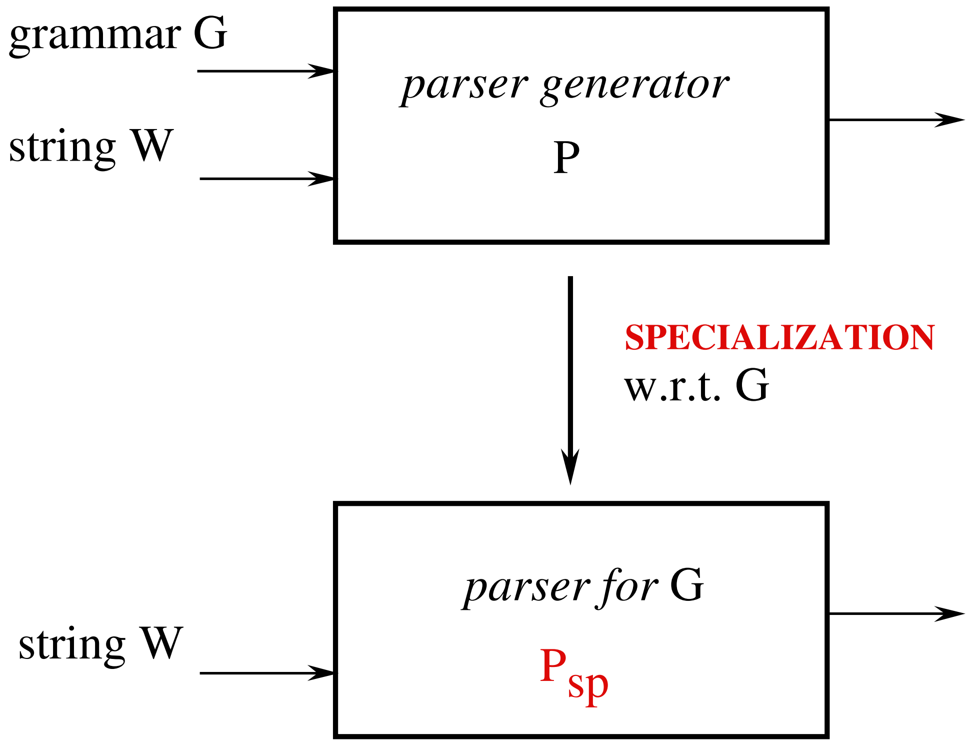
1. rev([ ],[ ]) ←
10. rev([H|T],R) ← rev_acc(L,[H],R)
6. rev_acc([ ],Acc,Acc) ←
8. rev_acc([H|T],Acc,R) ← rev_acc(T,[H|Acc],R)

UNFOLD/FOLD TRANSFORMATION RULES

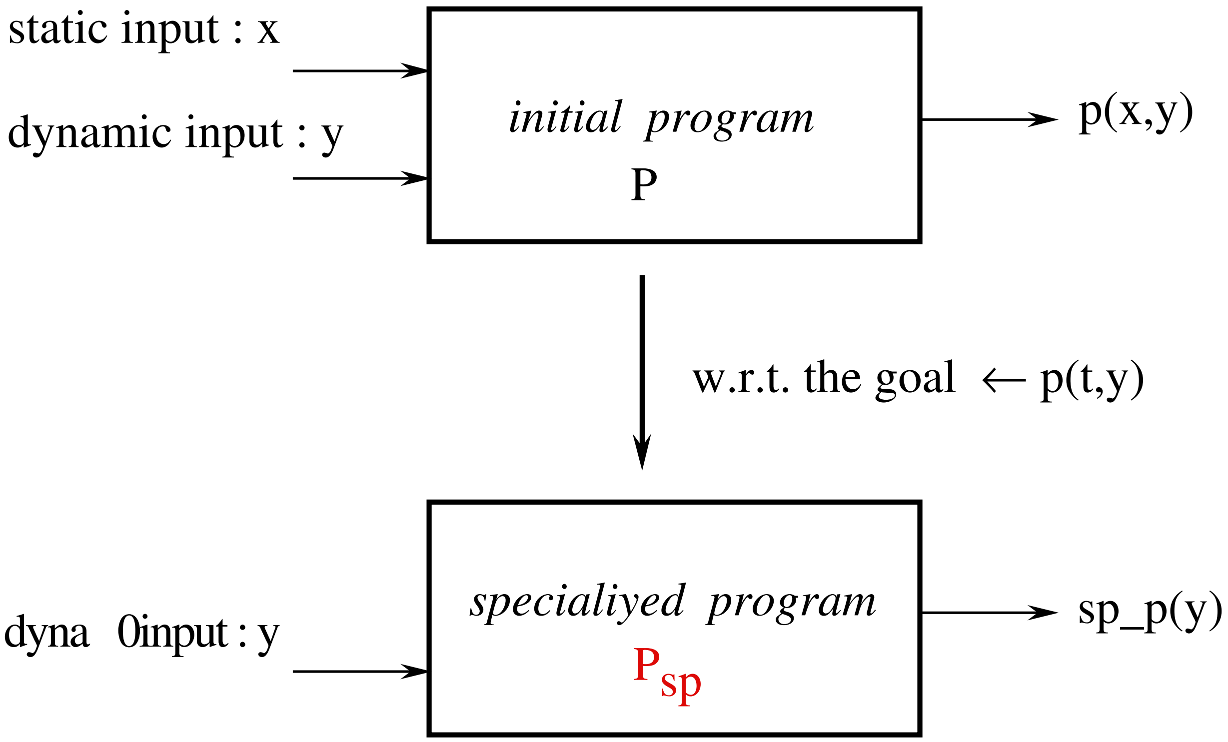
1.	*Definition Introduction*	DI
2.	*Definition Elimination*	DE
3.	*Unfolding*	U
4.	*Folding*	F
5.	*Goal Replacement*	GR
6.	*Equalities & Equivalences*	EQ

collectively called *unfold / fold transformation rules*

PROGRAM SPECIALIZATION : AN EXAMPLE



LOGIC PROGRAM SPECIALIZATION



such that

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

PARTIAL DEDUCTION [Lloyd-Shepherdson 91]

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

In the above quantified formula, t and y range over possibly non-ground terms.
In the goal ← p(t,y), y is a variable.
Failure is not considered.

SPECIALIZING A MATCHER

pattern P

string S

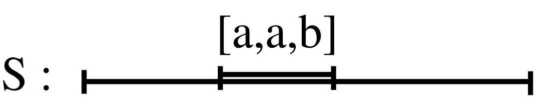
general matcher

match(P,S) ← match1(P,S,P,S)
match1([ ],X,Y,Z) ←
match1([A|Ps],[A|Ss],P,S) ←
    match1(Ps,Ss,P,S)
match1([A|Ps],[B|Ss],P,[C|S]) ←
    A ≠ B, match1(P,S,P,S)

match(P,S)

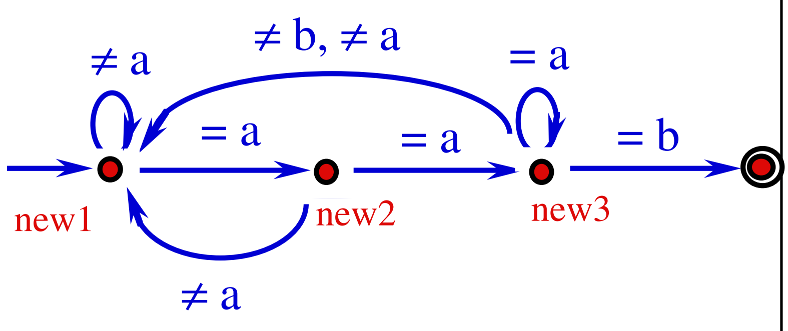
w.r.t. the goal ← match([a,a,b],S)

(using **PARTIAL DEDUCTION**)



matcher for the pattern [a,a,b]

ssring S



sp_match(P,S)

It is a deterministic finite automaton which accepts strings in Σ^* a a b Σ^*, as in the Knuth-Morris-Pratt algorithm.

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

PROGRAM SYNTHESIS

Given: a formula φ of the form:

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

and a program P defining spec and context

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

$\boxed{M(P \cup Eureka\_p) \models \varphi}$

Synthesis as abduction.

PROVING FORMULAS VIA UNFOLD / FOLD PROOFS

Given a definite program P and
a formula φ: $\forall X.\ (\exists Y.\ G_1(X,Y)\ \leftrightarrow\ \exists Z.\ G_2(X,Z))$

if

$$\begin{tikzcd}
P \arrow[d, "DI"'] & & P \arrow[d, "DI"] \\
P \cup \boxed{new_1(X) \leftarrow G_1(X,Y)} \arrow[d, "\begin{array}{c}\text{totally correct}\\ \text{transformations}\\ \text{using U / F / GR / EQ}\\ \text{with initial program P}\end{array}"'] & & \boxed{new_2(X) \leftarrow G_2(X,Z)} \cup P \arrow[d, "\begin{array}{c}\text{totally correct}\\ \text{transformations}\\ \text{using U / F / GR / EQ}\\ \text{with initial program P}\end{array}"] \\
P \cup \boxed{S_1} \arrow[rr, phantom, "\overset{new_1\,/\,new_2}{=}"] & & \boxed{S_2} \cup P
\end{tikzcd}$$

then $M(P) \models \varphi$.

UNFOLD / FOLD PROOF FOR FIBONACCI

Proof of the functionality of Fibonacci, assuming the functionality of plus, that is,

∀ X, Y, Z1, Z2. (plus(X,Y,Z1), plus(X,Y,Z2)) ↔ (plus(X,Y,Z1), Z1 = Z2).

We are given the program FIB :

1. fib(0,s(0)) ←
2. fib(s(0),s(0)) ←
3. fib(s(s(N)),F2) ← fib(s(N),F1), fib(N,F), plus(F1,F,F2)

We need to show that M(F) ⊨E, where E is equivalence:

∀ N, F1, F2. (fib(N,F1), fib(N,F2)) ↔ (fib(N,F1), F1 = F2).

FIB ∪ C_1: new1(N,F1,F2) ←
fib(N,F1), fib(N,F2)

unfold wrt fib(N,F1)
and fib(N,F2)

new1(0,s(0),s(0)) ←
new1(s(0),s(0),s(0)) ←
new1(s(s(N)),F1,F2) ←
fib(s(N),F11), fib(N,F12),
plus(F11,F12,F1),
fib(s(N),F12), fib(N,F22),
plus(F21,F22,F2)

fold (twice)
using C_1

FIB ∪ new1 (0),s(0)) ←
new1(s(0),s(0),s(0)) ←
new1(s(s(N)),F1,F2) ←
new1(s(N),F11,F21),
new1(N,F12,F22),
plus(F11,F12,F1),
plus(F21,F22,F2)

C_2: new2(N,F1,F2) ← ∪ FIB
fib(N,F1), F1=F2

unfold wrt fib(N,F1)

new2(0,s(0),s(0)) ←
new2(s(0),s(0),s(0)) ←
new2(s(s(N)),F1,F2) ←
fib(s(N),F11), fib(N,F12),
plus(F11,F12,F1), F1 = F2

GR (*functionality of* plus)

. . .
new2(s(s(N)),F1,F2) ←
fib(s(N),F11), fib(N,F12),
plus(F11,F12,F1), plus(F11,F12,F2)

generalization + equality introduction (twice)

. . .
new2(s(s(N)),F1,F2) ←
fib(s(N),F11), fib(N,F12),
plus(F11,F12,F1), plus(F21,F22,F2),
F11 = F21, F12 = F22

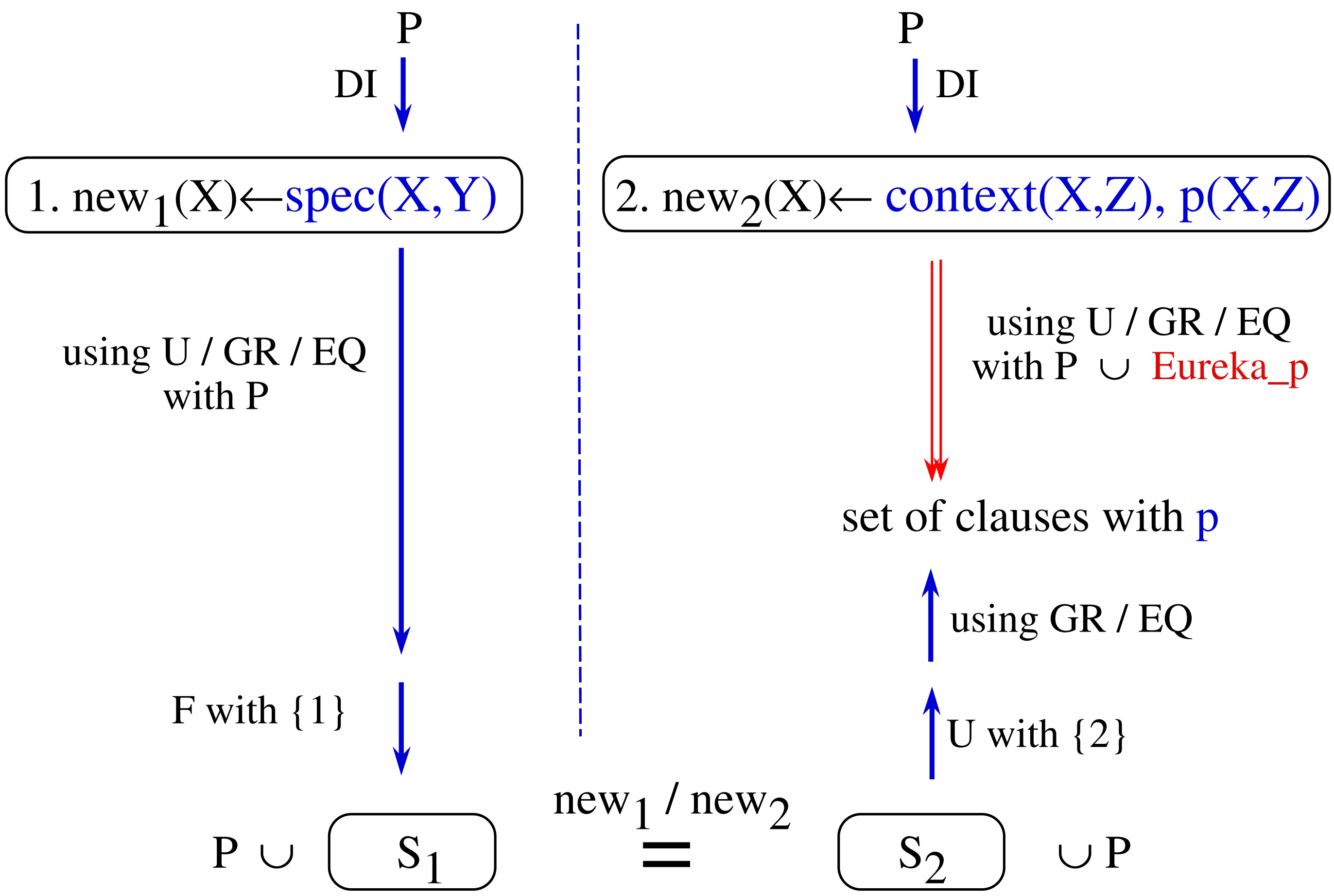
fold (twice) using C_2

new1 / new2
=

new2(0,s(0),s(0)) ← ∪ FIB
new2(s(0),s(0),s(0)) ←
new2(s(s(N)),F1,F2) ←
new2(s(N),F11,F21),
new2(N,F12,F22),
plus(F11,F12,F1),
plus(F21,F22,F2)

SYNTHESIS VIA UNFOLD / FOLD PROOFS

Given a definite program P and
a formula φ: $\forall X.\ [\exists Y.\ spec(X,Y) \leftrightarrow \exists Z.\ context(X,Z), p(X,Z)]$



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

Specialization of Interpreters w.r.t. Programs is Compilation

[Futamura 71]

output = [program] (static_input, dynamic_input)
= [sp_program] (dynamic_input)

sp_program = [**PARTIAL_EVALUATOR**] (program, static_input)

output = [source_program] (input)
= [interpreter] (source_program, input)
= [[**PARTIAL_EVALUATOR**] (interpreter, source_program)] (input)
= [target_program] (input)

target_program = [**PARTIAL_EVALUATOR**] (interpreter, source_program)
= [[**PARTIAL_EVALUATOR**]
(**PARTIAL_EVALUATOR,** interpreter)] (source_program)
target_program = [compiler] (source_program)

compiler = [**PARTIAL_EVALUATOR**] (**PARTIAL_EVALUATOR,** interpreter)

Transformation and Synthesis
= PAPERS =

	Logic Programs	Functional Logic Programs	Concurrent Constraint Programs
PROGRAM TRANSFORMATION	**20**: *Review* **30**: *goal replacemnt* **31**: *termination* **32-33**: *queries w/o failures and queries with success*	**55**: *unfold/fold rules and their correctness*	**64**: *unfold/fold rules and their correctness*
PROGRAM SPECIALIZATION (= PARTIAL EVALUATION = PARTIAL DEDUCTION)	**20**: *Review* **34-35**: *Specializ. wrt call conditions*	**54-57**: *an algorithm and a system*	**65**: *usefulness of partial evaluation*
PROGRAM SYNTHESIS	**21**: *Synthesis via unfold/fold proofs*		

Paper numbers are those of the list in the home 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 Venezia	Udine	L'Aquila Pisa
PROGRAM SPECIALIZATION (= PARTIAL EVALUATION = PARTIAL DEDUCTION)	Roma 2 Venezia	Udine	Pisa Udine
PROGRAM SYNTHESIS	Roma 2		

FUNCTIONAL LOGIC PROGRAMS

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

conditional term rewriting — lhs — rhs — sequence of equations

- *Program*:
  1. even(0) $\rightarrow$ true
  2. even(s(x)) $\rightarrow$ odd(x)
  3. odd(x) $\rightarrow$ true $\Leftarrow$ even(x) = false
  4. odd(x) $\rightarrow$ false $\Leftarrow$ 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

  $\downarrow$ {y/s(x)}

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

  {4} true = true, false = false, even(x) = true

  $\downarrow$ {x/0}

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

  true, true, true $\longrightarrow$ even(s(s(0))) = true

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

[55] Transformation

- Various unfolding rules.
- Innermost folding rule.
- Correctness Theorem:
  preservation of the computed answer substitution semantics.

[54, 57] Partial Evaluation

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

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

- Equivalence of the operational and the fixpoint semantics.

```
the second clause should have been:
sum(X,0,X) <- sum(X,0,X)
but this is not possible by
Sabina's specialization. Impossible
to introduce recursion.(?)
```

Transformation of Concurrent Constraint Programs

[64]

- Language

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

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

  processes P ::= <D, A>

- 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 Concurrent Constraint Programs

[65]

- 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.

PRESERVING TERMINATION (1)

[30]

- Often one has to apply goal replacement (GR) for allowing folding:
  e.g., $A, B \rightarrow B, A$
  properties of predicates (such as associativity of append)

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

---

[31]

- 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 the well-termination.

PRESERVING TERMINATION (2)

[32]

- For correct unfolding it is often important to check that some goals do not produce finitely failing derivations using the leftmost selection rule.

Example.

```
p <- q, r                unfolding r                     p <- fails
q <- q        (non leftmost and finitely failing)
              ------------------------------------>

<- p   loops                                            <- p   fails
```

---

[33]

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