Declarative Language Extensions for Prolog Courses

Ulrich Neumerkel, Markus Triska
Technische Universität Wien
Austria

Jan Wielemaker
Universiteit van Amsterdam
The Netherlands

1,

Declarative Language Extensions for Prolog Courses

Ulrich Neumerkel, Markus Triska
Technische Universität Wien
Austria

Jan Wielemaker
Universiteit van Amsterdam
The Netherlands

Jan: SWI-Prolog — a popular programming environment

1, 2,

Declarative Language Extensions for Prolog Courses

Ulrich Neumerkel, Markus Triska
Technische Universität Wien
Austria

Jan Wielemaker
Universiteit van Amsterdam
The Netherlands

Jan: SWI-Prolog — a popular programming environment
Markus: CLP(FD) for SWI

1, 2, 3,

Declarative Language Extensions for Prolog Courses

Ulrich Neumerkel, Markus Triska
Technische Universität Wien
Austria

Jan Wielemaker
Universiteit van Amsterdam
The Netherlands

Jan: SWI-Prolog — a popular programming environment
Markus: CLP(FD) for SWI
Ulrich: GUPU — fully automatic environment for Prolog & constraint courses

1, 2, 3, 4,

Declarative Language Extensions for Prolog Courses

Ulrich Neumerkel, Markus Triska
Technische Universität Wien
Austria

Jan Wielemaker
Universiteit van Amsterdam
The Netherlands

Jan: SWI-Prolog — a popular programming environment
Markus: CLP(FD) for SWI
Ulrich: GUPU — fully automatic environment for Prolog & constraint courses

Progress so far:

- (Restricted) side-effect free I/O
- Uniform integer arithmetic
- Sound, safer unification

1, 2, 3, 4, 5

The many facets of Prolog

Logic Programming's Credo:
Algorithm = Logic + Control.

The many facets of Prolog

Logic Programming's Credo:
Algorithm = Logic + Control. Often: Algorithm ≈ Control

1, 2,

The many facets of Prolog

Logic Programming's Credo:

Algorithm = Logic + Control. Often: Algorithm $\approx$ Control $\Rightarrow$ Logic ≈ 0

Why?

1, 2, 3,

The many facets of Prolog

Logic Programming's Credo:

Algorithm = Logic + Control. Often: Algorithm $\approx$ Control $\Rightarrow$ Logic ≈ 0

Why?

- unification wrong ("unsound")
- negation wrong
- side effects inevitable
- moded arithmetic

The many facets of Prolog

Logic Programming's Credo:

Algorithm = Logic + Control. Often: Algorithm $\approx$ Control $\Rightarrow$ Logic ≈ 0

Why?

- unification wrong ("unsound")
- negation wrong
- side effects inevitable
- moded arithmetic

The pure core of Prolog:

1, 2, 3, 4, 5,

The many facets of Prolog

Logic Programming's Credo:

Algorithm = Logic + Control. Often: Algorithm $\approx$ Control $\Rightarrow$ Logic ≈ 0

Why?

- unification wrong ("unsound")
- negation wrong
- side effects inevitable
- moded arithmetic

The pure core of Prolog: monotonic subset

+ no negation
+ no side effects
+ better reasoning (explanations)
+ iterative deepening
+ compatible with constraints

Make this pure core stronger!

1, 2, 3, 4, 5, 6

Pure I/O via DCGs

Nondeterminism blocks general approaches.
(Or: Block nondeterminism)

```
:- use_module(library(pio)).
... --> [] | [_], ... .

?- phrase_from_file((..., "searched", ...), file).
?- phrase_from_file((..., "a" | ..., "b"), file).
?- phrase_from_file((..., ("a"|"b")), file).
```

! similar to monads
! depends on good GC
+ space-efficient
+ efficiency comparable with side-effecting I/O
– not completely side-effect free (one side-effect left)
? extensions to non-seekable devices

Uniform arithmetic

`is/2` vs. `s(X)` vs. constraints (`#=`)

Extending CLP(FD) to CLP(Z) (integer-programming)

```
?- X #>= 7^7^7.
```

Efficiency comparable with is/2 (for comparable cases)

Always terminating

```
?- X#>abs(X).
```

```
?- X#>Y, Y#>X, X#>=0.
```

Necessary to ensure termination of general unification: `?- X = 1.`

Cheap termination proofs for costly labeling:

`?- relation_(X, Zs), false.` terminates

$\Rightarrow$

`?- relation_(X, Zs), labeling([], Zs), false.` terminates.

Implementation in Prolog with attributed variables. No C!

Sound unification

ISO unification: NSTO-property
Two new unification modes with occurs-check

- silent failure (classical sound unification)
- error
  - + locates most STO cases
  - + good for learning/debugging/testing
  - – still inefficiencies

Conclusions

More programs are now monotonic.
Available in current SWI-Prolog distribution.
Adopt it to your systems and courses!