

# On generalising predicate abstraction

Birgit Elbl

CiE 06, Swansea

# Predicates as defined by logic programs

A logic program can be used to

- compute a function  $f: \mathbb{N}^n \longrightarrow \mathbb{N}$  or to

# Predicates as defined by logic programs

A logic program can be used to

- compute a function  $f: \mathbb{N}^n \longrightarrow \mathbb{N}$  or to
- decide membership in a subset of  $\mathbb{N}^n$

# Predicates as defined by logic programs

A logic program can be used to

- compute a function  $f: \mathbb{N}^n \longrightarrow \mathbb{N}$  or to
- decide membership in a subset of  $\mathbb{N}^n$

**But** symbolic LP-computation in Prolog-style is **more general**:

# Predicates as defined by logic programs

A logic program can be used to

- compute a function  $f: \mathbb{N}^n \longrightarrow \mathbb{N}$  or to
- decide membership in a subset of  $\mathbb{N}^n$

**But** symbolic LP-computation in Prolog-style is **more general**:  
Functions  $\llbracket p \rrbracket: \text{term}^n \longrightarrow \text{stream}(\text{answers})$  are computed.

# Predicates as defined by logic programs

A logic program can be used to

- compute a function  $f: \mathbb{N}^n \longrightarrow \mathbb{N}$  or to
- decide membership in a subset of  $\mathbb{N}^n$

**But** symbolic LP-computation in Prolog-style is **more general**:  
Functions  $\llbracket p \rrbracket: \text{term}^n \longrightarrow \text{stream}(\text{answers})$  are computed.

All computed functions satisfy the following conditions:

- The answers produced contain bindings **only for the variables in the arguments.**

# Predicates as defined by logic programs

A logic program can be used to

- compute a function  $f: \mathbb{N}^n \longrightarrow \mathbb{N}$  or to
- decide membership in a subset of  $\mathbb{N}^n$

**But** symbolic LP-computation in Prolog-style is **more general**:  
Functions  $\llbracket p \rrbracket: \text{term}^n \longrightarrow \text{stream}(\text{answers})$  are computed.

All computed functions satisfy the following conditions:

- The answers produced contain bindings **only for the variables in the arguments**.
- Application of a predicate symbol to tuples that are **variants** of each other produces **corresponding results**.

# Predicates as defined by logic programs

A logic program can be used to

- compute a function  $f: \mathbb{N}^n \longrightarrow \mathbb{N}$  or to
- decide membership in a subset of  $\mathbb{N}^n$

**But** symbolic LP-computation in Prolog-style is **more general**:  
Functions  $\llbracket p \rrbracket: \text{term}^n \longrightarrow \text{stream}(\text{answers})$  are computed.

All computed functions satisfy the following conditions:

- The answers produced contain bindings **only for the variables in the arguments**.
- Application of a predicate symbol to tuples that are **variants** of each other produces **corresponding results**.

This yields the subset of  $n$ -ary “**predicates**”:

$$\mathbb{P}_n \subseteq \text{term}^n \longrightarrow \text{stream}(\text{answers})$$

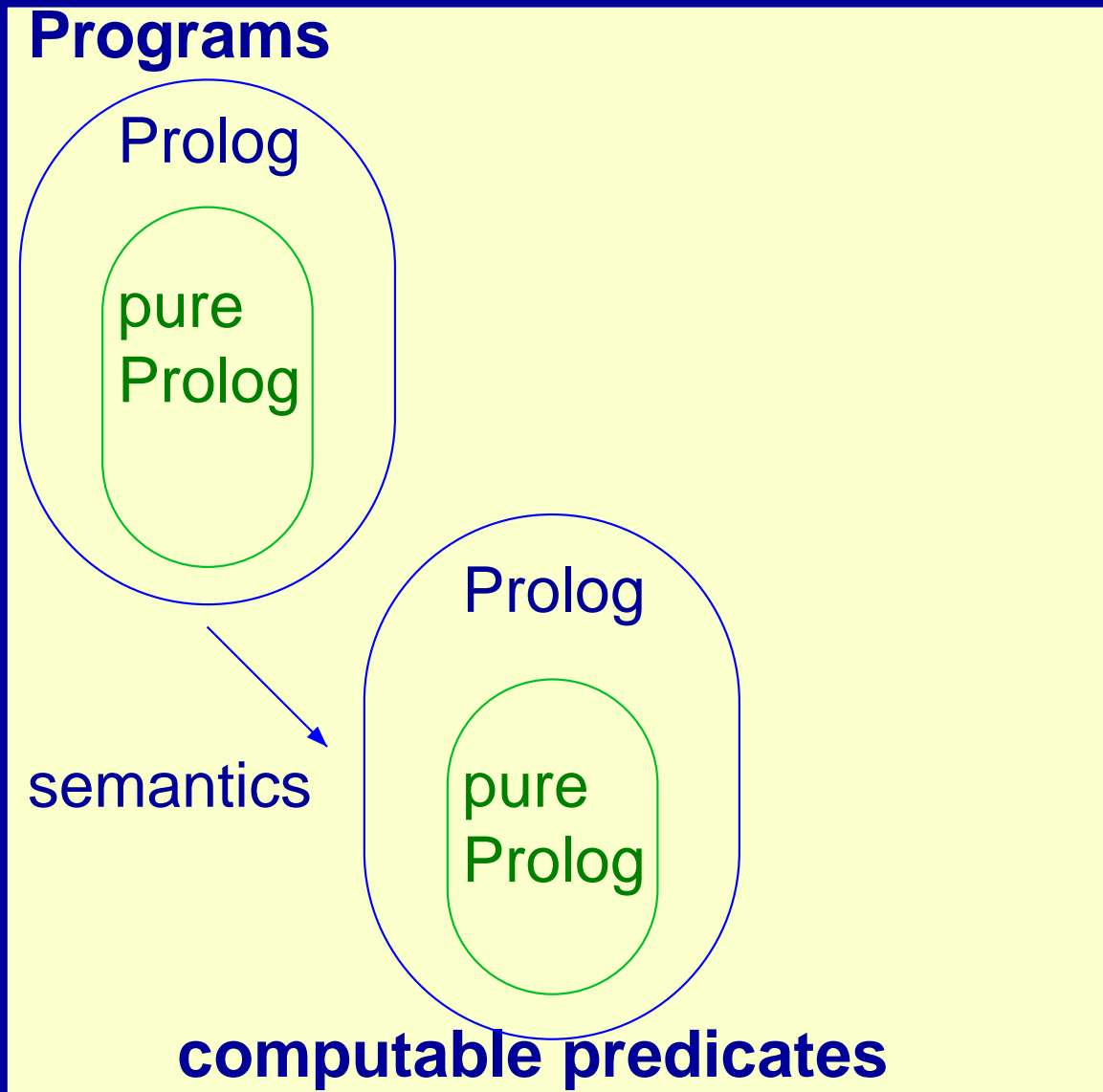
# Definable predicates

Programs

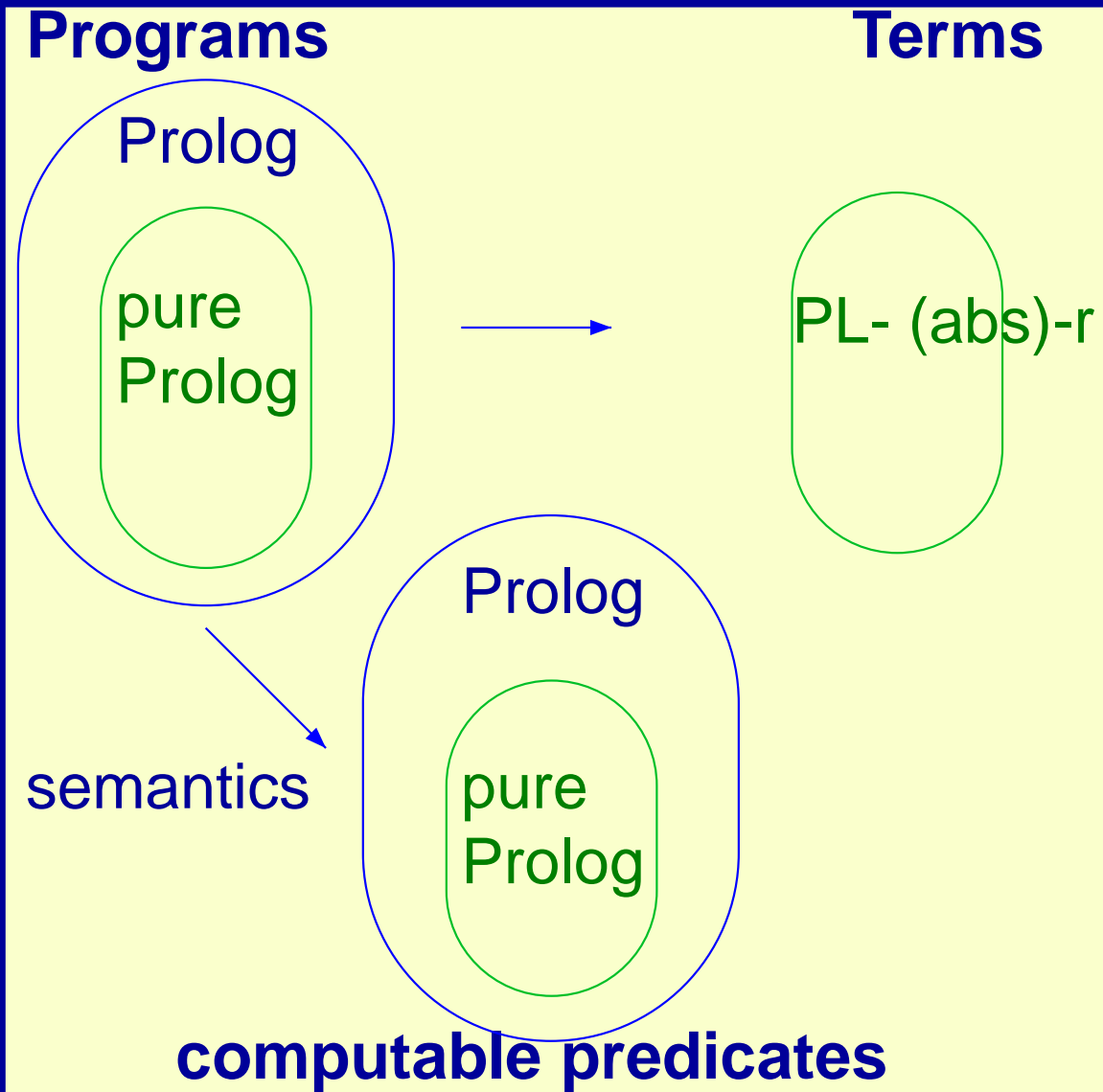
Prolog

pure  
Prolog

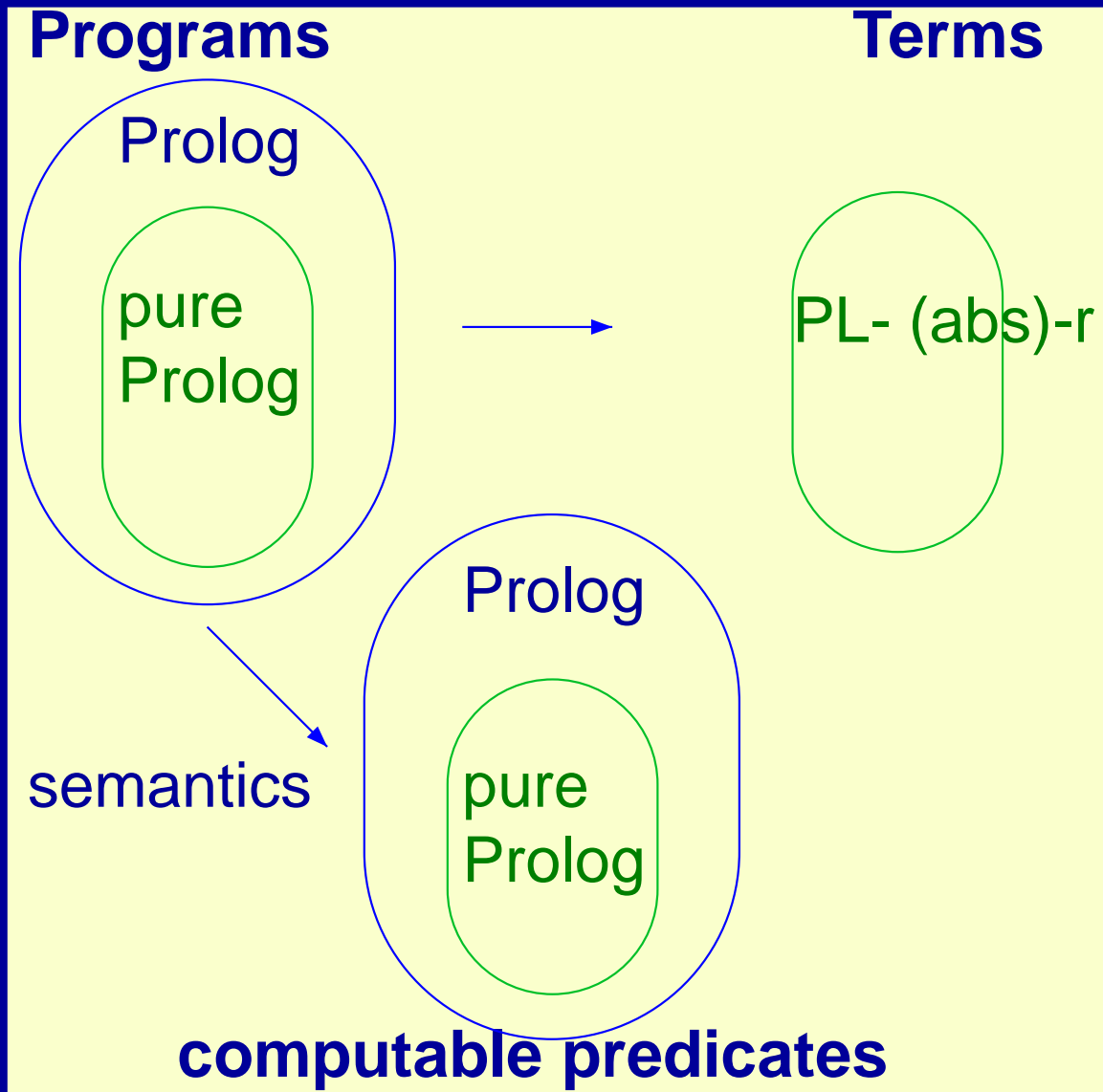
# Definable predicates



# Definable predicates



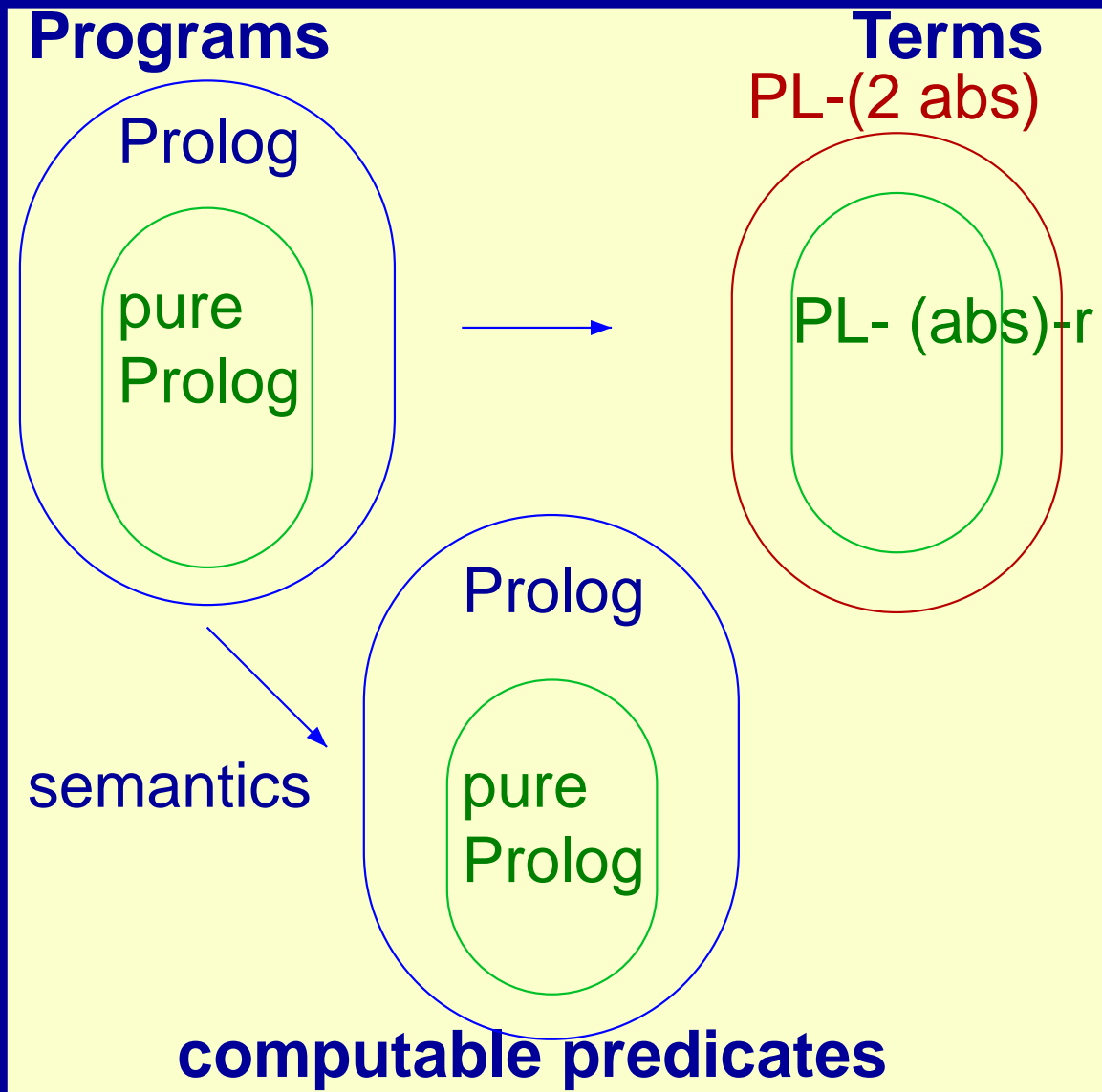
# Definable predicates



## PL-(abs)-r:

- only one form of predicate abstraction,
- subject to a restriction

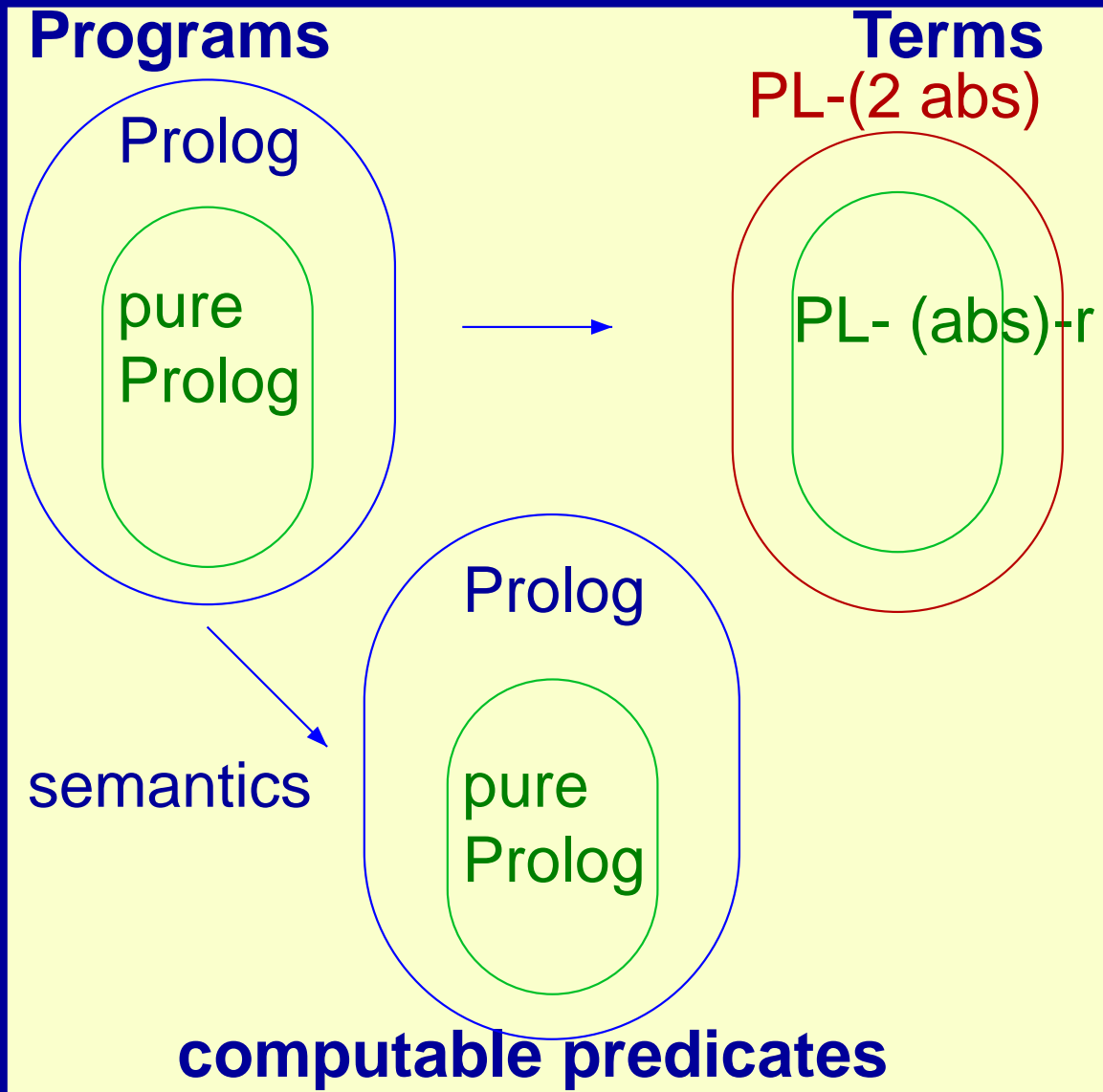
# Definable predicates



## PL-(abs)-r:

- only one form of predicate abstraction,
- subject to a restriction

# Definable predicates



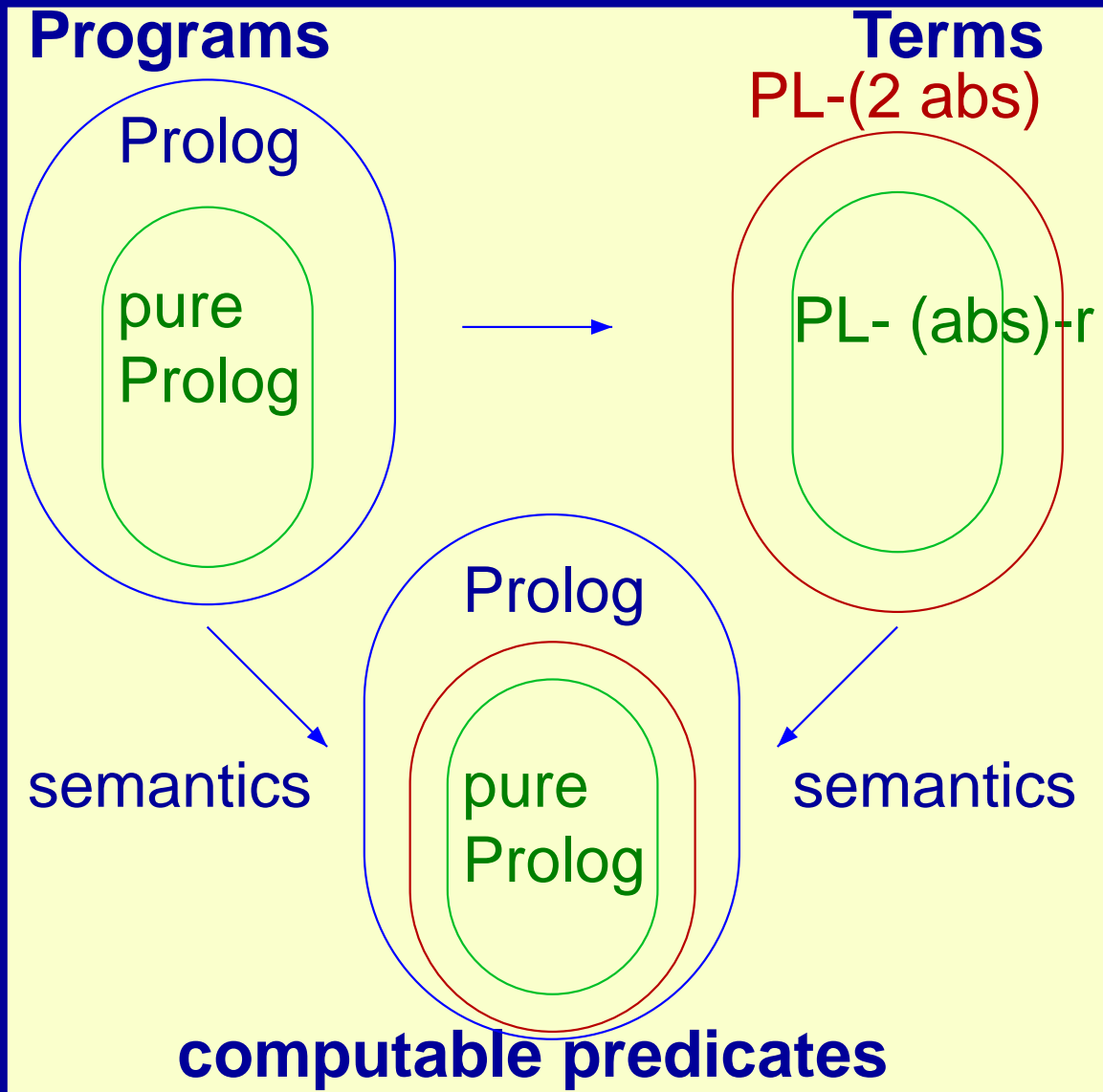
## PL-(abs)-r:

- only one form of predicate abstraction,
- subject to a restriction

## PL-(2 abs):

- restriction removed
- second form of predicate abstraction

# Definable predicates



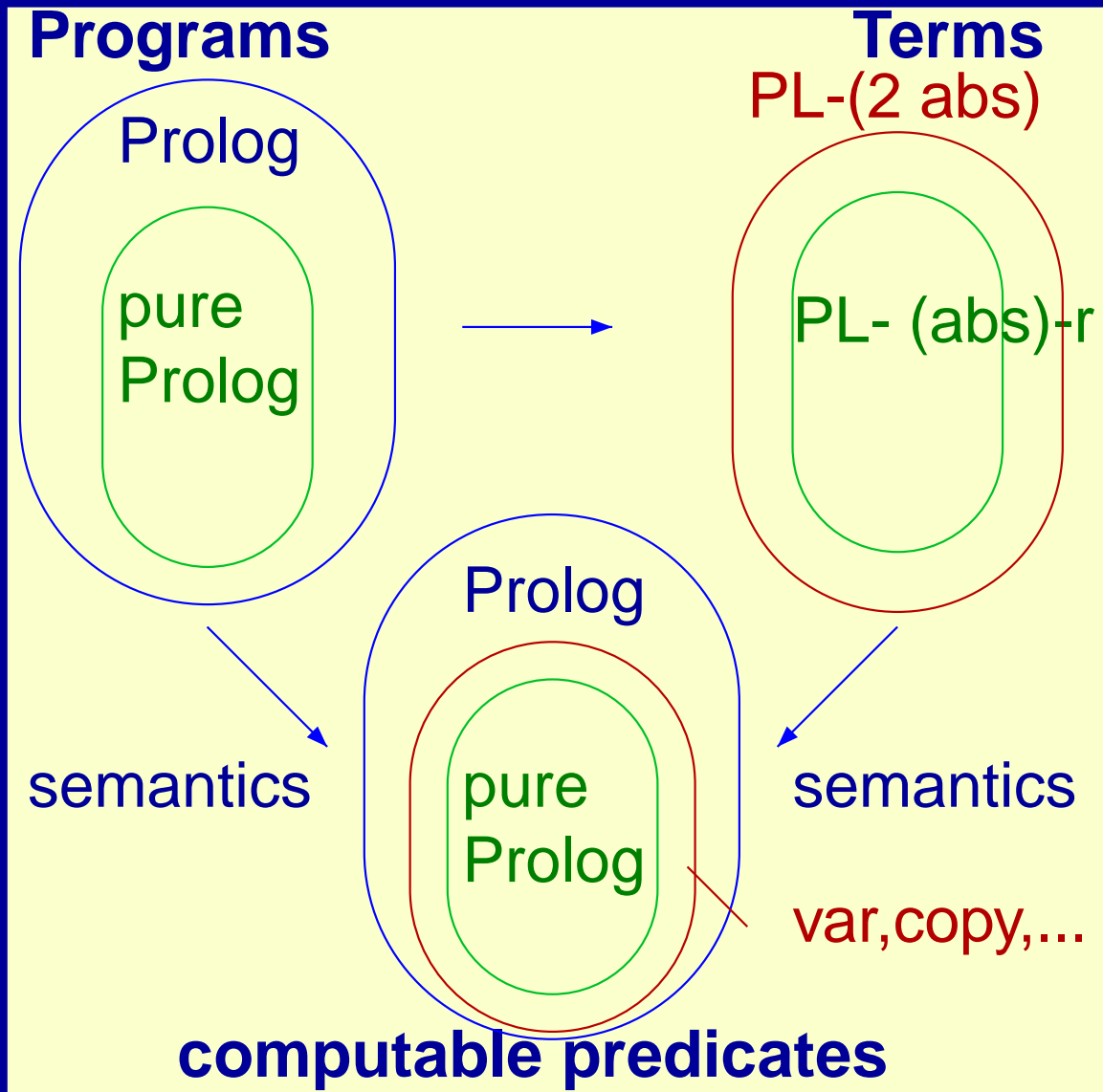
## PL-(abs)-r:

- only one form of predicate abstraction,
- subject to a restriction

## PL-(2 abs):

- restriction removed
- second form of predicate abstraction

# Definable predicates



## PL-(abs)-r:

- only one form of predicate abstraction,
- subject to a restriction

## PL-(2 abs):

- restriction removed
- second form of predicate abstraction

# System PL-(2 abs) / PL-(2 abs)-(let)

## Goals $G$

- are built from logical constants, equations and atoms  $P(\bar{t})$ ,

# System PL-(2 abs) / PL-(2 abs)-(let)

## Goals $G$

- are built from logical constants, equations and atoms  $P(\bar{t})$ ,
- using connectives and E-quantification

# System PL-(2 abs) / PL-(2 abs)-(let)

## Goals $G$

- are built from logical constants, equations and atoms  $P(\bar{t})$ ,
- using connectives and E-quantification
- (and  $\text{let } \bar{x} = \bar{t} \text{ in } G'$ )

# System PL-(2 abs) / PL-(2 abs)-(let)

## Goals $G$

- are built from logical constants, equations and atoms  $P(\bar{t})$ ,
- using connectives and E-quantification
- (and  $\text{let } \bar{x} = \bar{t} \text{ in } G'$ )

## Predicate terms $P$

- are predicate variables or

# System PL-(2 abs) / PL-(2 abs)-(let)

## Goals $G$

- are built from logical constants, equations and atoms  $P(\bar{t})$ ,
- using connectives and E-quantification
- (and  $\text{let } \bar{x} = \bar{t} \text{ in } G'$ )

## Predicate terms $P$

- are predicate variables or rec-terms:  $\text{rec } \bar{p}.(\bar{P})_i$  or

# System PL-(2 abs) / PL-(2 abs)-(let)

## Goals $G$

- are built from logical constants, equations and atoms  $P(\bar{t})$ ,
- using connectives and E-quantification
- (and  $\text{let } \bar{x} = \bar{t} \text{ in } G'$ )

## Predicate terms $P$

- are predicate variables or rec-terms:  $\text{rec } \bar{p}.(\bar{P})_i$  or
- obtained by predicate abstraction  $(\bar{x}.G')$  or  $(\bar{x} \mid G')$  where  $\text{LPV}(G) \subseteq \{\bar{x}\}$

# System PL-(2 abs) / PL-(2 abs)-(let)

## Goals $G$

- are built from logical constants, equations and atoms  $P(\bar{t})$ ,
- using connectives and E-quantification
- (and  $\text{let } \bar{x} = \bar{t} \text{ in } G'$ )

## Predicate terms $P$

- are predicate variables or rec-terms:  $\text{rec } \bar{p}.(\bar{P})_i$  or
- obtained by predicate abstraction  $(\bar{x}.G')$  or  $(\bar{x} \mid G')$  where  $\text{LPV}(G) \subseteq \{\bar{x}\}$

## Term systems PL-

where

# System PL-(2 abs) / PL-(2 abs)-(let)

## Goals $G$

- are built from logical constants, equations and atoms  $P(\bar{t})$ ,
- using connectives and E-quantification
- (and  $\text{let } \bar{x} = \bar{t} \text{ in } G'$ )

## Predicate terms $P$

- are predicate variables or rec-terms:  $\text{rec } \bar{p}.(\bar{P})_i$  or
- obtained by predicate abstraction  $(\bar{x}.G')$  or  $(\bar{x} \mid G')$  where  $\text{LPV}(G) \subseteq \{\bar{x}\}$

**Term systems** PL-  $\left\{ \begin{array}{l} (\text{abs})\text{-r} \\ (2 \text{ abs})\text{-r} \\ (2 \text{ abs}) \end{array} \right\}$

where “r” refers to the restriction  $\text{LPV}(G) \subseteq \{\bar{x}\}$  for  $(\bar{x}.G)$

# System PL-(2 abs) / PL-(2 abs)-(let)

## Goals $G$

- are built from logical constants, equations and atoms  $P(\bar{t})$ ,
- using connectives and E-quantification
- (and  $\text{let } \bar{x} = \bar{t} \text{ in } G'$ )

## Predicate terms $P$

- are predicate variables or rec-terms:  $\text{rec } \bar{p}.(\bar{P})_i$  or
- obtained by predicate abstraction  $(\bar{x}.G')$  or  $(\bar{x} \mid G')$  where  $\text{LPV}(G) \subseteq \{\bar{x}\}$

$$\text{Term systems PL-} \left\{ \begin{array}{l} (\text{abs})\text{-r} \\ (2 \text{ abs})\text{-r} \\ (2 \text{ abs}) \end{array} \right\} \left\{ \begin{array}{l} \text{-(let)} \end{array} \right\}$$

where “r” refers to the restriction  $\text{LPV}(G) \subseteq \{\bar{x}\}$  for  $(\bar{x}.G)$

# Semantics

**Denotational semantics**; cases predicate abstraction + let:

- $$\llbracket (\bar{x}.G) \rrbracket_{\varphi, \sigma}(\bar{t}) = \llbracket G \rrbracket_{\varphi, \sigma}(\{\bar{t}/\bar{x}\})$$

where  $\bar{x}$  are not in  $\sigma$

# Semantics

**Denotational semantics**; cases predicate abstraction + let:

- $\llbracket (\bar{x}.G) \rrbracket_{\varphi, \sigma}(\bar{t}) = \llbracket G \rrbracket_{\varphi, \sigma}(\{\bar{t}/\bar{x}\})$

- $\llbracket (\bar{x} \mid G) \rrbracket_{\varphi, \sigma}(\bar{t}) = \llbracket G \rrbracket_{\varphi, \{\bar{t}/\bar{x}\}}(\text{id}_{\text{LPV}(\bar{t})})$

where  $\bar{x}$  are not in  $\sigma$

# Semantics

**Denotational semantics**; cases predicate abstraction + let:

- $\llbracket (\bar{x}.G) \rrbracket_{\varphi, \sigma}(\bar{t}) = \llbracket G \rrbracket_{\varphi, \sigma}(\{\bar{t}/\bar{x}\})$
- $\llbracket (\bar{x} \mid G) \rrbracket_{\varphi, \sigma}(\bar{t}) = \llbracket G \rrbracket_{\varphi, \{\bar{t}/\bar{x}\}}(\text{id}_{\text{LPV}(\bar{t})})$
- $\llbracket \text{let } \bar{x} = \bar{t} \text{ in } G \rrbracket_{\varphi, \sigma} = \llbracket G \rrbracket_{\varphi, \{\bar{t}/\bar{x}\}} \circ \sigma$

where  $\bar{x}$  are not in  $\sigma$

# Semantics

**Denotational semantics**; cases predicate abstraction + let:

- $\llbracket (\bar{x}.G) \rrbracket_{\varphi, \sigma}(\bar{t}) = \llbracket G \rrbracket_{\varphi, \sigma}(\{\bar{t}/\bar{x}\})$
- $\llbracket (\bar{x} \mid G) \rrbracket_{\varphi, \sigma}(\bar{t}) = \llbracket G \rrbracket_{\varphi, \{\bar{t}/\bar{x}\}}(\text{id}_{\text{LPV}(\bar{t})})$
- $\llbracket \text{let } \bar{x} = \bar{t} \text{ in } G \rrbracket_{\varphi, \sigma} = \llbracket G \rrbracket_{\varphi, \{\bar{t}/\bar{x}\}} \circ \sigma$

where  $\bar{x}$  are not in  $\sigma$

**Lemma:** •  $\llbracket G \rrbracket_{\varphi, \sigma} = \llbracket G\sigma \rrbracket_{\varphi, \emptyset}$

# Semantics

**Denotational semantics**; cases predicate abstraction + let:

- $\llbracket (\bar{x}.G) \rrbracket_{\varphi, \sigma}(\bar{t}) = \llbracket G \rrbracket_{\varphi, \sigma}(\{\bar{t}/\bar{x}\})$
- $\llbracket (\bar{x} \mid G) \rrbracket_{\varphi, \sigma}(\bar{t}) = \llbracket G \rrbracket_{\varphi, \{\bar{t}/\bar{x}\}}(\text{id}_{\text{LPV}(\bar{t})})$
- $\llbracket \text{let } \bar{x} = \bar{t} \text{ in } G \rrbracket_{\varphi, \sigma} = \llbracket G \rrbracket_{\varphi, \{\bar{t}/\bar{x}\} \circ \sigma}$

where  $\bar{x}$  are not in  $\sigma$

- Lemma:**
- $\llbracket G \rrbracket_{\varphi, \sigma} = \llbracket G\sigma \rrbracket_{\varphi, \emptyset}$
  - $\llbracket \text{let } \bar{x} = \bar{t} \text{ in } G \rrbracket_{\varphi, \emptyset} = \llbracket G\{\bar{t}/\bar{x}\} \rrbracket_{\varphi, \emptyset}$

# Semantics

**Denotational semantics**; cases predicate abstraction + let:

- $\llbracket (\bar{x}.G) \rrbracket_{\varphi, \sigma}(\bar{t}) = \llbracket G \rrbracket_{\varphi, \sigma}(\{\bar{t}/\bar{x}\})$
- $\llbracket (\bar{x} \mid G) \rrbracket_{\varphi, \sigma}(\bar{t}) = \llbracket G \rrbracket_{\varphi, \{\bar{t}/\bar{x}\}}(\text{id}_{\text{LPV}(\bar{t})})$
- $\llbracket \text{let } \bar{x} = \bar{t} \text{ in } G \rrbracket_{\varphi, \sigma} = \llbracket G \rrbracket_{\varphi, \{\bar{t}/\bar{x}\}} \circ \sigma$

where  $\bar{x}$  are not in  $\sigma$

- Lemma:**
- $\llbracket G \rrbracket_{\varphi, \sigma} = \llbracket G\sigma \rrbracket_{\varphi, \emptyset}$
  - $\llbracket \text{let } \bar{x} = \bar{t} \text{ in } G \rrbracket_{\varphi, \emptyset} = \llbracket G\{\bar{t}/\bar{x}\} \rrbracket_{\varphi, \emptyset}$
  - $\llbracket (\bar{x}.G)(\bar{t}) \rrbracket_{\varphi, \emptyset} = \llbracket G[\bar{t}/\bar{x}] \rrbracket_{\varphi, \emptyset}$  if  $\text{LPV}(G) \subseteq \{\bar{x}\}$

where  $G[\bar{t}/\bar{x}]$  is obtained by substituting  $t_i$  for  $x_i$  for every free occurrence of  $x_i$  **not inside predicate abstraction**.

# Results

- **L.:** In PL-(2 abs) the predicate abstractions are different from each other and different from functional abstraction:

# Results

- **L.:** In PL-(2 abs) the predicate abstractions are different from each other and different from functional abstraction:
  - $\llbracket (x \mid G)(t) \rrbracket \neq \llbracket (x.G)(t) \rrbracket$  for some  $G, t$

# Results

- **L.:** In PL-(2 abs) the predicate abstractions are different from each other and different from functional abstraction:
  - $\llbracket (x \mid G)(t) \rrbracket \neq \llbracket (x.G)(t) \rrbracket$  for some  $G, t$
  - $\llbracket (x \mid G)(t) \rrbracket \neq \llbracket \text{let } x = t \text{ in } G \rrbracket$  for some  $G, t$

# Results

- **L.:** In PL-(2 abs) the predicate abstractions are different from each other and different from functional abstraction:
  - $\llbracket (x \mid G)(t) \rrbracket \neq \llbracket (x.G)(t) \rrbracket$  for some  $G, t$
  - $\llbracket (x \mid G)(t) \rrbracket \neq \llbracket \text{let } x = t \text{ in } G \rrbracket$  for some  $G, t$
  - $\llbracket (x.G)(t) \rrbracket \neq \llbracket \text{let } x = t \text{ in } G \rrbracket$  for some  $G, t$

# Results

- **L.:** In PL-(2 abs) the predicate abstractions are different from each other and different from functional abstraction:
  - $\llbracket (x \mid G)(t) \rrbracket \neq \llbracket (x.G)(t) \rrbracket$  for some  $G, t$
  - $\llbracket (x \mid G)(t) \rrbracket \neq \llbracket \text{let } x = t \text{ in } G \rrbracket$  for some  $G, t$
  - $\llbracket (x.G)(t) \rrbracket \neq \llbracket \text{let } x = t \text{ in } G \rrbracket$  for some  $G, t$
- **L.:** In PL-(2 abs)-r the forms of abstraction are equivalent:

$$\llbracket (\bar{x} \mid G)(\bar{t}) \rrbracket_{\varphi, \emptyset} = \llbracket (\bar{x}.G)(\bar{t}) \rrbracket_{\varphi, \emptyset} = \llbracket \text{let } \bar{x} = \bar{t} \text{ in } G \rrbracket_{\varphi, \emptyset}$$

# Results

- **L.:** In PL-(2 abs) the predicate abstractions are different from each other and different from functional abstraction:
  - $\llbracket (x \mid G)(t) \rrbracket \neq \llbracket (x.G)(t) \rrbracket$  for some  $G, t$
  - $\llbracket (x \mid G)(t) \rrbracket \neq \llbracket \text{let } x = t \text{ in } G \rrbracket$  for some  $G, t$
  - $\llbracket (x.G)(t) \rrbracket \neq \llbracket \text{let } x = t \text{ in } G \rrbracket$  for some  $G, t$
- **L.:** In PL-(2 abs)-r the forms of abstraction are equivalent:
$$\llbracket (\bar{x} \mid G)(\bar{t}) \rrbracket_{\varphi, \emptyset} = \llbracket (\bar{x}.G)(\bar{t}) \rrbracket_{\varphi, \emptyset} = \llbracket \text{let } \bar{x} = \bar{t} \text{ in } G \rrbracket_{\varphi, \emptyset}$$
- **Th.:** PL-(abs)-r and PL-(2 abs)-r are equivalent.

# Results

- **L.:** In PL-(2 abs) the predicate abstractions are different from each other and different from functional abstraction:
  - $\llbracket (x \mid G)(t) \rrbracket \neq \llbracket (x.G)(t) \rrbracket$  for some  $G, t$
  - $\llbracket (x \mid G)(t) \rrbracket \neq \llbracket \text{let } x = t \text{ in } G \rrbracket$  for some  $G, t$
  - $\llbracket (x.G)(t) \rrbracket \neq \llbracket \text{let } x = t \text{ in } G \rrbracket$  for some  $G, t$
- **L.:** In PL-(2 abs)-r the forms of abstraction are equivalent:

$$\llbracket (\bar{x} \mid G)(\bar{t}) \rrbracket_{\varphi, \emptyset} = \llbracket (\bar{x}.G)(\bar{t}) \rrbracket_{\varphi, \emptyset} = \llbracket \text{let } \bar{x} = \bar{t} \text{ in } G \rrbracket_{\varphi, \emptyset}$$

- **Th.:** PL-(abs)-r and PL-(2 abs)-r are equivalent.
- **Th.:** PL-(2 abs) is more expressive than PL-(2 abs)-r.

# Results

- **L.:** In PL-(2 abs) the predicate abstractions are different from each other and different from functional abstraction:
  - $\llbracket (x \mid G)(t) \rrbracket \neq \llbracket (x.G)(t) \rrbracket$  for some  $G, t$
  - $\llbracket (x \mid G)(t) \rrbracket \neq \llbracket \text{let } x = t \text{ in } G \rrbracket$  for some  $G, t$
  - $\llbracket (x.G)(t) \rrbracket \neq \llbracket \text{let } x = t \text{ in } G \rrbracket$  for some  $G, t$
- **L.:** In PL-(2 abs)-r the forms of abstraction are equivalent:

$$\llbracket (\bar{x} \mid G)(\bar{t}) \rrbracket_{\varphi, \emptyset} = \llbracket (\bar{x}.G)(\bar{t}) \rrbracket_{\varphi, \emptyset} = \llbracket \text{let } \bar{x} = \bar{t} \text{ in } G \rrbracket_{\varphi, \emptyset}$$

- **Th.:** PL-(abs)-r and PL-(2 abs)-r are equivalent.
- **Th.:** PL-(2 abs) is more expressive than PL-(2 abs)-r.

In particular: “var” and “copy” are definable in PL-(2 abs).

# Conclusion

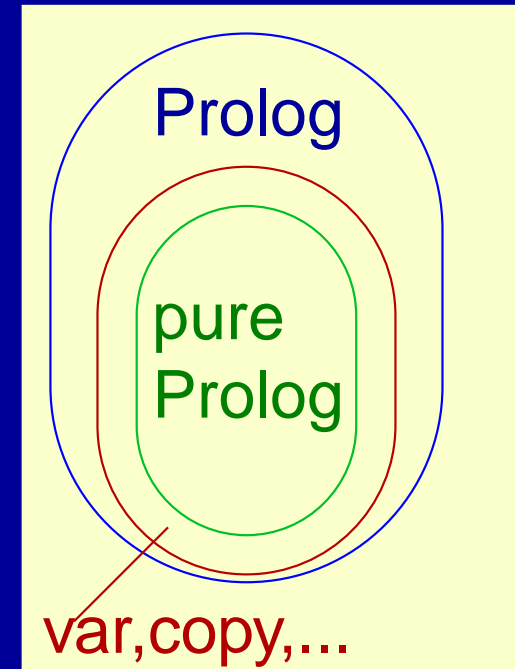
- A term system PL-(2 abs) has been introduced.

# Conclusion

- A term system PL-(2 abs) has been introduced.
- Semantics for this system is given in a denotational style, without referring to operational details.

# Conclusion

- A term system PL-(2 abs) has been introduced.
- Semantics for this system is given in a denotational style, without referring to operational details.
- The system is more expressive than pure Prolog. In particular, some meta-logical predicates are definable.



# Conclusion

- A term system PL-(2 abs) has been introduced.
- Semantics for this system is given in a denotational style, without referring to operational details.
- The system is more expressive than pure Prolog. In particular, some meta-logical predicates are definable.
- The system is not obtained by adding new built-in-predicates or imperative elements but by **generalising predicate abstraction.**

