Higher-order Runtime Verification Challenges in (Constraint) Logic Programming

Nataliia Stulova[1], José F. Morales,[1] and Manuel V. Hermenegildo[1,2]

[1]IMDEA Software Institute
[2]School of Computer Science, Technical University of Madrid (UPM)
@ Madrid, Spain

1. Runtime Verification

Runtime verification:

A technique that detects errors in programs, based on:

- Providing a specification for the program.
- Observing the behavior of the running program.
- Detecting (and reacting to) any violations of the specified behavior.

Some research challenges:

- Better formalisms and specification languages.
- Reducing run-time overhead:
  - Optimizing program instrumentation.
  - Combination with static analysis.

2. Higher order in the (C)LP setting

Prolog is a **declarative** programming language where:

- Programs are expressed in terms of relations (facts, rules).
- Computation is initiated by running query on those relations.
- **Higher-order** programming is supported. This allows for:
  - Improving language expressiveness.
  - Code reuse (patterns, templates, etc.).

```
1 min(X,Y,Cmp,M) :- Cmp(R,X,Y), R <= 0, M = X.  % rule
2 min(X,Y,_  ,Y).                               % fact
3
4 less( 0,A,A).              lt('=',A,A).
5 less(-1,A,B) :- A < B.     lt('<',A,B) :- A < B.
6 less( 1,_,_).              lt('>',_,_).
```

- Prolog-based multi-paradigm language.
- Rich assertion language for program specification.
- Availability of both compile- and run-time analysis.

3. Specification for HO Predicates? We Can Do It!

Currently in most (C)LP systems not much can be specified about the *predicate arguments* of higher-order predicates:

```
1 % Predicate assertion format in Ciao:
2 %
3 % :- pred P : Precondition => Postcondition.
4
5 :- pred min(X,Y,Cmp,M) : callable(Cmp).
```

We introduce the notion of "predicate properties (*predprops*):" to apply recursively the assertion language to arguments that contain predicates:

```
1 :- comparator(Cmp) {
2   :- pred Cmp(Res,X,Y) : num(M)    , num(N)    => between(-1,1,Res).
3 }.
4
5 :- pred min(X,Y,Cmp,M) : comparator(Cmp).
```

4. Instrumenting Programs

During the compilation the initial program with specification:

```
1 :- pred Head :  Pre
2              => Post.
3
4
5 Head :- Body_1.
6 Head :- Body_2.
```

is transformed into program with checks, that trigger on each predicate call and exit:

```
1 Head :- precond_chk,
2         NewHead,
3         postcond_chk.
4
5 NewHead :- Body_1.
6 NewHead :- Body_2.
```

while the initial program flow is preserved.

5. How it works

```
1  call: min(1,3,less,M)
2    precond_chk(callable(less))
3    call: less(Res,1,3)
4      call: 1 < 3
5      exit: 1 < 3
6    exit: less( -1,1,3)  % 2nd rule
7
8    call: -1 <= 0
9    exit: -1 <= 0
10 exit: min(1,3,less,1)
11 %---------------------------------------%
12 call: min(1,3,lt,M)
13   precond_chk(callable(lt))
14   call: lt(Res,1,3)
15     call: 1 < 3
16     exit: 1 < 3
17   exit: lt('<',1,3) % 2nd rule
18 % -------------------- NO PROBLEMS DETECTED
19   call: '<' <= 0
20   fail: '<' <= 0
21   ... % backtracking
22   precond_chk(callable(lt))
23   redo: lt('<',1,3)
24   exit: lt('>'1,3) % 3rd rule
25 % -------------------- NO PROBLEMS DETECTED
26   call: '>' <= 0
27   fail: '>' <= 0
28 fail: min(1,3,lt,M)
```

```
1  call: min(1,3,less,M)
2    precond_chk((num(1),num(3))
3    call: less(Res,1,3)
4      call: 1 < 3
5      exit: 1 < 3
6    exit: less( -1,1,3)  % 2nd rule
7    postcond_chk(between(-1,1,-1))
8    call: -1 <= 0
9    exit: -1 <= 0
10 exit: min(1,3,less,1)
11 %---------------------------------------%
12 call: min(1,3,lt,M)
13   precond_chk((num(1),num(3))
14   call: lt(Res,1,3)
15     call: 1 < 3
16     exit: 1 < 3
17   exit: lt('<',1,3) % 2nd rule
18   postcond_chk(between(-1,1,'<'))
19   call: '<' <= 0
20   fail: '<' <= 0
21   ... % backtracking
22   precond_chk((num(1),num(3))
23   redo: lt('<',1,3)
24   exit: lt('>'1,3) % 3rd rule
25   postcond_chk(between(-1,1,'>'))
26   call: '>' <= 0
27   fail: '>' <= 0
28 fail: min(1,3,lt,M)
```

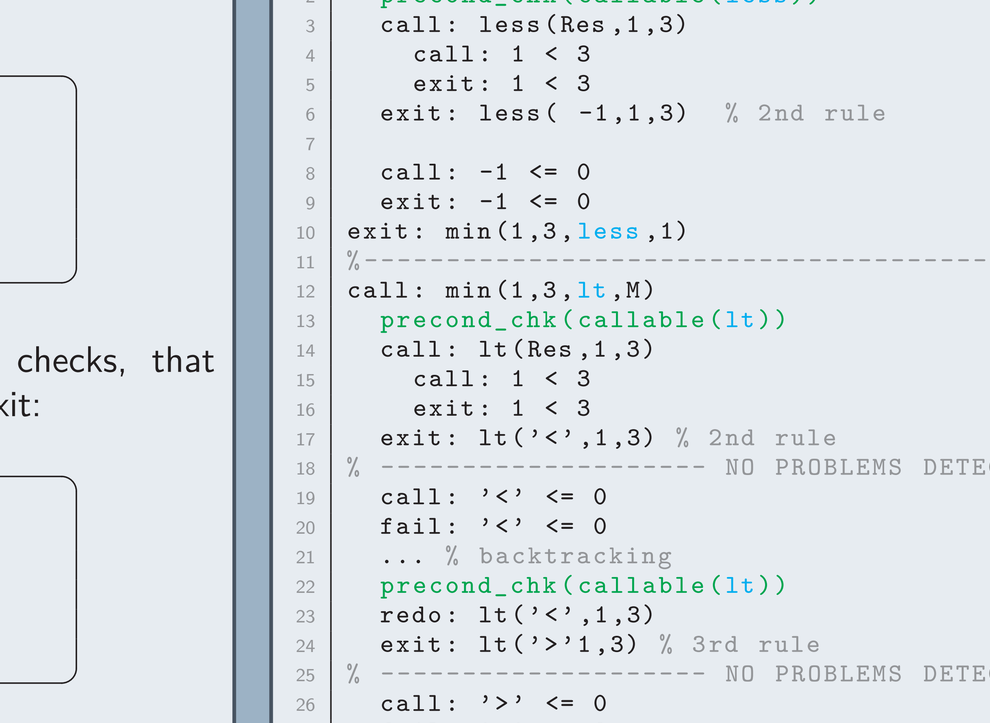
6. Current/Future Work

Work in progress:

- Improving Ciao program instrumentation.
- Reducing run-time check overhead by caching checks.

Future work:

- Extend static analysis-based check simplification during compilation to the higher-order case.
- Reducing overhead for higher-order property checks.



7. Quick Links

ciao-lang.org

CLIP group

clip.dia.fi.upm.es

software.imdea.org

dblp computer science bibliography

(publications)