# Better Termination for Prolog with Constraints

Markus Triska, Ulrich Neumerkel
Technische Universität Wien
Austria

Jan Wielemaker
Universiteit van Amsterdam
The Netherlands

1,

# Better Termination for Prolog with Constraints

Markus Triska, Ulrich Neumerkel
Technische Universität Wien
Austria

Jan Wielemaker
Universiteit van Amsterdam
The Netherlands

Long term goal:

1, 2,

# Better Termination for Prolog with Constraints

Markus Triska, Ulrich Neumerkel
Technische Universität Wien
Austria

Jan Wielemaker
Universiteit van Amsterdam
The Netherlands

Long term goal:

Make the pure, monotonic part of Prolog stronger

1, 2, 3,

# Better Termination for Prolog with Constraints

Markus Triska, Ulrich Neumerkel
Technische Universität Wien
Austria

Jan Wielemaker
Universiteit van Amsterdam
The Netherlands

Long term goal:

Make the pure, monotonic part of Prolog stronger

+ iterative deepening
+ compatible with constraints
+ simpler to model/analyze
+ better reasoning (explanations: slices instead of traces)

1, 2, 3, 4, 5,

# Better Termination for Prolog with Constraints

Markus Triska, Ulrich Neumerkel
Technische Universität Wien
Austria

Jan Wielemaker
Universiteit van Amsterdam
The Netherlands

Long term goal:

Make the pure, monotonic part of Prolog stronger

- + iterative deepening
- + compatible with constraints
- + simpler to model/analyze
- + better reasoning (explanations: slices instead of traces)
- + simpler to teach

1, 2, 3, 4, 5, 6

# Better Termination for Prolog with Constraints

Markus Triska, Ulrich Neumerkel
Technische Universität Wien
Austria

Jan Wielemaker
Universiteit van Amsterdam
The Netherlands

Long term goal:

Make the pure, monotonic part of Prolog stronger

+ iterative deepening
+ compatible with constraints
+ simpler to model/analyze
+ better reasoning (explanations: slices instead of traces)
+ simpler to teach

Current progress:

- occurs check reconsidered
- arithmetic as generalized, terminating CLP(FD)

1, 2, 3, 4, 5, 6

# Termination and Nontermination

- Minimal procedural notion
- Connected to declarative notions

# Termination and Nontermination

- Minimal procedural notion
- Connected to declarative notions

- Hard to understand — existential vs. universal termination

- Hard to analyze correctly

Models in $\mathbb{N}$ (cTI)

`?- X = s(Y)` ... $x = 1 + y$

`?- X = s(X)` ... $x = 1 + x$

- Hard to implement — unnecessary nontermination

1, 2

# Sound unification

ISO unification: defined if NSTO (not subject to occurs check).
All other cases *implementation dependent* (= havoc).

1,

# Sound unification

ISO unification: defined if NSTO (not subject to occurs check).
All other cases *implementation dependent* (= havoc).
Definition beyond ISO: Two new unification modes with occurs-check.
Controlled with Prolog flag `occurs_check`:

`true`

+ classical unification
+ difficult to use for real programs
– no direct feedback

`error`, if occurs-check fails

+ locates most STO cases
+ identifies implementation dependent cases
+ good for learning/debugging/testing
– current implementation worst case exp.
– undisciplined change of flag may reveal implementation details

Efficiency better than anticipated. Linear append/3. No overheads for DCGs.

1, 2

# Sound unification — implementation

Desirable properties:

1. `X = X` always succeeds
2. `NewVar = AnyTerm` always succeeds
3. `LocalVar = AnyTerm` always succeeds
4. Does not reveal sharing of terms

- `unify_with_occurs_check(X,X) :- acyclic(X).`
  violates 1,2,3 but agrees with 4
- Robinson-style unification (SWI):
  agrees with 1,2,3 but violates 4

compile time (ECLiPSe-Prolog or manual term expansion)
+ no overhead
– inflexible, recompilation needed to change unification mode
run time (SWI)
+ very small overhead
+ flexible, no recompilation (used with unit testing environment **`plunit`**)

# 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

⇒

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

Implementation in Prolog with attributed variables. No C!

# CLP(FD) - testing

Regression testing

– maintenance high

– produces false alarms for legitimate changes (consistency, operators)

- still inevitable

Observation: Many bugs can be reproduced in small queries

# CLP(FD) - testing

Regression testing

– maintenance high

– produces false alarms for legitimate changes (consistency, operators)

- still inevitable

Observation: Many bugs can be reproduced in small queries

Model based testing

- What model? Reimplementation, another implementation
- oracle required
- conflicts specification vs. implementation
- easily overspecified

Our solution: Take a very small model.

1, 2

# CLP(FD) - testing with a small model

Recent bug:

```
?- [D,E,F,G,H,I] ins -3..3,
   E #= min(F,G-(H+I)),
   D #> 0,
   [A,A,B,C,B,A] = [D,E,F,G,H,I].
```

# CLP(FD) - testing with a small model

Recent bug:

```
?- [D,E,F,G,H,I] ins -3..3,
   E #= min(F,G-(H+I)),
   D #> 0,
   [A,A,B,C,B,A] = [D,E,F,G,H,I].
```

Too complex: Consistency vs. correctness
Simpler approach:

1, 2,

# CLP(FD) - testing with a small model

Recent bug:

```
?- [D,E,F,G,H,I] ins -3..3,
   E #= min(F,G-(H+I)),
   D #> 0,
   [A,A,B,C,B,A] = [D,E,F,G,H,I].
```

Too complex: Consistency vs. correctness
Simpler approach:
`?-` $A$`,` $B$`.` succeeds unconditionally
`?-` $B$`.` fails
⇒ inconsistency

1, 2, 3,

# CLP(FD) - testing with a small model

Recent bug:

```
?- [D,E,F,G,H,I] ins -3..3,
   E #= min(F,G-(H+I)),
   D #> 0,
   [A,A,B,C,B,A] = [D,E,F,G,H,I].
```

Too complex: Consistency vs. correctness
Simpler approach:
`?-` *A*`,` *B*`.` succeeds unconditionally
`?-` *B*`.` fails
⇒ inconsistency
Search for inconsistent pairs! Good search language needed.
+ very robust to changes
+ no false alarms (only hardware errors and resource overflows)
+ would be impossible/very costly with nonterminating CLP(FD)

1, 2, 3, 4

# Conclusions

- More programs terminate
- Programs can be accurately analyzed
- Available in current SWI-Prolog distribution.
- Adopt it to your systems and courses!
- Further step in purification:

  Side-effect free I/O.

  Tomorrow, Saturday at CICLOPS