Mechanisms for side-effect free I/O

Ulrich Neumerkel
TU Wien, Austria

1,

Mechanisms for side-effect free I/O

Ulrich Neumerkel
TU Wien, Austria

Long term goal:
Make the pure, monotonic part of Prolog stronger

1, 2,

Mechanisms for side-effect free I/O

Ulrich Neumerkel
TU Wien, Austria

Long term goal:

Make the pure, monotonic part of Prolog stronger

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

1, 2, 3,

Mechanisms for side-effect free I/O

Ulrich Neumerkel
TU Wien, Austria

Long term goal:

Make the pure, monotonic part of Prolog stronger

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

1, 2, 3, 4,

Mechanisms for side-effect free I/O

Ulrich Neumerkel
TU Wien, Austria

Long term goal:

Make the pure, monotonic part of Prolog stronger

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

1, 2, 3, 4, 5,

Mechanisms for side-effect free I/O

Ulrich Neumerkel
TU Wien, Austria

Long term goal:

Make the pure, monotonic part of Prolog stronger

+ compatible with constraints
+ iterative deepening
+ simpler to model/analyze
+ better reasoning (explanations: slices instead of traces)
+ simpler to teach (GUPU)
+ simpler to test (test for properties)
+ more stable systems

1, 2, 3, 4, 5, 6,

Mechanisms for side-effect free I/O

Ulrich Neumerkel
TU Wien, Austria

Long term goal:

Make the pure, monotonic part of Prolog stronger

+ compatible with constraints
+ iterative deepening
+ simpler to model/analyze
+ better reasoning (explanations: slices instead of traces)
+ simpler to teach (GUPU)
+ simpler to test (test for properties)
+ more stable systems
+ more efficient systems

1, 2, 3, 4, 5, 6, 7

Steps towards side effect free I/O — acknowledgements

(reverse chronological order)

1,

Steps towards side effect free I/O — acknowledgements

(reverse chronological order)

- Vitor Santos Costa, YAP — 2009

1, 2,

Steps towards side effect free I/O — acknowledgements

(reverse chronological order)

- Vitor Santos Costa, YAP — 2009
- Jan Wielemaker, SWI — `library(pio)` 2008

1, 2, 3,

Steps towards side effect free I/O — acknowledgements

(reverse chronological order)

- Vitor Santos Costa, YAP — 2009
- Jan Wielemaker, SWI — `library(pio)` 2008
- Mats Carlson, SICStus — `call_cleanup/2` 1997

1, 2, 3, 4,

Steps towards side effect free I/O — acknowledgements

(reverse chronological order)

- Vitor Santos Costa, YAP — 2009
- Jan Wielemaker, SWI — `library(pio)` 2008
- Mats Carlson, SICStus — `call_cleanup/2` 1997
- Roger Scowen, Klaus Däßler *et mul.*, WG 17 — `ISO/IEC 13211-1` 1995

1, 2, 3, 4, 5,

Steps towards side effect free I/O — acknowledgements

(reverse chronological order)

- Vitor Santos Costa, YAP — 2009
- Jan Wielemaker, SWI — `library(pio)` 2008
- Mats Carlson, SICStus — `call_cleanup/2` 1997
- Roger Scowen, Klaus Däßler *et mul.*, WG 17 — `ISO/IEC 13211-1` 1995
  5 Compliance, 8.11 Stream selection and control

Side-effect free reading

```
phrase_from_file(Phrase, File)

... --> [] | [_], ... .

?- phrase_from_file( ( ..., "searchstring", ... ), file).
```

1,

Side-effect free reading

```
phrase_from_file(Phrase, File)

... --> [] | [_], ... .

?- phrase_from_file( ( ..., "searchstring", ... ), file).
```

Finer control:

```
phrase_of_from_file(Phrase, Reader, File)

phrase_from_file(Phrase, File) :-
   phrase_of_from_file(Phrase, read_pending_input, File).
```

- permits to reuse side-effectful readers like `read/1`
- permits to control buffering on the token-level.

1, 2

Side-effect free reading — Implementation

```
phrase_of_from_file(Ph, Reader, File) :-
  setup_call_cleanup(
    open(File, read, Stream),
    ( reader_to_lazy_list(Reader,Stream, Xs), phrase(Ph, Xs) ),
    close(Stream)).

reader_to_lazy_list(Reader,Stream, Xs) :-
  stream_property(Stream, position(Pos)),
  freeze(Xs, step(Reader,Stream, Pos, Xs)).

step(Reader,Stream, Pos, Xs0) :-
  set_stream_position(Stream, Pos),
  (  at_end_of_stream(Stream)
  -> Xs0 = []
  ;  phrase(call(Reader,Stream), Xs0, Xs),
     reader_to_lazy_list(Reader,Stream, Xs)
  ).
```

Fine print of side-effect free reading

- ISO stream control
- coroutining
- resource control
  - DCGs
  - cleanup

Interactions!

ISO I/O: stream control

- side-effectful
- undervalued

`stream_property/2`, `at_end_of_stream/1`, `set_stream_position/2`

- simplifies generic interfaces: only one predicate required
- extensions to more efficient readers straight forward:

  explicit vs. implicit blocking

```
step(Reader,Stream, Pos, Xs0) :-
  set_stream_position(Stream, Pos),
  (  at_end_of_stream(Stream)
  -> Xs0 = []
  ;  phrase(call(Reader,Stream), Xs0, Xs),
     reader_to_lazy_list(Reader,Stream, Xs)
  ).
```

coroutining — freeze/2

- simple interface, no general unification
- more complex interfaces cannot be controlled
- we need resource control!

coroutining — freeze/2

- simple interface, no general unification
- more complex interfaces cannot be controlled
- we need resource control!

ISO DCGs

- currently in WG17's queue (Paulo Moura)
- hide internal representation, avoid dangling streams, via `phrase/2,3`, `call/3`
  - – required errors
  - – undefined

    STO, setof/3, directives, double opening, errors during side effects
  - – implementation dependent
  - + **implementation defined**
  - . implementation specific
  - – fixed implementation
- guarantee steadfastness

```
?- phrase(a,Xs0,[]).
?- phrase(a,Xs0,Xs), Xs = [].
```

DCGs vs. coroutining

- steadfastness violated

```
?- freeze(Xs0,throw(error)), phrase(a,Xs0,[]).
?- freeze(Xs0,throw(error)), phrase(a,Xs0,Xs), Xs = [].
```

```
a, [_] --> [_].
```

Specify (somehow) order of unification.

```
a([_|A], [_|A]).
```

In SWI:

```
a([_|A], Xs) :- Xs = [_|A].
```

Cleanup mechanism

- currently in WG17's queue
- Cannot be implemented within 13211-1:1995
- Increases robustness — e.g. unrelated errors, interrupts
- Prevents leakage of resources — e.g. connecting to a database
- Must-have for server processes (but don't use `call_cleanup/2`)

`setup_call_cleanup(Setup, Goal, Cleanup)`

- respects nondeterminism of `Goal`
- (Relatively) easy to implement

Current state of adoption:

1. YAP
2. SWI
3. SICStus
4. B-Prolog
5. XSB

- Hard to specify due to non-functional properties

Further plans

- finish DCG codification
- finish `setup_call_cleanup/3` codification
  http://www.complang.tuwien.ac.at/ulrich/iso-prolog/cleanup
- adopt `setup_call_cleanup/3` to further systems
  !!please contact me!!
- side-effect free output: GC-controlled, currently in testing
- 13211-2 (Modules) compatibility (even finer print)
- I/O on unseekable devices