Clingo = ASP + Control

Martin Gebser Roland Kaminski Benjamin Kaufmann
Torsten Schaub

University of Potsdam

Outline

Potassco

Outline

Potassco

Clingo species

Clingo = *gringo* | *clasp*

- Before
  - *Clingo* — easy solving
    iClingo — incremental solving
    oClingo — reactive solving
- After
  - *Clingo* — easy solving
    + incremental solving
    + reactive solving
    + complex solving
- *Clingo* series 4 = ASP + Control

Potassco

Clingo species

- Before
  - *Clingo* — easy solving
  - *iClingo* — incremental solving
  - *oClingo* — reactive solving
- After

  Clingo — easy solving
  + incremental solving
  + reactive solving
  + complex solving
- *Clingo* series 4 = ASP + Control

Clingo = *gringo* | *clasp*

Potassco

Clingo species

- Before
  - *Clingo* — easy solving
  - *iClingo* — incremental solving
  - *oClingo* — reactive solving
- After

  Clingo — easy solving
  + incremental solving
  + reactive solving
  + complex solving

- *Clingo* series 4 = ASP + Control

Clingo = *gringo* | *clasp*

Potassco

Clingo species

- Before
  - *Clingo* — easy solving
  - *iClingo* — incremental solving
  - *oClingo* — reactive solving
- After

  Clingo — easy solving
  + incremental solving
  + reactive solving
  + complex solving
- *Clingo* series 4 = ASP + Control

Clingo = *gringo* | *clasp*

Potassco

Clingo species

- Before
  - *Clingo* — easy solving
  - *iClingo* — incremental solving
  - *oClingo* — reactive solving
- After
  - *Clingo* — easy solving
  - + incremental solving
  - + reactive solving
  - + complex solving
- *Clingo* series 4 = ASP + Control

Clingo = *gringo* | *clasp*

Potassco

Clingo species

- Before

  Clingo = *gringo* | *clasp*

  - *Clingo* — easy solving
  - *iClingo* — incremental solving
  - *oClingo* — reactive solving

- After

  Clingo = *gringo* | *clasp*

  - *Clingo* — easy solving
  - + incremental solving
  - + reactive solving
  - + complex solving

- *Clingo* series 4 = ASP + Control

Potassco

Clingo species

- Before

  $Clingo = gringo \mid clasp$

  - *Clingo* — easy solving
  - *iClingo* — incremental solving
  - *oClingo* — reactive solving
- After

  $Clingo = gringo^* \mid clasp^*$

  - *Clingo* — easy solving
  - + incremental solving
  - + reactive solving
  - + complex solving
- *Clingo* series 4 = ASP + Control

Potassco

Clingo species

- Before

  $Clingo \;=\; gringo \mid clasp$

  - *Clingo* — easy solving
  - *iClingo* — incremental solving
  - *oClingo* — reactive solving
- After

  $Clingo \;=\; (gringo^* \mid clasp^*)^*$

  - *Clingo* — easy solving
  - + incremental solving
  - + reactive solving
  - + complex solving
- *Clingo* series 4 = ASP + Control

Potassco

Clingo species

- Before

  $Clingo \;=\; gringo \mid clasp$

  - *Clingo* — easy solving
  - *iClingo* — incremental solving
  - *oClingo* — reactive solving
- After

  $Clingo \;=\; (gringo^* \mid clasp^*)^*$

  - *Clingo* — easy solving
  - $+$ incremental solving
  - $+$ reactive solving
  - $+$ complex solving
- *Clingo* series 4 $=$ ASP $+$ Control

Potassco

Clingo species

- Before

  $Clingo \;=\; gringo \mid clasp$

  - *Clingo* — easy solving
  - *iClingo* — incremental solving
  - *oClingo* — reactive solving
- After

  $Clingo \;=\; (gringo^* \mid clasp^*)^*$

  - *Clingo* — easy solving
  - $+$ incremental solving
  - $+$ reactive solving
  - $+$ complex solving
- *Clingo* series 4 $=$ ASP $+$ Control

Potassco

Outline

Potassco

Clingo series 4

- ASP
  - `#program <name> [ (<parameters>) ]`
    - Example `#program play(t).`
    - Default `#program base.`
  - `#external <atom> [ : <body> ]`
    - Example `#external mark(X,Y,P,t) : field(X,Y), player(P).`
  - Grounding yields a pair (P, E) of ground rules and ground atoms, whose semantics is captured by module theory
- Control
  - Lua
    - Example `prg:solve(), prg:ground(parts), ...`
  - Python
    - Example `prg.solve(), prg.ground(parts), ...`

  embedded scripting language (`#script`)
  libraries

Potassco

Clingo series 4

- ASP
  - `#program <name> [ (<parameters>) ]`
    - Example `#program play(t).`
    - Default `#program base.`
  - `#external <atom> [ : <body> ]`
    - Example `#external mark(X,Y,P,t) : field(X,Y), player(P).`
  - Grounding yields a pair (P, E) of ground rules and ground atoms, whose semantics is captured by module theory
- Control
  - Lua
    - Example `prg:solve(), prg:ground(parts), ...`
  - Python
    - Example `prg.solve(), prg.ground(parts), ...`

  embedded scripting language (`#script`)
  libraries

Potassco

Clingo series 4

- ASP
  - `#program <name> [ (<parameters>) ]`
    - Example `#program play(t).`
    - Default `#program base.`
  - `#external <atom> [ : <body> ]`
    - Example `#external mark(X,Y,P,t) : field(X,Y), player(P).`
  - Grounding yields a pair (P, E) of ground rules and ground atoms, whose semantics is captured by module theory
- Control
  - Lua
    - Example `prg:solve(), prg:ground(parts), ...`
  - Python
    - Example `prg.solve(), prg.ground(parts), ...`

  embedded scripting language (`#script`)
  libraries

Potassco

Clingo series 4

- ASP
  - `#program <name> [ (<parameters>) ]`
    - Example `#program play(t).`
    - Default `#program base.`
  - `#external <atom> [ : <body> ]`
    - Example `#external mark(X,Y,P,t) : field(X,Y), player(P).`
  - Grounding yields a pair (P, E) of ground rules and ground atoms, whose semantics is captured by module theory
- Control
  - Lua
    - Example `prg:solve(), prg:ground(parts), ...`
  - Python
    - Example `prg.solve(), prg.ground(parts), ...`
  - embedded scripting language (`#script`)
  - libraries

Potassco

Clingo series 4

- ASP
  - `#program <name> [ (<parameters>) ]`
    - Example `#program play(t).`
    - Default `#program base.`
  - `#external <atom> [ : <body> ]`
    - Example `#external mark(X,Y,P,t) : field(X,Y), player(P).`
  - Grounding yields a pair (P, E) of ground rules and ground atoms, whose semantics is captured by module theory
- Control
  - Lua
    - Example `prg:solve(), prg:ground(parts), ...`
  - Python
    - Example `prg.solve(), prg.ground(parts), ...`
  - embedded scripting language (`#script`)
  - libraries

Potassco

Vanilla *Clingo*

- Emulating *Clingo* in *Clingo* 4

```
#script (python)
def main(prg):
    parts = []
    parts.append(("base", []))
    prg.ground(parts)
    prg.solve()
#end.
```

Potassco

Vanilla *Clingo*

- Emulating *Clingo* in *Clingo* 4

```
#script (python)
def main(prg):
    parts = []
    parts.append(("base", []))
    prg.ground(parts)
    prg.solve()
#end.
```

Vanilla *Clingo*

- Emulating *Clingo* in *Clingo* 4

```
#script (python)
def main(prg):
    parts = []
    parts.append(("base", []))
    prg.ground(parts)
    prg.solve()
#end.
```

Potassco

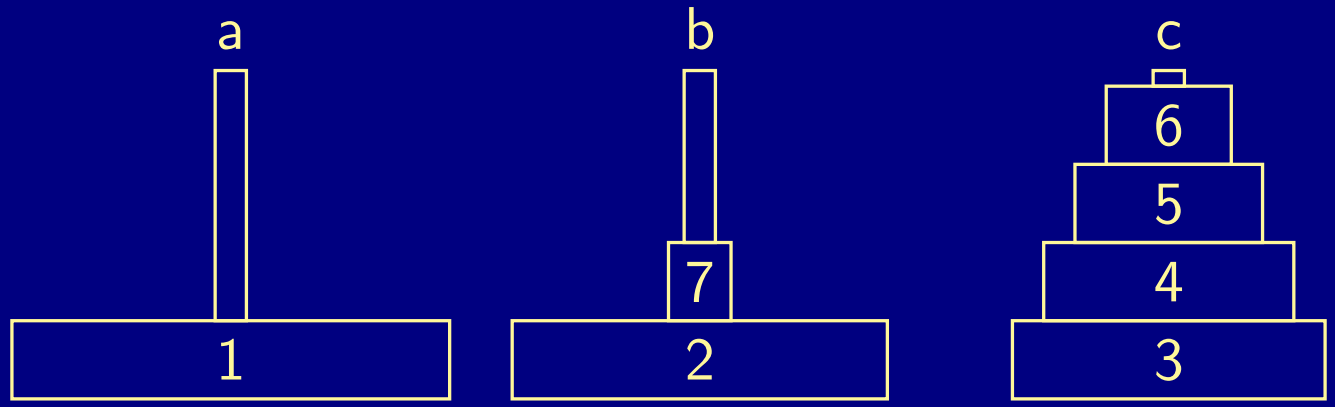
Outline

Potassco

Towers of Hanoi Instance

- Emulating *iClingo* in *Clingo* 4
  - Incremental grounding
  - Incremental solving

Towers of Hanoi Instance




```
peg(a;b;c).
init_on(1,a).
init_on((2;7),b).
init_on((3;4;5;6),c).
```

```
disk(1..7).

goal_on((3;4),a).
goal_on((1;2;5;6;7),c).
```

Potassco

Towers of Hanoi Encoding (base)

```
#program base.

on(D,P,0) :- init_on(D,P).
```

Towers of Hanoi Encoding (cumulative)

```
#program cumulative(t).

1 { move(D,P,t) : disk(D), peg(P) } 1.

moved(D,t) :- move(D,_,t).
blocked(D,P,t) :- on(D+1,P,t-1), disk(D).
blocked(D,P,t) :- blocked(D+1,P,t), disk(D).
:- move(D,P,t), blocked(D-1,P,t).
:- moved(D,t), on(D,P,t-1), blocked(D,P,t).

on(D,P,t) :- on(D,P,t-1), not moved(D,t).
on(D,P,t) :- move(D,P,t).
:- not 1 { on(D,P,t) : peg(P) } 1, disk(D).
```

Potassco

Towers of Hanoi Encoding (volatile)

```
#program volatile(t).

#external query(t).

:- goal_on(D,P), not on(D,P,t), query(t).
```

Incremental Solving (embedded)

```
#script (python)

from gringo import SolveResult, Fun

def main(prg):
    ret, parts, step = SolveResult.UNSAT, [], 1
    parts.append(("base", []))
    while ret == SolveResult.UNSAT:
        parts.append(("cumulative", [step]))
        parts.append(("volatile", [step]))
        prg.ground(parts)
        prg.release_external(Fun("query", [step-1]))
        prg.assign_external(Fun("query", [step]), True)
        ret, parts, step = prg.solve(), [], step+1

#end.
```

Potassco

Incremental Solving (library)

```
from sys import stdout
from gringo import SolveResult, Fun, Control

prg = Control()
prg.load("toh.lp")

ret, parts, step = SolveResult.UNSAT, [], 1
parts.append(("base", []))
while ret == SolveResult.UNSAT:
    parts.append(("cumulative", [step]))
    parts.append(("volatile", [step]))
    prg.ground(parts)
    prg.release_external(Fun("query", [step-1]))
    prg.assign_external(Fun("query", [step]), True)
    f = lambda m: stdout.write(str(m))
    ret, parts, step = prg.solve(on_model=f), [], step+1
```

Potassco

Outline

Potassco

Summary

- ASP is an under-the-hood technology !
- *Clingo* 4
  - Multi-shot ASP solving
    - Continously changing programs
  - Opens up new areas of applications
    - Agents
    - Assisted Living
    - Robotics
    - Planning
    - Interaction
    - Query-answering
    - etc
- `http://potassco.sourceforge.net`

Potassco

Summary

- ASP is an under-the-hood technology !
- *Clingo* 4
  - Multi-shot ASP solving
    - Continously changing programs
  - Opens up new areas of applications
    - Agents
    - Assisted Living
    - Robotics
    - Planning
    - Interaction
    - Query-answering
    - etc
- `http://potassco.sourceforge.net`

Potassco

Summary

- ASP is an under-the-hood technology !
- *Clingo* 4
  - Multi-shot ASP solving
    - Continously changing programs
  - Opens up new areas of applications
    - Agents
    - Assisted Living
    - Robotics
    - Planning
    - Interaction
    - Query-answering
    - etc
- `http://potassco.sourceforge.net`

Potassco