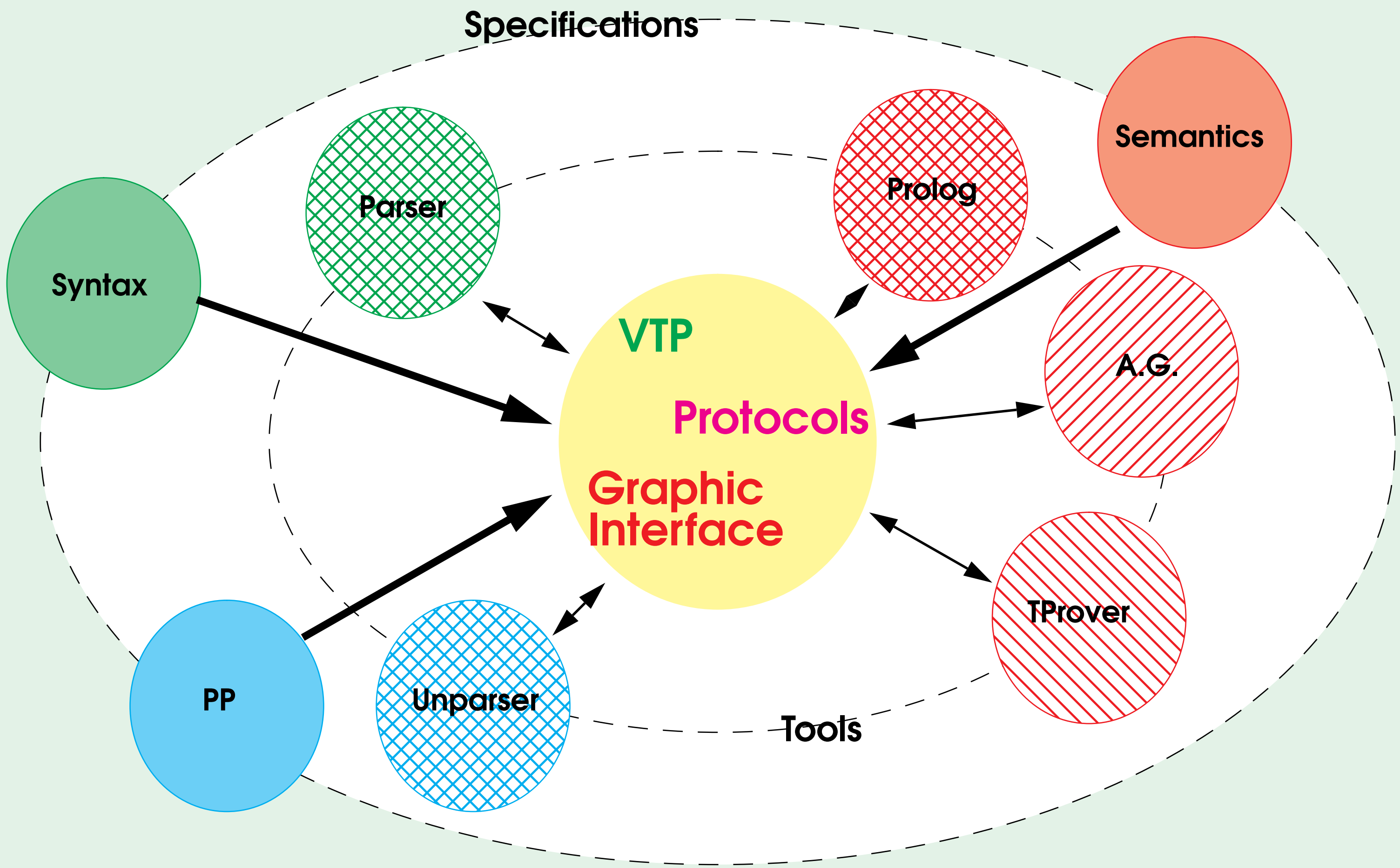
Logical Programming and Error Recovery

Thierry Despeyroux
INRIA-Sophia Antipolis

- Context and Motivations
- What is the problem ?
- A proposal to solve it
- Implementation

Motivations



CENTAUR

Esprit projects GIPE/GIPE2

INRIA

Motivations

Specifications

- Abstract Syntax
- BNF
- Pretty Printing Rules
- Typing Rules
- Dynamic Semantics
- Translation Rules

Tools

- Syntactical Editor
- Parser
- Unparser
- Type Checker
- Interpreter
- Compiler
  Program transformation

Proofs

- Effectiveness
- Compatibility
  parser/unparser
- Subject Reduction Theorem
- Proof of translation
  Proof of transformation

How far from a specification to a product?

- Implementation
- M/M interface
- Efficiency
- *Error Recovery*
  - * **Where**
  - * **What**
  - * **Proceed if possible**
  - * **Not pollute the original code**

A simple Typol example

type_check:

```
Env |- V : T      Env |- E : T
--------------------------------
   Env |-  V:=E  : T

Env |- int X : int

Env |- true : bool

Env |- false : bool

look_up(Env |- X : T)
-------------------------
  Env |- var X : T

Env |- A : T      Env |- B : T
---------------------------------
     Env |- A+B : T
```

look_up:

```
{X:T,Q} |- X : T

   Q |- X : T
-------------------
{X:T,Q} |- X : T
```

Executing a semantic specification = Constructing a proof

{X:bool, X:int} |- X:=5 : ??

problem

```
    ------------------
    {X:int} |- X : int
---------------------------     ---------------------------
{X:bool, X:int} |- X : int      {X:bool,X:int} |- 5 : int
-----------------------------------------------------------
          {X:bool, X:int} |- X:=5 : int
```

proof

answer: ***?? = int***

The proof does not express any order of execution.

What happens in case of error ?

{X:bool} |- X:=5 : ??

problem

answer: **no.**

{X:bool} |- X : bool ?

{X:bool} |- X:=5 : bool

uncomplete proofs

? {X:bool} |- 5 : int

{X:bool} |- X:=5 : int

Uncomplete proofs may leads to missing bindings:

linear declarations:
e1 |- A -> e2   e2 |- B -> e3
----
e1 |- A;B -> e3

Adding Recovery Rules

type_check:

Env |- *V* : T   Env |- *E* : T

   Env |- *V:=E* : T

Env |- int X : *int*

Env |- *true* : *bool*

Env |- *false* : *bool*

look_up(Env |- X : T)

 Env |- var X : T

Env |- *A* : T   Env |- *B* : T

     Env |- *A+B* : T

recovery:

Env |- Exp : T

{X:int} |- *X:=5* : ??

problem

{X:int} |- *X:=5* : T

type error on X:=5

Recovery rules cannot be used everywhere.

They must be used only when nothing else is possible, so there is an implicit notion of negation.

Accuracy

$\{X{:}int\} \vdash X{:=}Y{+}5 : ??$

problem

$$\dfrac{\dfrac{}{\{X{:}int\} \vdash X : \mathrm{int}} \qquad \dfrac{}{\{X{:}int\} \vdash Y{+}5 : int}}{\{X{:}int\} \vdash X{:=}Y{+}5 : int}$$

type error on Y+5

$$\dfrac{\dfrac{}{\{X{:}int\} \vdash X : int} \qquad \dfrac{\dfrac{}{\{X{:}int\} \vdash Y : int} \quad \dfrac{}{\{X{:}int\} \vdash 5 : int\}}}{\{X{:}bool\} \vdash Y{+}5 : int}}{\{X{:}int\} \vdash X{:=}Y{+}5 : int}$$

type error on Y

Accuracy gives a partial order on proofs.

Some are of greater interest than others...

A more complete example

type_check:

Env |- *V* : T   Env |- *E* : T

Env |- *V:=E* : T

Env |- int X : *int*

Env |- *true* : *bool*

Env |- *false* : *bool*

look_up(Env |- X : T)

Env |- var X : T

Env |- *A* : T   Env |- *B* : T

Env |- *A+B* : T

recovery:

Env |- *A* : T1   Env |- *B* : T2

Env |- *A+B* : T

Env |- Exp : T

look_up:

{X:T,Q} |- X : T

Q |- X : T

{X:T,Q} |- X : T

recovery

Env |- X : T

Design choice: when several recovery rules with the same accuracy can be used, we prefer the first one.

Incremental development

Correctness

P: initial program
P(A): a goal
$\overline{\text{P}}$(R) where R = recovery | norecovery : the same program with some recovery rules
$\overline{\text{P}}$(R,A) a goal for the program with error recovery

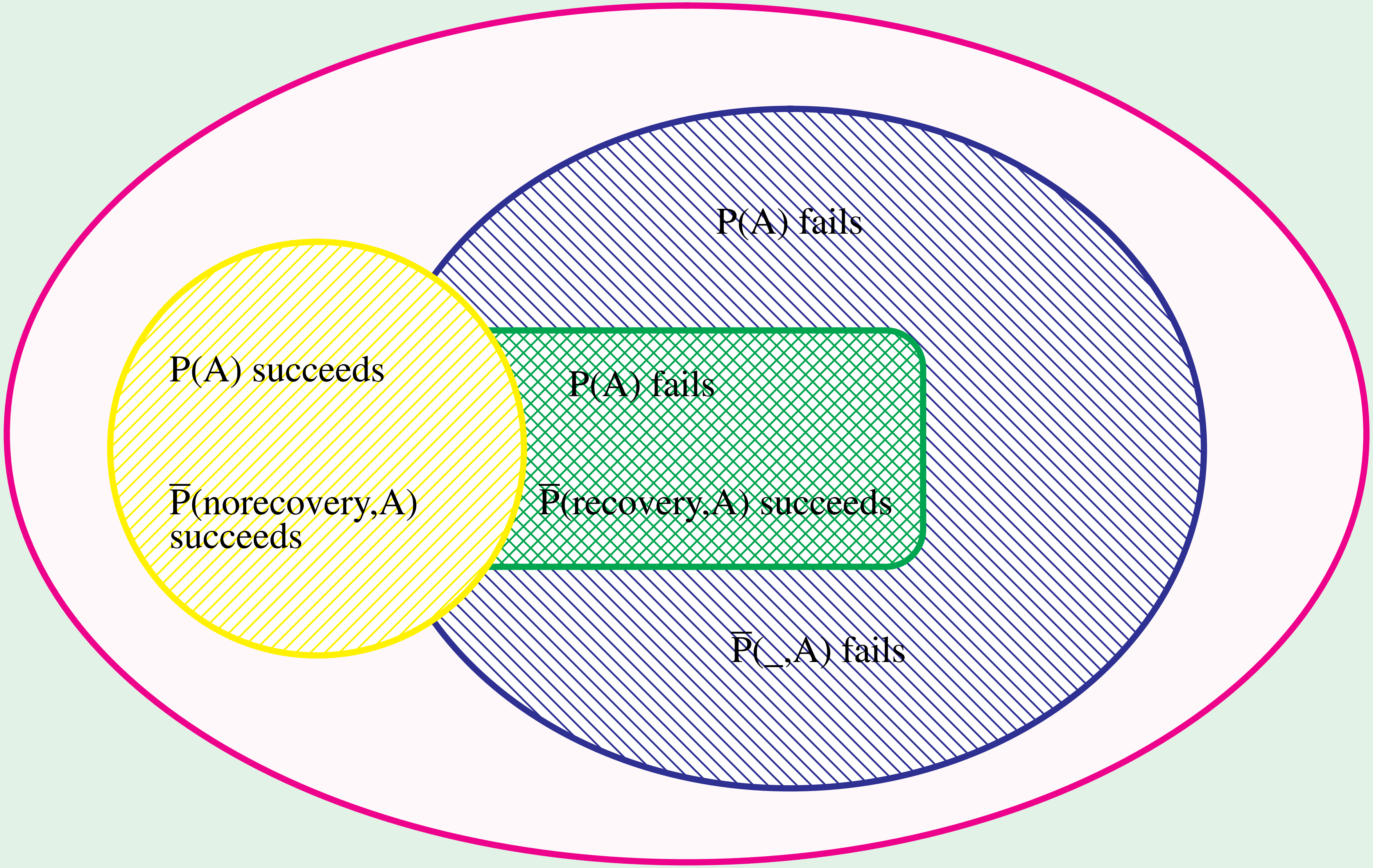
- **Completeness:** If P(A) succeeds with some substitution σ then $\overline{\text{P}}$(norecovery,A) succeeds with the same substitution.

- **Soudness:** If $\overline{\text{P}}$(norecovery,A) succeeds with some substitution σ then P(A) succeeds with the same substitution.

- **Expressiveness:** If $\overline{\text{P}}$(R,A) succeeds with R=recovery then P(A) fails.

- **Accuracy:** Given a goal, if two valid complete proofs with recovery P_1 and P_2 can be built and P_1 is more accurate than P_2, then P_2 will be rejected.

Correctness



Implementation

- Take possible backtracking into account

  Use soft cuts, not cuts

```
------------------------------
{X:bool, X:int} |- X : bool          • • •
-------------------------------------------------
            {X:bool, X:int} |- X:=5 : bool
```

- Take coroutining into acount

  Check for delayed goals inside a negation or before a soft cut

  If a goal is delayed before a soft cut and fails after it,
  error recovery will not be effective

- Don't slow down normal execution

  But if you can provide pertinent error messages the user will be happy

Implementation

$P'_1(\Pi_i A_i) \text{ :- } Q^1(\Pi_k B_{1,k}).$
...
$P'_n(\Pi_i A_i) \text{ :- } Q^n(\Pi_k B_{n,k}).$

recovery

$P''_1(\Pi_i A_i) \text{ :- } R^1(\Pi_k C_{1,k}).$
...
$P''_m(\Pi_i A_i) \text{ :- } R^m(\Pi_k C_{n,k}).$

$\overline{P}(\text{Rec}, \quad \Pi_i A_i) \text{ :- } \overline{P'}(\text{norecovery}, \Pi_i A_i).$
$\overline{P}(\text{recovery}, \Pi_i A_i) \text{ :- snot}(\overline{P'}(\text{norecovery}, \Pi_i A_i)),$
$\qquad \overline{P'}(\text{recovery}, \Pi_i A_i).$
$\overline{P}(\text{recovery}, \Pi_i A_i) \text{ :- snot}(\overline{P'}(\_, \Pi_i A_i)),$
$\qquad \overline{P''}(\Pi_i A_i).$

$\overline{P'}_1(\text{Rec}, \Pi_i A_i) \text{ :- } \overline{Q}^1(\text{Rec}, \Pi_k B_{1,k}).$
...
$\overline{P'}_n(\text{Rec}, \Pi_i A_i) \text{ :- } \overline{Q}^n(\text{Rec}, \Pi_k B_{n,k}).$

$\overline{P}''_1(\Pi_i A_i) \text{ :- scall}(\overline{R}^1(\text{Rec}, \Pi_k C_{1,k}), \text{soft\_cut}.$
...
$\overline{P}''_m(\Pi_i A_i) \text{ :- scall}(\overline{R}^m(\text{Rec}, \Pi_k C_{n,k}), \text{soft\_cut}.$

MU-Prolog / ECLiPSe

Conclusion

- clean limit between the original code and error recovery
- easy incremental development of error recovery
- the semantics can be given at the meta-level (reasoning with proofs, not execution)
- a previous version implemented for Typol in Centaur
- the method is not limited to Typol, but can be used with logical programming in general