From SAT to ASP and back!?

Torsten Schaub

University of Potsdam & INRIA Rennes

Outline

Potassco

Outline

Potassco

Answer Set Programming (ASP)
in a Nutshell

- ASP is an approach to declarative problem solving, combining
  - a rich yet simple modeling language
  - with effective solving capacities
- tailored to knowledge representation and reasoning
- ASP has its roots in
  - databases
  - logic programming
  - knowledge representation and (non-monotonic) reasoning
  - constraint solving (in particular, SAT)
- ASP allows for solving all search problems in NP (and NP^{NP})
- The versatility of ASP is reflected by the solver *clasp*, winning first places at ASP, CASC, MISC, PB, and SAT
- ASP embraces many emerging application areas

Potassco

Answer Set Programming (ASP) *in a Nutshell*

- ASP is an approach to declarative problem solving, combining
  - a rich yet simple modeling language
  - with effective solving capacities

  tailored to knowledge representation and reasoning
- ASP has its roots in
  - databases
  - logic programming
  - knowledge representation and (non-monotonic) reasoning
  - constraint solving (in particular, SAT)
- ASP allows for solving all search problems in NP (and NP^{NP})
- The versatility of ASP is reflected by the solver *clasp*, winning first places at ASP, CASC, MISC, PB, and SAT
- ASP embraces many emerging application areas

Potassco

Answer Set Programming (ASP)
in a Nutshell

- ASP is an approach to declarative problem solving, combining
  - a rich yet simple modeling language
  - with effective solving capacities

  tailored to knowledge representation and reasoning
- ASP has its roots in
  - databases
  - logic programming
  - knowledge representation and (non-monotonic) reasoning
  - constraint solving (in particular, SAT)
- ASP allows for solving all search problems in NP (and NP^{NP})
- The versatility of ASP is reflected by the solver *clasp*, winning first places at ASP, CASC, MISC, PB, and SAT
- ASP embraces many emerging application areas

Potassco

Answer Set Programming (ASP) *in a Nutshell*

- ASP is an approach to declarative problem solving, combining
  - a rich yet simple modeling language
  - with effective solving capacities

  tailored to knowledge representation and reasoning
- ASP has its roots in
  - databases
  - logic programming
  - knowledge representation and (non-monotonic) reasoning
  - constraint solving (in particular, SAT)
- ASP allows for solving all search problems in NP (and NP^{NP})
- The versatility of ASP is reflected by the solver *clasp*, winning first places at ASP, CASC, MISC, PB, and SAT
- ASP embraces many emerging application areas

Potassco

Answer Set Programming (ASP) *in a Nutshell*

- ASP is an approach to declarative problem solving, combining
  - a rich yet simple modeling language
  - with effective solving capacities

  tailored to knowledge representation and reasoning
- ASP has its roots in
  - databases
  - logic programming
  - knowledge representation and (non-monotonic) reasoning
  - constraint solving (in particular, SAT)
- ASP allows for solving all search problems in NP (and NP^{NP})
- The versatility of ASP is reflected by the solver *clasp*, winning first places at ASP, CASC, MISC, PB, and SAT
- ASP embraces many emerging application areas

Potassco

Answer Set Programming (ASP) *in a Nutshell*

- ASP is an approach to declarative problem solving, combining
  - a rich yet simple modeling language
  - with effective solving capacities

  tailored to knowledge representation and reasoning
- ASP has its roots in
  - databases
  - logic programming
  - knowledge representation and (non-monotonic) reasoning
  - constraint solving (in particular, SAT)
- ASP allows for solving all search problems in NP (and NP^{NP})
- The versatility of ASP is reflected by the solver *clasp*, winning first places at ASP, CASC, MISC, PB, and SAT
- ASP embraces many emerging application areas

Potassco

Outline

Potassco

Some biased moments in time

- '70/'80 Capturing incomplete information

Some biased moments in time

- '70/'80 Capturing incomplete information
  - Databases Closed world assumption
  - Logic programming Negation as failure
  - Non-monotonic reasoning
    Auto-epistemic and Default logics, Circumscription

Some biased moments in time

- '70/'80 Capturing incomplete information
  - Databases Closed world assumption
    - Axiomatic characterization
  - Logic programming Negation as failure
  - Non-monotonic reasoning
    Auto-epistemic and Default logics, Circumscription

Potassco

Some biased moments in time

- '70/'80 Capturing incomplete information
  - Databases Closed world assumption
    - Axiomatic characterization
  - Logic programming Negation as failure
    - Herbrand interpretations
    - Fix-point characterizations
  - Non-monotonic reasoning
    Auto-epistemic and Default logics, Circumscription

Potassco

Some biased moments in time

- '70/'80 Capturing incomplete information
  - Databases Closed world assumption
    - Axiomatic characterization
  - Logic programming Negation as failure
    - Herbrand interpretations
    - Fix-point characterizations
  - Non-monotonic reasoning
    Auto-epistemic and Default logics, Circumscription
    - Extensions of first-order logic
    - Modalities, fix-points, second-order logic

Potassco

Some biased moments in time

- '70/'80 Capturing incomplete information
  - Databases Closed world assumption
  - Logic programming Negation as failure
  - Non-monotonic reasoning
    Auto-epistemic and Default logics, Circumscription
- '90 Amalgamation and computation

Some biased moments in time

- '70/'80 Capturing incomplete information
  - Databases Closed world assumption
  - Logic programming Negation as failure
  - Non-monotonic reasoning
    Auto-epistemic and Default logics, Circumscription
- '90 Amalgamation and computation
  - Logic programming semantics
    Well-founded and stable models semantics
  - ASP solving
    "Stable models = Well-founded semantics + Branch"

Potassco

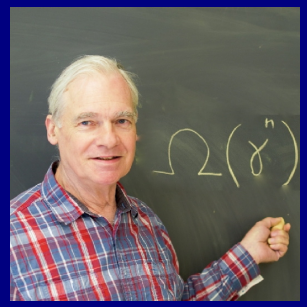
Some biased moments in time

- '70/'80 Capturing incomplete information
  - Databases Closed world assumption
  - Logic programming Negation as failure
  - Non-monotonic reasoning
    Auto-epistemic and Default logics, Circumscription
- '90 Amalgamation and computation
  - Logic programming semantics
    Well-founded and stable models semantics
    - Stable models semantics derived from non-monotonic logics
    - Alternating fix-point theory
  - ASP solving
    "Stable models = Well-founded semantics + Branch"

Potassco

Some biased moments in time

- '70/'80 Capturing incomplete information
  - Databases Closed world assumption
  - Logic programming Negation as failure
  - Non-monotonic reasoning
    Auto-epistemic and Default logics, Circumscription
- '90 Amalgamation and computation
  - Logic programming semantics
    Well-founded and stable models **semantics**
    - Stable models semantics derived from non-monotonic logics
    - **Alternating fix-point theory**
  - ASP solving
    "Stable models = Well-founded semantics + Branch"



Potassco

Some biased moments in time

- '70/'80 Capturing incomplete information
  - Databases Closed world assumption
  - Logic programming Negation as failure
  - Non-monotonic reasoning
    Auto-epistemic and Default logics, Circumscription
- '90 Amalgamation and computation
  - Logic programming semantics
    Well-founded and stable models semantics
    - Stable models semantics derived from non-monotonic logics
    - Alternating fix-point theory
  - ASP solving
    "Stable models = Well-founded semantics + Branch"
    - Modeling — Grounding — Solving
    - Icebreakers: `lparse` and `smodels`

Potassco

Some biased moments in time

- '70/'80 Capturing incomplete information
  - Databases Closed world assumption
  - Logic programming Negation as failure
  - Non-monotonic reasoning
    Auto-epistemic and Default logics, Circumscription
- '90 Amalgamation and computation
  - Logic programming semantics
    Well-founded and stable models semantics
  - ASP solving
    "Stable models = Well-founded semantics + Branch"
- '00 Applications and semantic rediscoveries

Potassco

Some biased moments in time

- '70/'80 Capturing incomplete information
  - Databases Closed world assumption
  - Logic programming Negation as failure
  - Non-monotonic reasoning
    Auto-epistemic and Default logics, Circumscription
- '90 Amalgamation and computation
  - Logic programming semantics
    Well-founded and stable models semantics
  - ASP solving
    "Stable models = Well-founded semantics + Branch"
- '00 Applications and semantic rediscoveries
  - Growing dissemination Decision Support for Space Shuttle
  - Constructive logics Equilibrium Logic

Potassco

Some biased moments in time

- '70/'80 Capturing incomplete information
  - Databases Closed world assumption
  - Logic programming Negation as failure
  - Non-monotonic reasoning
    Auto-epistemic and Default logics, Circumscription
- '90 Amalgamation and computation
  - Logic programming semantics
    Well-founded and stable models semantics
  - ASP solving
    "Stable models = Well-founded semantics + Branch"
- '00 Applications and semantic rediscoveries
  - Growing dissemination Decision Support for Space Shuttle
    - Bio-informatics, Linux Package Configuration, Music composition, Robotics, System Design, etc
  - Constructive logics Equilibrium Logic

Potassco

Some biased moments in time

- '70/'80 Capturing incomplete information
  - Databases Closed world assumption
  - Logic programming Negation as failure
  - Non-monotonic reasoning
    Auto-epistemic and Default logics, Circumscription
- '90 Amalgamation and computation
  - Logic programming semantics
    Well-founded and stable models semantics
  - ASP solving
    "Stable models = Well-founded semantics + Branch"
- '00 Applications and semantic rediscoveries
  - Growing dissemination Decision Support for Space Shuttle
  - Constructive logics Equilibrium Logic
    - Roots: Logic of Here-and-There (Heyting'30), G3 (Gödel'32)

Potassco

Some biased moments in time

- '70/'80 Capturing incomplete information
  - Databases Closed world assumption
  - Logic programming Negation as failure
  - Non-monotonic reasoning
    Auto-epistemic and Default logics, Circumscription
- '90 Amalgamation and computation
  - Logic programming semantics
    Well-founded and stable models semantics
  - ASP solving
    "Stable models = Well-founded semantics + Branch"
- '00 Applications and semantic rediscoveries
  - Growing dissemination Decision Support for Space Shuttle
  - Constructive logics Equilibrium Logic
- '10 Integration

Potassco

Some biased moments in time

- '70/'80 Capturing incomplete information
  - Databases Closed world assumption
  - Logic programming Negation as failure
  - Non-monotonic reasoning
    Auto-epistemic and Default logics, Circumscription
- '90 Amalgamation and computation
  - Logic programming semantics
    Well-founded and stable models semantics
  - ASP solving
    "Stable models = Well-founded semantics + Branch"
- '00 Applications and semantic rediscoveries
  - Growing dissemination Decision Support for Space Shuttle
  - Constructive logics Equilibrium Logic
- '10 Integration — let's see . . .

Potassco

Some biased moments in time

- '70/'80 Capturing incomplete information
  - Databases Closed world assumption
  - Logic programming Negation as failure
  - Non-monotonic reasoning
    Auto-epistemic and Default logics, Circumscription
- '90 Amalgamation and computation
  - Logic programming semantics
    Well-founded and stable models semantics
  - ASP solving
    "Stable models = Well-founded semantics + Branch"
- '00 Applications and semantic rediscoveries
  - Growing dissemination Decision Support for Space Shuttle
  - Constructive logics Equilibrium Logic
- '10 Integration

Potassco

Outline

Potassco

Propositional Normal Logic Programs

- A logic program Π is a set of rules of the form

$$\underbrace{a}_{\text{head}} \leftarrow \underbrace{b_1, \ldots, b_m, \neg c_1, \ldots, \neg c_n}_{\text{body}}$$

  - a and all b_i, c_j are atoms (propositional variables)
  - $\leftarrow$, ,, $\neg$ denote if, and, and (default) negation
  - intuitive reading: head must be true if body holds

- Semantics given by stable models, informally,
  models of Π justifying each true atom by some rule in Π

Potassco

Logic Programs

- A logic program Π is a set of rules of the form

$$\underbrace{a}_{\text{head}} \leftarrow \underbrace{b_1, \ldots, b_m, \neg c_1, \ldots, \neg c_n}_{\text{body}}$$

  - a and all b_i, c_j are atoms (propositional variables)
  - $\leftarrow$, ,, $\neg$ denote if, and, and (default) negation
  - intuitive reading: head must be true if body holds

- Semantics given by stable models, informally, models of Π justifying each true atom by some rule in Π

Logic Programs

- A logic program Π is a set of rules of the form

$$\underbrace{a}_{\text{head}} \leftarrow \underbrace{b_1, \ldots, b_m, \neg c_1, \ldots, \neg c_n}_{\text{body}}$$

  - a and all b_i, c_j are atoms (propositional variables)
  - $\leftarrow$, ,, $\neg$ denote if, and, and (default) negation
  - intuitive reading: head must be true if body holds

- Semantics given by stable models, informally,
  models of Π justifying each true atom by some rule in Π

Normal Logic Programs

- A logic program Π is a set of rules of the form

$$\underbrace{a}_{\text{head}} \leftarrow \underbrace{b_1, \ldots, b_m, \neg c_1, \ldots, \neg c_n}_{\text{body}}$$

  - a and all b_i, c_j are atoms (propositional variables)
  - $\leftarrow$, ,, $\neg$ denote if, and, and (default) negation
  - intuitive reading: head must be true if body holds

- Semantics given by stable models, informally,
  models of Π justifying each true atom by some rule in Π

- Disclaimer The following formalities apply to normal logic programs

Potassco

Some truth tabling, back to SAT

a	b	c	$(\neg b \rightarrow a) \wedge (b \rightarrow c)$
F	F	F	
F	F	T	
F	T	F	
F	T	T	
T	F	F	
T	F	T	
T	T	F	
T	T	T	

Some truth tabling, back to SAT

a	b	c	$(\neg b \rightarrow a) \wedge (b \rightarrow c)$
F	**F**	**F**	$(\neg\mathbf{F} \rightarrow \mathbf{F}) \wedge (\mathbf{F} \rightarrow \mathbf{F})$
F	**F**	**T**	$(\neg\mathbf{F} \rightarrow \mathbf{F}) \wedge (\mathbf{F} \rightarrow \mathbf{T})$
F	**T**	**F**	$(\neg\mathbf{T} \rightarrow \mathbf{F}) \wedge (\mathbf{T} \rightarrow \mathbf{F})$
F	**T**	**T**	$(\neg\mathbf{T} \rightarrow \mathbf{F}) \wedge (\mathbf{T} \rightarrow \mathbf{T})$
T	**F**	**F**	$(\neg\mathbf{F} \rightarrow \mathbf{T}) \wedge (\mathbf{F} \rightarrow \mathbf{F})$
T	**F**	**T**	$(\neg\mathbf{F} \rightarrow \mathbf{T}) \wedge (\mathbf{F} \rightarrow \mathbf{T})$
T	**T**	**F**	$(\neg\mathbf{T} \rightarrow \mathbf{T}) \wedge (\mathbf{T} \rightarrow \mathbf{F})$
T	**T**	**T**	$(\neg\mathbf{T} \rightarrow \mathbf{T}) \wedge (\mathbf{T} \rightarrow \mathbf{T})$

Potassco

Some truth tabling, back to SAT

a	b	c	$(\neg b \rightarrow a) \wedge (b \rightarrow c)$
F	**F**	**F**	$(\mathbf{T} \rightarrow \mathbf{F}) \wedge (\mathbf{F} \rightarrow \mathbf{F})$
F	**F**	**T**	$(\mathbf{T} \rightarrow \mathbf{F}) \wedge (\mathbf{F} \rightarrow \mathbf{T})$
F	**T**	**F**	$(\mathbf{F} \rightarrow \mathbf{F}) \wedge (\mathbf{T} \rightarrow \mathbf{F})$
F	**T**	**T**	$(\mathbf{F} \rightarrow \mathbf{F}) \wedge (\mathbf{T} \rightarrow \mathbf{T})$
T	**F**	**F**	$(\mathbf{T} \rightarrow \mathbf{T}) \wedge (\mathbf{F} \rightarrow \mathbf{F})$
T	**F**	**T**	$(\mathbf{T} \rightarrow \mathbf{T}) \wedge (\mathbf{F} \rightarrow \mathbf{T})$
T	**T**	**F**	$(\mathbf{F} \rightarrow \mathbf{T}) \wedge (\mathbf{T} \rightarrow \mathbf{F})$
T	**T**	**T**	$(\mathbf{F} \rightarrow \mathbf{T}) \wedge (\mathbf{T} \rightarrow \mathbf{T})$

Potassco

Some truth tabling, back to SAT

a	b	c	$(\neg b \rightarrow a) \wedge (b \rightarrow c)$
F	F	F	$(\mathbf{T} \rightarrow \mathbf{F}) \wedge (\mathbf{F} \rightarrow \mathbf{F})$
F	F	T	$(\mathbf{T} \rightarrow \mathbf{F}) \wedge (\mathbf{F} \rightarrow \mathbf{T})$
F	T	F	$(\mathbf{F} \rightarrow \mathbf{F}) \wedge (\mathbf{T} \rightarrow \mathbf{F})$
F	T	T	$(\mathbf{F} \rightarrow \mathbf{F}) \wedge (\mathbf{T} \rightarrow \mathbf{T})$
T	F	F	$(\mathbf{T} \rightarrow \mathbf{T}) \wedge (\mathbf{F} \rightarrow \mathbf{F})$
T	F	T	$(\mathbf{T} \rightarrow \mathbf{T}) \wedge (\mathbf{F} \rightarrow \mathbf{T})$
T	T	F	$(\mathbf{F} \rightarrow \mathbf{T}) \wedge (\mathbf{T} \rightarrow \mathbf{F})$
T	T	T	$(\mathbf{F} \rightarrow \mathbf{T}) \wedge (\mathbf{T} \rightarrow \mathbf{T})$

Potassco

Some truth tabling, back to SAT

a	b	c	$(\neg b \rightarrow a) \wedge (b \rightarrow c)$
F	**F**	**F**	$\mathbf{F} \wedge (\mathbf{F} \rightarrow \mathbf{F})$
F	**F**	**T**	$\mathbf{F} \wedge (\mathbf{F} \rightarrow \mathbf{T})$
F	**T**	**F**	$(\mathbf{F} \rightarrow \mathbf{F}) \wedge \mathbf{F}$
F	**T**	**T**	$(\mathbf{F} \rightarrow \mathbf{F}) \wedge (\mathbf{T} \rightarrow \mathbf{T})$
T	**F**	**F**	$(\mathbf{T} \rightarrow \mathbf{T}) \wedge (\mathbf{F} \rightarrow \mathbf{F})$
T	**F**	**T**	$(\mathbf{T} \rightarrow \mathbf{T}) \wedge (\mathbf{F} \rightarrow \mathbf{T})$
T	**T**	**F**	$(\mathbf{F} \rightarrow \mathbf{T}) \wedge \mathbf{F}$
T	**T**	**T**	$(\mathbf{F} \rightarrow \mathbf{T}) \wedge (\mathbf{T} \rightarrow \mathbf{T})$

Potassco

Some truth tabling, back to SAT

a	b	c	$(\neg b \rightarrow a) \wedge (b \rightarrow c)$
F	F	F	$\mathbf{F} \wedge (\mathbf{F} \rightarrow \mathbf{F})$
F	F	T	$\mathbf{F} \wedge (\mathbf{F} \rightarrow \mathbf{T})$
F	T	F	$(\mathbf{F} \rightarrow \mathbf{F}) \wedge \mathbf{F}$
F	T	T	$(\mathbf{F} \rightarrow \mathbf{F}) \wedge (\mathbf{T} \rightarrow \mathbf{T})$
T	F	F	$(\mathbf{T} \rightarrow \mathbf{T}) \wedge (\mathbf{F} \rightarrow \mathbf{F})$
T	F	T	$(\mathbf{T} \rightarrow \mathbf{T}) \wedge (\mathbf{F} \rightarrow \mathbf{T})$
T	T	F	$(\mathbf{F} \rightarrow \mathbf{T}) \wedge \mathbf{F}$
T	T	T	$(\mathbf{F} \rightarrow \mathbf{T}) \wedge (\mathbf{T} \rightarrow \mathbf{T})$

Potassco

Some truth tabling, back to SAT

a	b	c	$(\neg b \rightarrow a) \wedge (b \rightarrow c)$
F	F	F	F $\wedge$ T
F	F	T	F $\wedge$ T
F	T	F	T $\wedge$ F
F	T	T	T $\wedge$ T
T	F	F	T $\wedge$ T
T	F	T	T $\wedge$ T
T	T	F	T $\wedge$ F
T	T	T	T $\wedge$ T

Potassco

Some truth tabling, back to SAT

a	b	c	$(\neg b \rightarrow a) \wedge (b \rightarrow c)$
F	F	F	F
F	F	T	F
F	T	F	F
F	T	T	T
T	F	F	T
T	F	T	T
T	T	F	F
T	T	T	T

Potassco

Some truth tabling, back to SAT

a	b	c	$(\neg b \rightarrow a) \wedge (b \rightarrow c)$
F	F	F	F
F	F	T	F
F	T	F	F
F	T	T	T
T	F	F	T
T	F	T	T
T	T	F	F
T	T	T	T

- We get four models: $\{b, c\}$, $\{a\}$, $\{a, c\}$, and $\{a, b, c\}$

Potassco

Some truth tabling, and now ASP

a	b	c	$(\neg b \rightarrow a) \wedge (b \rightarrow c)$
F	F	F	
F	F	T	
F	T	F	
F	T	T	
T	F	F	
T	F	T	
T	T	F	
T	T	T	

Potassco

Some truth tabling, and now ASP

a	b	c	$(\neg b \rightarrow a) \wedge (b \rightarrow c)$
F	**F**	**F**	$(\neg \mathbf{F} \rightarrow a) \wedge (b \rightarrow c)$
F	**F**	**T**	$(\neg \mathbf{F} \rightarrow a) \wedge (b \rightarrow c)$
F	**T**	**F**	$(\neg \mathbf{T} \rightarrow a) \wedge (b \rightarrow c)$
F	**T**	**T**	$(\neg \mathbf{T} \rightarrow a) \wedge (b \rightarrow c)$
T	**F**	**F**	$(\neg \mathbf{F} \rightarrow a) \wedge (b \rightarrow c)$
T	**F**	**T**	$(\neg \mathbf{F} \rightarrow a) \wedge (b \rightarrow c)$
T	**T**	**F**	$(\neg \mathbf{T} \rightarrow a) \wedge (b \rightarrow c)$
T	**T**	**T**	$(\neg \mathbf{T} \rightarrow a) \wedge (b \rightarrow c)$

Potassco

Some truth tabling, and now ASP

a	b	c	$(\neg b \rightarrow a) \wedge (b \rightarrow c)$
F	**F**	**F**	$(\mathbf{T} \rightarrow a) \wedge (b \rightarrow c)$
F	**F**	**T**	$(\mathbf{T} \rightarrow a) \wedge (b \rightarrow c)$
F	**T**	**F**	$(\mathbf{F} \rightarrow a) \wedge (b \rightarrow c)$
F	**T**	**T**	$(\mathbf{F} \rightarrow a) \wedge (b \rightarrow c)$
T	**F**	**F**	$(\mathbf{T} \rightarrow a) \wedge (b \rightarrow c)$
T	**F**	**T**	$(\mathbf{T} \rightarrow a) \wedge (b \rightarrow c)$
T	**T**	**F**	$(\mathbf{F} \rightarrow a) \wedge (b \rightarrow c)$
T	**T**	**T**	$(\mathbf{F} \rightarrow a) \wedge (b \rightarrow c)$

Potassco

Some truth tabling, and now ASP

a	b	c	$(\neg b \rightarrow a) \wedge (b \rightarrow c)$
F	**F**	**F**	$a \wedge (b \rightarrow c)$
F	**F**	**T**	$a \wedge (b \rightarrow c)$
F	**T**	**F**	$(\mathbf{F} \rightarrow a) \wedge (b \rightarrow c)$
F	**T**	**T**	$(\mathbf{F} \rightarrow a) \wedge (b \rightarrow c)$
T	**F**	**F**	$a \wedge (b \rightarrow c)$
T	**F**	**T**	$a \wedge (b \rightarrow c)$
T	**T**	**F**	$(\mathbf{F} \rightarrow a) \wedge (b \rightarrow c)$
T	**T**	**T**	$(\mathbf{F} \rightarrow a) \wedge (b \rightarrow c)$

Potassco

Some truth tabling, and now ASP

a	b	c	$(\neg b \rightarrow a) \wedge (b \rightarrow c)$
F	**F**	**F**	$a \wedge (b \rightarrow c)$
F	**F**	**T**	$a \wedge (b \rightarrow c)$
F	**T**	**F**	$(\mathbf{F} \rightarrow a) \wedge (b \rightarrow c)$
F	**T**	**T**	$(\mathbf{F} \rightarrow a) \wedge (b \rightarrow c)$
T	**F**	**F**	$a \wedge (b \rightarrow c)$
T	**F**	**T**	$a \wedge (b \rightarrow c)$
T	**T**	**F**	$(\mathbf{F} \rightarrow a) \wedge (b \rightarrow c)$
T	**T**	**T**	$(\mathbf{F} \rightarrow a) \wedge (b \rightarrow c)$

Potassco

Some truth tabling, and now ASP

a	b	c	$(\neg b \rightarrow a) \wedge (b \rightarrow c)$
F	**F**	**F**	$a \wedge (b \rightarrow c)$
F	**F**	**T**	$a \wedge (b \rightarrow c)$
F	**T**	**F**	$\mathbf{T} \wedge (b \rightarrow c)$
F	**T**	**T**	$\mathbf{T} \wedge (b \rightarrow c)$
T	**F**	**F**	$a \wedge (b \rightarrow c)$
T	**F**	**T**	$a \wedge (b \rightarrow c)$
T	**T**	**F**	$\mathbf{T} \wedge (b \rightarrow c)$
T	**T**	**T**	$\mathbf{T} \wedge (b \rightarrow c)$

Potassco

Some truth tabling, and now ASP

a	b	c	$(\neg b \rightarrow a) \wedge (b \rightarrow c)$
F	**F**	**F**	$a \wedge (b \rightarrow c)$
F	**F**	**T**	$a \wedge (b \rightarrow c)$
F	**T**	**F**	$(b \rightarrow c)$
F	**T**	**T**	$(b \rightarrow c)$
T	**F**	**F**	$a \wedge (b \rightarrow c)$
T	**F**	**T**	$a \wedge (b \rightarrow c)$
T	**T**	**F**	$(b \rightarrow c)$
T	**T**	**T**	$(b \rightarrow c)$

Reduct

Potassco

Some truth tabling, and now ASP

a	b	c	$(\neg b \rightarrow a) \wedge (b \rightarrow c)$
F	F	F	$a \wedge (b \rightarrow c)$
F	F	T	$a \wedge (b \rightarrow c)$
F	T	F	$(b \rightarrow c)$
F	T	T	$(b \rightarrow c)$
T	F	F	$a \wedge (b \rightarrow c)$
T	F	T	$a \wedge (b \rightarrow c)$
T	T	F	$(b \rightarrow c)$
T	T	T	$(b \rightarrow c)$

Reduct

Potassco

Some truth tabling, and now ASP

a	b	c	$(\neg b \rightarrow a) \wedge (b \rightarrow c)$	
F	F	F	$a \wedge (b \rightarrow c)$	
F	F	T	$a \wedge (b \rightarrow c)$	
F	T	F	$(b \rightarrow c)$	
F	T	T	$(b \rightarrow c)$	$\models$
T	F	F	$a \wedge (b \rightarrow c)$	$\models a$
T	F	T	$a \wedge (b \rightarrow c)$	$\models a$
T	T	F	$(b \rightarrow c)$	
T	T	T	$(b \rightarrow c)$	$\models$

Reduct

Some truth tabling, and now ASP

a	b	c	$(\neg b \rightarrow a) \wedge (b \rightarrow c)$	
F	F	F	$a \wedge (b \rightarrow c)$	
F	F	T	$a \wedge (b \rightarrow c)$	
F	T	F	$(b \rightarrow c)$	
F	T	T	$(b \rightarrow c)$	$\models$
T	F	F	$a \wedge (b \rightarrow c)$	$\models a$
T	F	T	$a \wedge (b \rightarrow c)$	$\models a$
T	T	F	$(b \rightarrow c)$	
T	T	T	$(b \rightarrow c)$	$\models$

Reduct

Some truth tabling, and now ASP

a	b	c	$(\neg b \rightarrow a) \wedge (b \rightarrow c)$	
F	F	F	$a \wedge (b \rightarrow c)$	$\models a$
F	F	T	$a \wedge (b \rightarrow c)$	$\models a$
F	T	F	$(b \rightarrow c)$	$\models$
F	T	T	$(b \rightarrow c)$	$\models$
T	F	F	$a \wedge (b \rightarrow c)$	$\models a$
T	F	T	$a \wedge (b \rightarrow c)$	$\models a$
T	T	F	$(b \rightarrow c)$	$\models$
T	T	T	$(b \rightarrow c)$	$\models$

Reduct

Potassco

Some truth tabling, and now ASP

a	b	c	$(\neg b \rightarrow a) \wedge (b \rightarrow c)$		
F	F	F	$a \wedge (b \rightarrow c)$		
F	F	T	$a \wedge (b \rightarrow c)$		
F	T	F	$(b \rightarrow c)$		
F	T	T	$(b \rightarrow c)$		
T	F	F	$a \wedge (b \rightarrow c)$	$\models a$	Stable model
T	F	T	$a \wedge (b \rightarrow c)$		
T	T	F	$(b \rightarrow c)$		
T	T	T	$(b \rightarrow c)$		

Reduct

- We get one stable model: $\{a\}$

Some truth tabling, and now ASP

a	b	c	$(\neg b \rightarrow a) \wedge (b \rightarrow c)$	
F	F	F	$a \wedge (b \rightarrow c)$	
F	F	T	$a \wedge (b \rightarrow c)$	
F	T	F	$(b \rightarrow c)$	
F	T	T	$(b \rightarrow c)$	
T	F	F	$a \wedge (b \rightarrow c)$	$\models a$ Stable model
T	F	T	$a \wedge (b \rightarrow c)$	
T	T	F	$(b \rightarrow c)$	
T	T	T	$(b \rightarrow c)$	
			Reduct	

- We get one stable model: $\{a\}$
- Stable models = Smallest models of (respective) reducts

Potassco

Outline

Potassco

From ASP to SAT

a	b	c	$(\neg b \rightarrow a) \wedge (b \rightarrow c)$
F	F	F	
F	F	T	
F	T	F	
F	T	T	
T	F	F	
T	F	T	
T	T	F	
T	T	T	

From ASP to SAT

a	b	c	$(\neg b \rightarrow a) \wedge (b \rightarrow c)$
F	F	F	
F	F	T	
F	T	F	
F	T	T	
T	F	F	
T	F	T	
T	T	F	
T	T	T	

- Consider the schema: $\neg\neg A \rightarrow A$

Potassco

From ASP to SAT

a	b	c	$(\neg b \rightarrow a) \wedge (b \rightarrow c) \wedge (\neg\neg a \rightarrow a) \wedge (\neg\neg b \rightarrow b) \wedge (\neg\neg c \rightarrow c)$
F	**F**	**F**	
F	**F**	**T**	
F	**T**	**F**	
F	**T**	**T**	
T	**F**	**F**	
T	**F**	**T**	
T	**T**	**F**	
T	**T**	**T**	

- Consider the schema: $\neg\neg A \rightarrow A$

Potassco

From ASP to SAT

a	b	c	$(\neg b \rightarrow a) \wedge (b \rightarrow c) \wedge (\neg\neg a \rightarrow a) \wedge (\neg\neg b \rightarrow b) \wedge (\neg\neg c \rightarrow c)$
F	**F**	**F**	$(\neg\mathbf{F} \rightarrow a) \wedge (b \rightarrow c) \wedge (\neg\neg\mathbf{F} \rightarrow a) \wedge (\neg\neg\mathbf{F} \rightarrow b) \wedge (\neg\neg\mathbf{F} \rightarrow c)$
F	**F**	**T**	$(\neg\mathbf{F} \rightarrow a) \wedge (b \rightarrow c) \wedge (\neg\neg\mathbf{F} \rightarrow a) \wedge (\neg\neg\mathbf{F} \rightarrow b) \wedge (\neg\neg\mathbf{T} \rightarrow c)$
F	**T**	**F**	$(\neg\mathbf{T} \rightarrow a) \wedge (b \rightarrow c) \wedge (\neg\neg\mathbf{F} \rightarrow a) \wedge (\neg\neg\mathbf{T} \rightarrow b) \wedge (\neg\neg\mathbf{F} \rightarrow c)$
F	**T**	**T**	$(\neg\mathbf{T} \rightarrow a) \wedge (b \rightarrow c) \wedge (\neg\neg\mathbf{F} \rightarrow a) \wedge (\neg\neg\mathbf{T} \rightarrow b) \wedge (\neg\neg\mathbf{T} \rightarrow c)$
T	**F**	**F**	$(\neg\mathbf{F} \rightarrow a) \wedge (b \rightarrow c) \wedge (\neg\neg\mathbf{T} \rightarrow a) \wedge (\neg\neg\mathbf{F} \rightarrow b) \wedge (\neg\neg\mathbf{F} \rightarrow c)$
T	**F**	**T**	$(\neg\mathbf{F} \rightarrow a) \wedge (b \rightarrow c) \wedge (\neg\neg\mathbf{T} \rightarrow a) \wedge (\neg\neg\mathbf{F} \rightarrow b) \wedge (\neg\neg\mathbf{T} \rightarrow c)$
T	**T**	**F**	$(\neg\mathbf{T} \rightarrow a) \wedge (b \rightarrow c) \wedge (\neg\neg\mathbf{T} \rightarrow a) \wedge (\neg\neg\mathbf{T} \rightarrow b) \wedge (\neg\neg\mathbf{F} \rightarrow c)$
T	**T**	**T**	$(\neg\mathbf{T} \rightarrow a) \wedge (b \rightarrow c) \wedge (\neg\neg\mathbf{T} \rightarrow a) \wedge (\neg\neg\mathbf{T} \rightarrow b) \wedge (\neg\neg\mathbf{T} \rightarrow c)$

Potassco

From ASP to SAT

a	b	c	$(\neg b \rightarrow a) \wedge (b \rightarrow c) \wedge (\neg\neg a \rightarrow a) \wedge (\neg\neg b \rightarrow b) \wedge (\neg\neg c \rightarrow c)$
F	**F**	**F**	$(\mathbf{T} \rightarrow a) \wedge (b \rightarrow c) \wedge (\mathbf{F} \rightarrow a) \wedge (\mathbf{F} \rightarrow b) \wedge (\mathbf{F} \rightarrow c)$
F	**F**	**T**	$(\mathbf{T} \rightarrow a) \wedge (b \rightarrow c) \wedge (\mathbf{F} \rightarrow a) \wedge (\mathbf{F} \rightarrow b) \wedge (\mathbf{T} \rightarrow c)$
F	**T**	**F**	$(\mathbf{F} \rightarrow a) \wedge (b \rightarrow c) \wedge (\mathbf{F} \rightarrow a) \wedge (\mathbf{T} \rightarrow b) \wedge (\mathbf{F} \rightarrow c)$
F	**T**	**T**	$(\mathbf{F} \rightarrow a) \wedge (b \rightarrow c) \wedge (\mathbf{F} \rightarrow a) \wedge (\mathbf{T} \rightarrow b) \wedge (\mathbf{T} \rightarrow c)$
T	**F**	**F**	$(\mathbf{T} \rightarrow a) \wedge (b \rightarrow c) \wedge (\mathbf{T} \rightarrow a) \wedge (\mathbf{F} \rightarrow b) \wedge (\mathbf{F} \rightarrow c)$
T	**F**	**T**	$(\mathbf{T} \rightarrow a) \wedge (b \rightarrow c) \wedge (\mathbf{T} \rightarrow a) \wedge (\mathbf{F} \rightarrow b) \wedge (\mathbf{T} \rightarrow c)$
T	**T**	**F**	$(\mathbf{F} \rightarrow a) \wedge (b \rightarrow c) \wedge (\mathbf{T} \rightarrow a) \wedge (\mathbf{T} \rightarrow b) \wedge (\mathbf{F} \rightarrow c)$
T	**T**	**T**	$(\mathbf{F} \rightarrow a) \wedge (b \rightarrow c) \wedge (\mathbf{T} \rightarrow a) \wedge (\mathbf{T} \rightarrow b) \wedge (\mathbf{T} \rightarrow c)$

Potassco

From ASP to SAT

a	b	c	$(\neg b \rightarrow a) \wedge (b \rightarrow c) \wedge (\neg\neg a \rightarrow a) \wedge (\neg\neg b \rightarrow b) \wedge (\neg\neg c \rightarrow c)$
F	**F**	**F**	$a \wedge (b \rightarrow c) \wedge (\mathbf{F} \rightarrow a) \wedge (\mathbf{F} \rightarrow b) \wedge (\mathbf{F} \rightarrow c)$
F	**F**	**T**	$a \wedge (b \rightarrow c) \wedge (\mathbf{F} \rightarrow a) \wedge (\mathbf{F} \rightarrow b) \wedge c$
F	**T**	**F**	$(\mathbf{F} \rightarrow a) \wedge (b \rightarrow c) \wedge (\mathbf{F} \rightarrow a) \wedge b \wedge (\mathbf{F} \rightarrow c)$
F	**T**	**T**	$(\mathbf{F} \rightarrow a) \wedge (b \rightarrow c) \wedge (\mathbf{F} \rightarrow a) \wedge b \wedge c$
T	**F**	**F**	$a \wedge (b \rightarrow c) \wedge a \wedge (\mathbf{F} \rightarrow b) \wedge (\mathbf{F} \rightarrow c)$
T	**F**	**T**	$a \wedge (b \rightarrow c) \wedge a \wedge (\mathbf{F} \rightarrow b) \wedge c$
T	**T**	**F**	$(\mathbf{F} \rightarrow a) \wedge (b \rightarrow c) \wedge a \wedge b \wedge (\mathbf{F} \rightarrow c)$
T	**T**	**T**	$(\mathbf{F} \rightarrow a) \wedge (b \rightarrow c) \wedge a \wedge b \wedge c$

Potassco

From ASP to SAT

a	b	c	$(\neg b \rightarrow a) \wedge (b \rightarrow c) \wedge (\neg\neg a \rightarrow a) \wedge (\neg\neg b \rightarrow b) \wedge (\neg\neg c \rightarrow c)$
F	**F**	**F**	$a \wedge (b \rightarrow c) \wedge$ **T** $\wedge$ **T** $\wedge$ **T**
F	**F**	**T**	$a \wedge (b \rightarrow c) \wedge$ **T** $\wedge$ **T** $\wedge$ c
F	**T**	**F**	**T** $\wedge (b \rightarrow c) \wedge$ **T** $\wedge$ $b \wedge$ **T**
F	**T**	**T**	**T** $\wedge (b \rightarrow c) \wedge$ **T** $\wedge$ $b \wedge$ c
T	**F**	**F**	$a \wedge (b \rightarrow c) \wedge$ $a \wedge$ **T** $\wedge$ **T**
T	**F**	**T**	$a \wedge (b \rightarrow c) \wedge$ $a \wedge$ **T** $\wedge$ c
T	**T**	**F**	**T** $\wedge (b \rightarrow c) \wedge$ $a \wedge$ $b \wedge$ **T**
T	**T**	**T**	**T** $\wedge (b \rightarrow c) \wedge$ $a \wedge$ $b \wedge$ c

Potassco

From ASP to SAT

a	b	c	$(\neg b \rightarrow a) \wedge (b \rightarrow c) \wedge (\neg\neg a \rightarrow a) \wedge (\neg\neg b \rightarrow b) \wedge (\neg\neg c \rightarrow c)$
F	**F**	**F**	$a \wedge (b \rightarrow c)$
F	**F**	**T**	$a \wedge (b \rightarrow c) \wedge c$
F	**T**	**F**	$(b \rightarrow c) \wedge b$
F	**T**	**T**	$(b \rightarrow c) \wedge b \wedge c$
T	**F**	**F**	$a \wedge (b \rightarrow c) \wedge a$
T	**F**	**T**	$a \wedge (b \rightarrow c) \wedge a \wedge c$
T	**T**	**F**	$(b \rightarrow c) \wedge a \wedge b$
T	**T**	**T**	$(b \rightarrow c) \wedge a \wedge b \wedge c$

Potassco

From ASP to SAT

a	b	c	$(\neg b \rightarrow a) \wedge (b \rightarrow c) \wedge \ulcorner (a \rightarrow a) \urcorner \wedge \ulcorner (b \rightarrow b) \urcorner \wedge \ulcorner (c \rightarrow c) \urcorner$
F	**F**	**F**	$a \wedge (b \rightarrow c)$
F	**F**	**T**	$a \wedge (b \rightarrow c) \wedge c$
F	**T**	**F**	$(b \rightarrow c) \wedge b$
F	**T**	**T**	$(b \rightarrow c) \wedge b \wedge c$
T	**F**	**F**	$a \wedge (b \rightarrow c) \wedge a$
T	**F**	**T**	$a \wedge (b \rightarrow c) \wedge a \wedge c$
T	**T**	**F**	$(b \rightarrow c) \wedge a \wedge b$
T	**T**	**T**	$(b \rightarrow c) \wedge a \wedge b \wedge c$

Potassco

From ASP to SAT

a	b	c	$(\neg b \rightarrow a) \wedge (b \rightarrow c) \wedge \ulcorner (\neg a \rightarrow a) \urcorner \wedge \ulcorner (\neg b \rightarrow b) \urcorner \wedge \ulcorner (\neg c \rightarrow c) \urcorner$	
F	**F**	**F**	$a \wedge (b \rightarrow c)$	$\models a$
F	**F**	**T**	$a \wedge (b \rightarrow c) \wedge c$	$\models a, c$
F	**T**	**F**	$(b \rightarrow c) \wedge b$	$\models b, c$
F	**T**	**T**	$(b \rightarrow c) \wedge b \wedge c$	$\models b, c$
T	**F**	**F**	$a \wedge (b \rightarrow c) \wedge a$	$\models a$
T	**F**	**T**	$a \wedge (b \rightarrow c) \wedge a \wedge c$	$\models a, c$
T	**T**	**F**	$(b \rightarrow c) \wedge a \wedge b$	$\models a, b, c$
T	**T**	**T**	$(b \rightarrow c) \wedge a \wedge b \wedge c$	$\models a, b, c$

Potassco

From ASP to SAT

a	b	c	$(\neg b \rightarrow a) \wedge (b \rightarrow c) \wedge \neg(a \rightarrow a) \wedge \neg(b \rightarrow b) \wedge \neg(c \rightarrow c)$	
F	F	F	$a \wedge (b \rightarrow c)$	$\models a$
F	F	T	$a \wedge (b \rightarrow c) \wedge c$	$\models a, c$
F	T	F	$(b \rightarrow c) \wedge b$	$\models b, c$
F	T	T	$(b \rightarrow c) \wedge b \wedge c$	$\models b, c$
T	F	F	$a \wedge (b \rightarrow c) \wedge a$	$\models a$
T	F	T	$a \wedge (b \rightarrow c) \wedge a \wedge c$	$\models a, c$
T	T	F	$(b \rightarrow c) \wedge a \wedge b$	$\models a, b, c$
T	T	T	$(b \rightarrow c) \wedge a \wedge b \wedge c$	$\models a, b, c$

Potassco

From ASP to SAT

a	b	c	$(\neg b \rightarrow a) \wedge (b \rightarrow c) \wedge \sqcap(a \rightarrow a\sqcap) \wedge \sqcap(b \rightarrow b\sqcap) \wedge \sqcap(c \rightarrow c\sqcap)$	
F	**F**	**F**	$a \wedge (b \rightarrow c)$	$\not\models a$
F	**F**	**T**	$a \wedge (b \rightarrow c) \wedge c$	$\not\models a, c$
F	**T**	**F**	$(b \rightarrow c) \wedge b$	$\not\models b, c$
F	**T**	**T**	$(b \rightarrow c) \wedge b \wedge c$	$\models b, c$
T	**F**	**F**	$a \wedge (b \rightarrow c) \wedge a$	$\not\models a$
T	**F**	**T**	$a \wedge (b \rightarrow c) \wedge a \wedge c$	$\not\models a, c$
T	**T**	**F**	$(b \rightarrow c) \wedge a \wedge b$	$\not\models a, b, c$
T	**T**	**T**	$(b \rightarrow c) \wedge a \wedge b \wedge c$	$\models a, b, c$

- We get four stable models: $\{b, c\}$, $\{a\}$, $\{a, c\}$, $\{a, b, c\}$

Potassco

From ASP to SAT

a	b	c	$(\neg b \rightarrow a) \wedge (b \rightarrow c) \wedge (\neg\neg a \rightarrow a) \wedge (\neg\neg b \rightarrow b) \wedge (\neg\neg c \rightarrow c)$
F	F	F	$a \wedge (b \rightarrow c)$
F	F	T	$a \wedge (b \rightarrow c) \wedge$ c
F	T	F	$(b \rightarrow c) \wedge$ b
F	T	T	$(b \rightarrow c) \wedge$ $b \wedge c$
T	F	F	$a \wedge (b \rightarrow c) \wedge$ a
T	F	T	$a \wedge (b \rightarrow c) \wedge$ $a \wedge c$
T	T	F	$(b \rightarrow c) \wedge$ $a \wedge b$
T	T	T	$(b \rightarrow c) \wedge$ $a \wedge b \wedge c$

Potassco

From ASP to SAT

a	b	c	$(\neg b \rightarrow a) \wedge (b \rightarrow c) \wedge (\neg\neg a \rightarrow a) \wedge (\neg\neg b \rightarrow b) \wedge (\neg\neg c \rightarrow c)$
F	**F**	**F**	$a \wedge (b \rightarrow c)$
F	**F**	**T**	$a \wedge (b \rightarrow c) \wedge c$
F	**T**	**F**	$(b \rightarrow c) \wedge b$
F	**T**	**T**	$(b \rightarrow c) \wedge b \wedge c$
T	**F**	**F**	$a \wedge (b \rightarrow c) \wedge a$
T	**F**	**T**	$a \wedge (b \rightarrow c) \wedge a \wedge c$
T	**T**	**F**	$(b \rightarrow c) \wedge a \wedge b$
T	**T**	**T**	$(b \rightarrow c) \wedge a \wedge b \wedge c$

- Note $\neg\neg A \rightarrow A$ is equivalent to $A \vee \neg A$

Potassco

From ASP to SAT

a	b	c	$(\neg b \rightarrow a) \wedge (b \rightarrow c) \wedge (\neg\neg a \rightarrow a) \wedge (\neg\neg b \rightarrow b) \wedge (\neg\neg c \rightarrow c)$
F	F	F	$a \wedge (b \rightarrow c)$
F	F	T	$a \wedge (b \rightarrow c) \wedge$ c
F	T	F	$(b \rightarrow c) \wedge$ b
F	T	T	$(b \rightarrow c) \wedge$ $b \wedge c$
T	F	F	$a \wedge (b \rightarrow c) \wedge$ a
T	F	T	$a \wedge (b \rightarrow c) \wedge$ $a \wedge c$
T	T	F	$(b \rightarrow c) \wedge$ $a \wedge b$
T	T	T	$(b \rightarrow c) \wedge$ $a \wedge b \wedge c$

- Note $\neg\neg A \rightarrow A$ is equivalent to $A \vee \neg A$
- SAT = ASP + Tertium non datur

Potassco

From ASP to SAT

a	b	c	$(\neg b \rightarrow a) \wedge (b \rightarrow c) \wedge (\neg\neg a \rightarrow a) \wedge (\neg\neg b \rightarrow b) \wedge (\neg\neg c \rightarrow c)$
F	F	F	$a \wedge (b \rightarrow c)$
F	F	T	$a \wedge (b \rightarrow c) \wedge$ c
F	T	F	$(b \rightarrow c) \wedge$ b
F	T	T	$(b \rightarrow c) \wedge$ $b \wedge c$
T	F	F	$a \wedge (b \rightarrow c) \wedge$ a
T	F	T	$a \wedge (b \rightarrow c) \wedge$ $a \wedge c$
T	T	F	$(b \rightarrow c) \wedge$ $a \wedge b$
T	T	T	$(b \rightarrow c) \wedge$ $a \wedge b \wedge c$

- Note $\neg\neg A \rightarrow A$ is equivalent to $A \vee \neg A$
- SAT = ASP + Tertium non datur (also called choice)

Potassco

Outline

Potassco

From SAT to ASP

$$\Pi = \{a \leftarrow \neg b \quad b \leftarrow \neg a \quad x \leftarrow a, \neg c \quad x \leftarrow y \quad y \leftarrow x, b\}$$

$$CF(\Pi) = \{a \leftrightarrow \neg b \quad b \leftrightarrow \neg a \quad x \leftrightarrow (a \wedge \neg c) \vee y \quad y \leftrightarrow x \wedge b\}$$
$$\cup \{c \leftrightarrow \bot\}$$

$$LF(\Pi) = \{(x \vee y) \rightarrow a \wedge \neg c\}$$

Classical models of $CF(\Pi)$

$\{b\}$, $\{b, c\}$, $\{b, x, y\}$, $\{b, c, x, y\}$, $\{a, c\}$, $\{a, b, c\}$, $\{a, x\}$, $\{a, c, x\}$, $\{a, x, y\}$, $\{a, c, x, y\}$, $\{a, b, x, y\}$, $\{a, b, c, x, y\}$

Potassco

From SAT to ASP

$$\Pi = \{a \leftarrow \neg b \quad b \leftarrow \neg a \quad x \leftarrow a, \neg c \quad x \leftarrow y \quad y \leftarrow x, b\}$$

$$CF(\Pi) = \{a \leftrightarrow \neg b \quad b \leftrightarrow \neg a \quad x \leftrightarrow (a \wedge \neg c) \vee y \quad y \leftrightarrow x \wedge b\}$$
$$\cup \{c \leftrightarrow \bot\}$$

$$LF(\Pi) = \{(x \vee y) \rightarrow a \wedge \neg c\}$$

Classical models of Π (only true atoms shown)

$\{b\}$, $\{b, c\}$, $\{b, x, y\}$, $\{b, c, x, y\}$, $\{a, c\}$, $\{a, b, c\}$, $\{a, x\}$, $\{a, c, x\}$, $\{a, x, y\}$, $\{a, c, x, y\}$, $\{a, b, x, y\}$, $\{a, b, c, x, y\}$

Potassco

From SAT to ASP

$$\Pi = \{a \leftarrow \neg b \quad b \leftarrow \neg a \quad x \leftarrow a, \neg c \quad x \leftarrow y \quad y \leftarrow x, b\}$$

$$CF(\Pi) = \{a \leftrightarrow \neg b \quad b \leftrightarrow \neg a \quad x \leftrightarrow (a \wedge \neg c) \vee y \quad y \leftrightarrow x \wedge b\} \cup \{c \leftrightarrow \bot\}$$

$$LF(\Pi) = \{(x \vee y) \rightarrow a \wedge \neg c\}$$

Classical models of Π

$\{b\}$, $\{b, c\}$, $\{b, x, y\}$, $\{b, c, x, y\}$, $\{a, c\}$, $\{a, b, c\}$, $\{a, x\}$, $\{a, c, x\}$, $\{a, x, y\}$, $\{a, c, x, y\}$, $\{a, b, x, y\}$, $\{a, b, c, x, y\}$

- Unsupported atoms
- Unfounded atoms

Potassco

From SAT to ASP

$$\begin{aligned}
\Pi &= \{a \leftarrow \neg b \quad b \leftarrow \neg a \quad x \leftarrow a, \neg c \quad x \leftarrow y \quad y \leftarrow x, b\} \\
CF(\Pi) &= \{a \leftrightarrow \neg b \quad b \leftrightarrow \neg a \quad x \leftrightarrow (a \wedge \neg c) \vee y \quad y \leftrightarrow x \wedge b\} \\
&\cup \{c \leftrightarrow \bot\} \\
LF(\Pi) &= \{(x \vee y) \rightarrow a \wedge \neg c\}
\end{aligned}$$

Classical models of Π

$\{b\}$, $\{b, c\}$, $\{b, x, y\}$, $\{b, c, x, y\}$, $\{a, c\}$, $\{a, b, c\}$, $\{a, x\}$, $\{a, c, x\}$, $\{a, x, y\}$, $\{a, c, x, y\}$, $\{a, b, x, y\}$, $\{a, b, c, x, y\}$

- Unsupported atoms
- Unfounded atoms

Potassco

From SAT to ASP

$$\begin{aligned}
\Pi &= \{a \leftarrow \neg b \quad b \leftarrow \neg a \quad x \leftarrow a, \neg c \quad x \leftarrow y \quad y \leftarrow x, b\} \\
CF(\Pi) &= \{a \leftrightarrow \neg b \quad b \leftrightarrow \neg a \quad x \leftrightarrow (a \wedge \neg c) \vee y \quad y \leftrightarrow x \wedge b\} \\
&\cup \{c \leftrightarrow \bot\} \\
LF(\Pi) &= \{(x \vee y) \rightarrow a \wedge \neg c\}
\end{aligned}$$

Classical models of $CF(\Pi)$

$\{b\}$, $\{b, c\}$, $\{b, x, y\}$, $\{b, c, x, y\}$, $\{a, c\}$, $\{a, b, c\}$, $\{a, x\}$, $\{a, c, x\}$, $\{a, x, y\}$, $\{a, c, x, y\}$, $\{a, b, x, y\}$, $\{a, b, c, x, y\}$

- ~~Unsupported atoms~~ eliminated by completion formulas $CF(\Pi)$
- Unfounded atoms

Potassco

From SAT to ASP

$$
\begin{aligned}
\Pi &= \{a \leftarrow \neg b \quad b \leftarrow \neg a \quad x \leftarrow a, \neg c \quad x \leftarrow y \quad y \leftarrow x, b\} \\
CF(\Pi) &= \{a \leftrightarrow \neg b \quad b \leftrightarrow \neg a \quad x \leftrightarrow (a \wedge \neg c) \vee y \quad y \leftrightarrow x \wedge b\} \\
&\cup \{c \leftrightarrow \bot\} \\
LF(\Pi) &= \{(x \vee y) \rightarrow a \wedge \neg c\}
\end{aligned}
$$

Classical models of $CF(\Pi)$

$\{b\}$, $\{b, c\}$, $\{b, x, y\}$, $\{b, c, x, y\}$, $\{a, c\}$, $\{a, b, c\}$, $\{a, x\}$, $\{a, c, x\}$, $\{a, x, y\}$, $\{a, c, x, y\}$, $\{a, b, x, y\}$, $\{a, b, c, x, y\}$

- ~~Unsupported atoms~~ eliminated by completion formulas $CF(\Pi)$
- Unfounded atoms

Potassco

From SAT to ASP

$$
\begin{aligned}
\Pi &= \{a \leftarrow \neg b \quad b \leftarrow \neg a \quad x \leftarrow a, \neg c \quad x \leftarrow y \quad y \leftarrow x, b\} \\
CF(\Pi) &= \{a \leftrightarrow \neg b \quad b \leftrightarrow \neg a \quad x \leftrightarrow (a \wedge \neg c) \vee y \quad y \leftrightarrow x \wedge b\} \\
&\cup \{c \leftrightarrow \bot\} \\
LF(\Pi) &= \{(x \vee y) \rightarrow a \wedge \neg c\}
\end{aligned}
$$

Classical models of $CF(\Pi) \cup LF(\Pi)$

$\{b\}$, $\{b, c\}$, $\{b, x, y\}$, $\{b, c, x, y\}$, $\{a, c\}$, $\{a, b, c\}$, $\{a, x\}$, $\{a, c, x\}$, $\{a, x, y\}$, $\{a, c, x, y\}$, $\{a, b, x, y\}$, $\{a, b, c, x, y\}$

- ~~Unsupported atoms~~ eliminated by completion formulas $CF(\Pi)$
- ~~Unfounded atoms~~ eliminated by loop formulas $LF(\Pi)$

Potassco

From SAT to ASP

$$
\begin{aligned}
\Pi &= \{a \leftarrow \neg b \quad b \leftarrow \neg a \quad x \leftarrow a, \neg c \quad x \leftarrow y \quad y \leftarrow x, b\} \\
CF(\Pi) &= \{a \leftrightarrow \neg b \quad b \leftrightarrow \neg a \quad x \leftrightarrow (a \wedge \neg c) \vee y \quad y \leftrightarrow x \wedge b\} \\
&\cup \{c \leftrightarrow \bot\} \\
LF(\Pi) &= \{(x \vee y) \rightarrow a \wedge \neg c\}
\end{aligned}
$$

Theorem (Lin and Zhao)

X is a stable model of Π iff $X \models CF(\Pi) \cup LF(\Pi)$

Potassco

From SAT to ASP

$$
\begin{aligned}
\Pi &= \{a \leftarrow \neg b \quad b \leftarrow \neg a \quad x \leftarrow a, \neg c \quad x \leftarrow y \quad y \leftarrow x, b\} \\
CF(\Pi) &= \{a \leftrightarrow \neg b \quad b \leftrightarrow \neg a \quad x \leftrightarrow (a \wedge \neg c) \vee y \quad y \leftrightarrow x \wedge b\} \\
&\cup \{c \leftrightarrow \bot\} \\
LF(\Pi) &= \{(x \vee y) \rightarrow a \wedge \neg c\}
\end{aligned}
$$

Theorem (Lin and Zhao)

X is a stable model of Π iff $X \models CF(\Pi) \cup LF(\Pi)$

- Size of $CF(\Pi)$ is linear in the size of Π
- Size of $LF(\Pi)$ may be exponential in the size of Π

Potassco

ASP and SAT

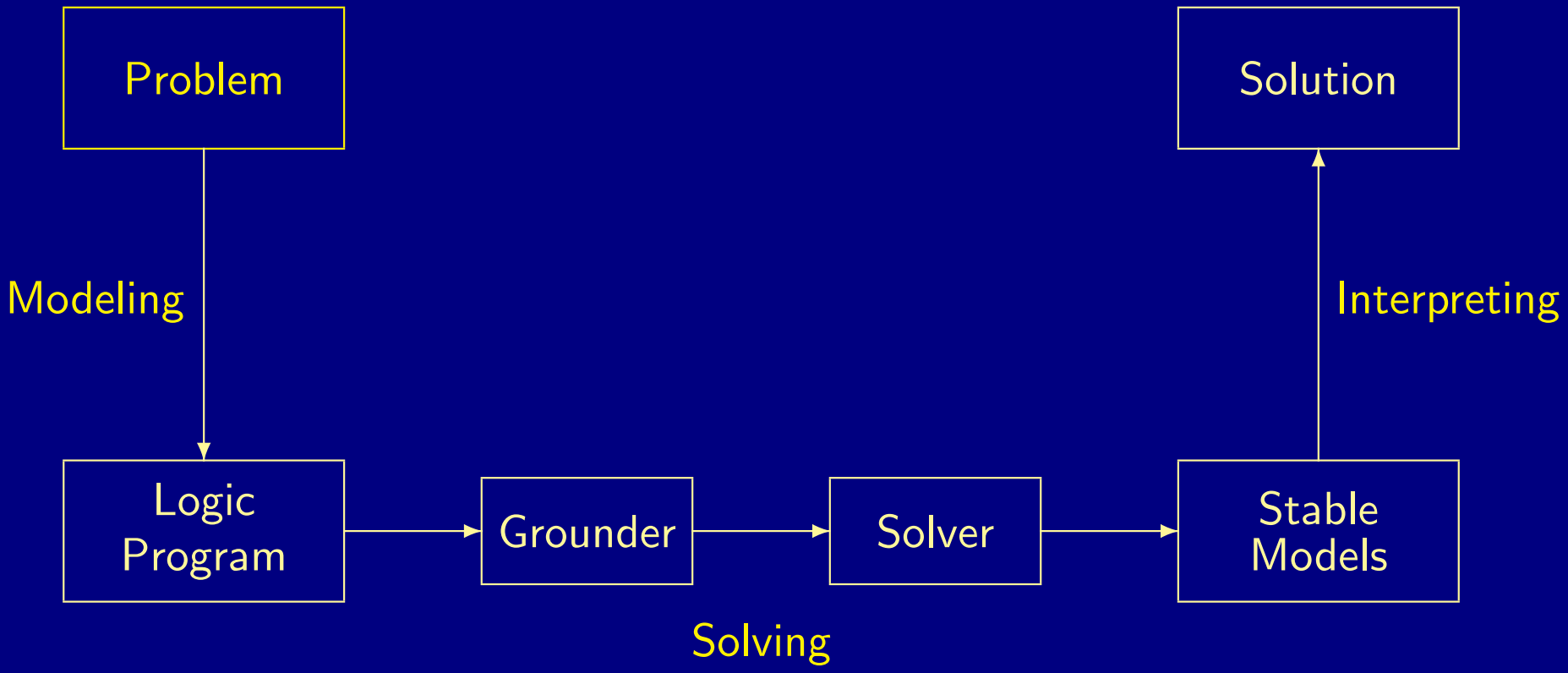
- SAT = ASP + Tertium non datur
- ASP = SAT + Completion and Loop formulas

Potassco

Outline

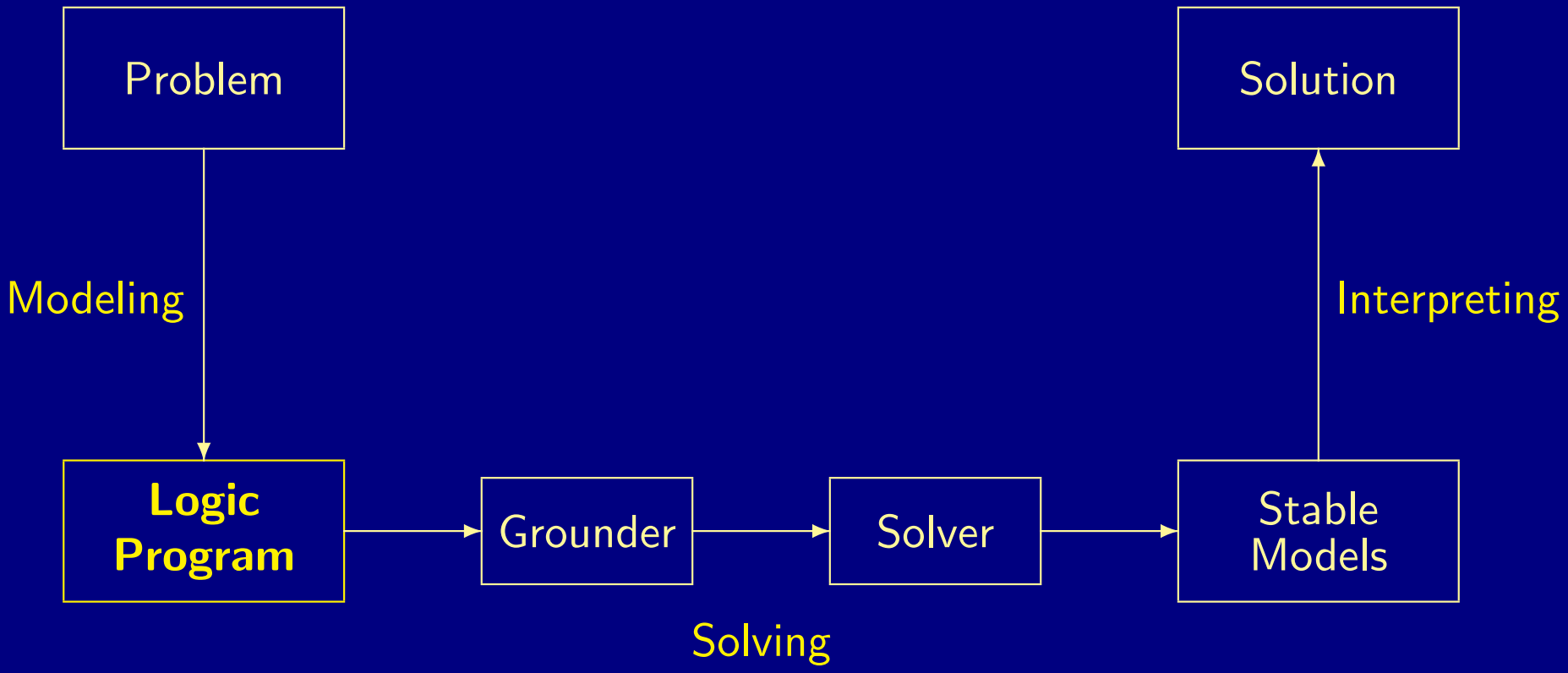
Potassco

ASP solving process



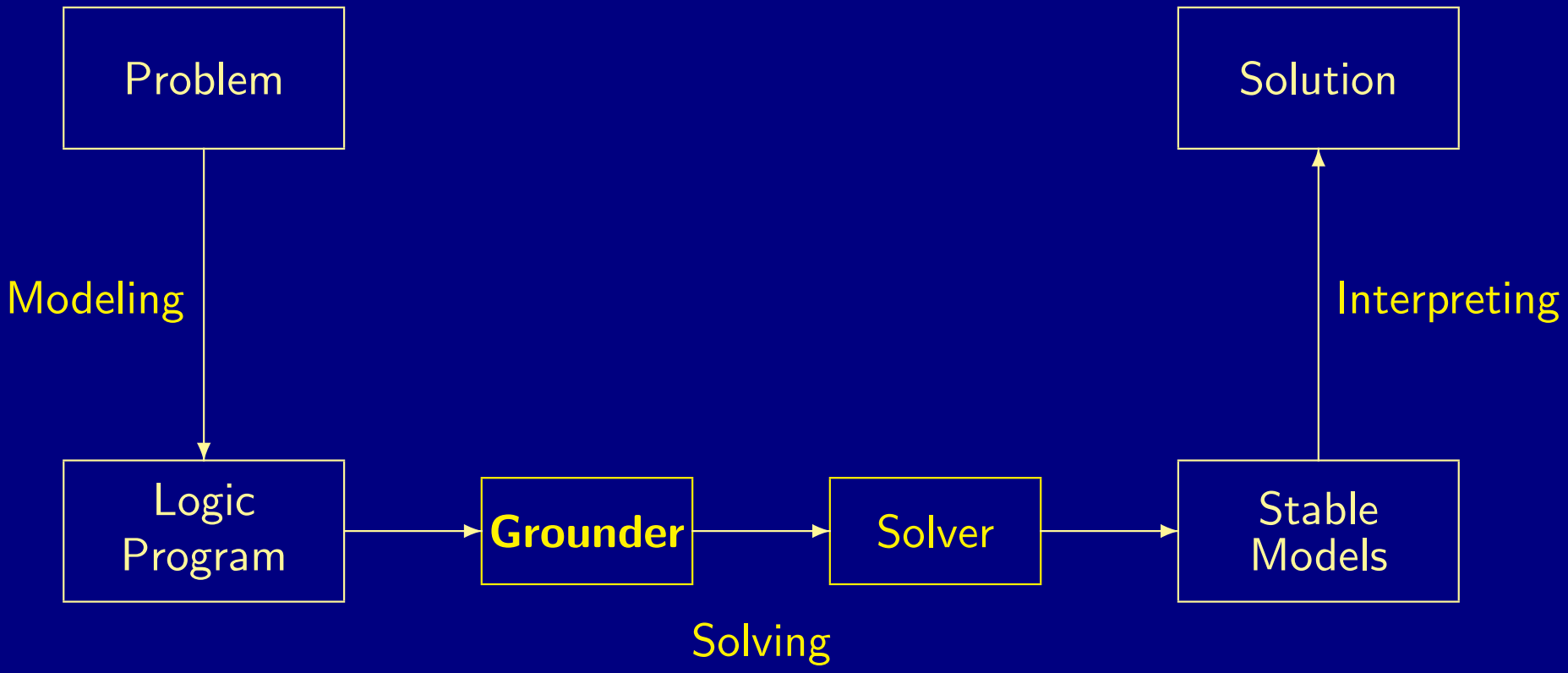
Potassco

ASP solving process



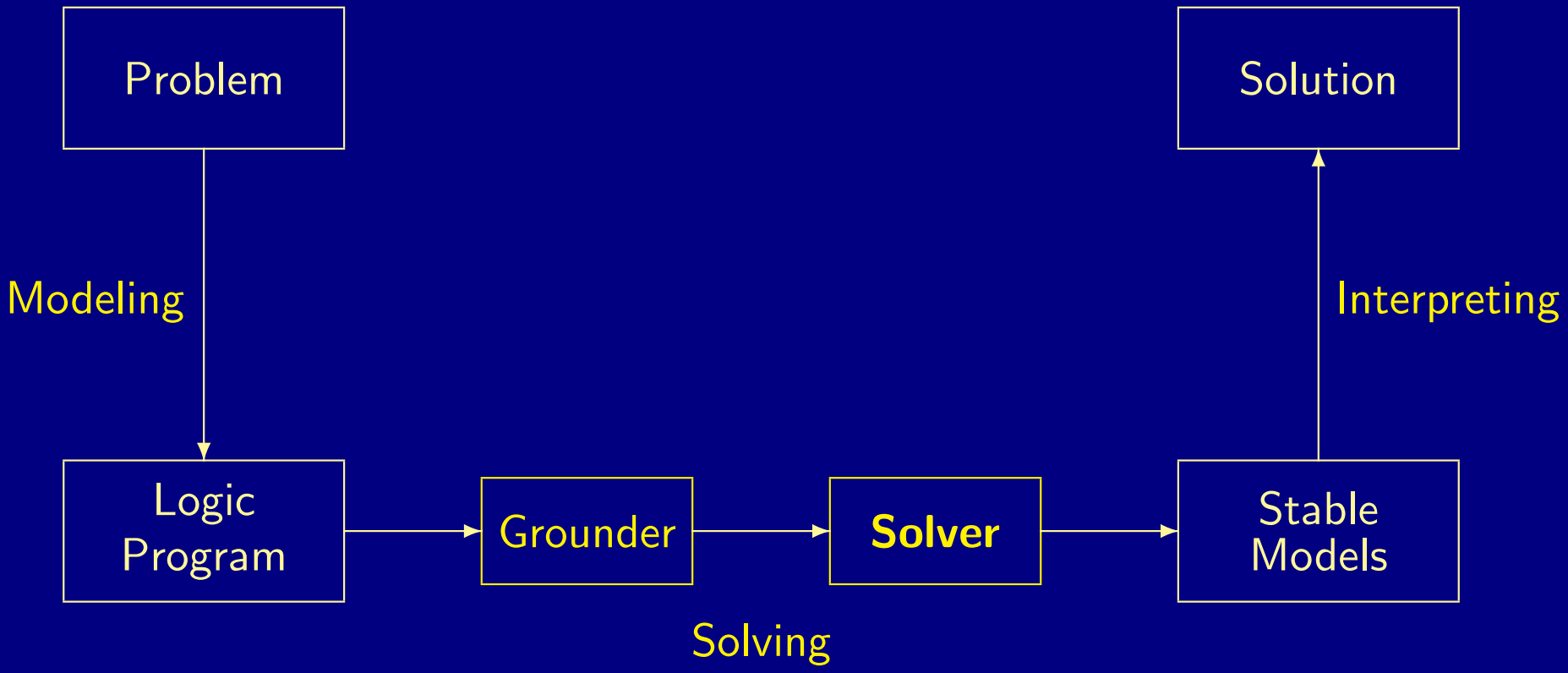
Potassco

ASP solving process



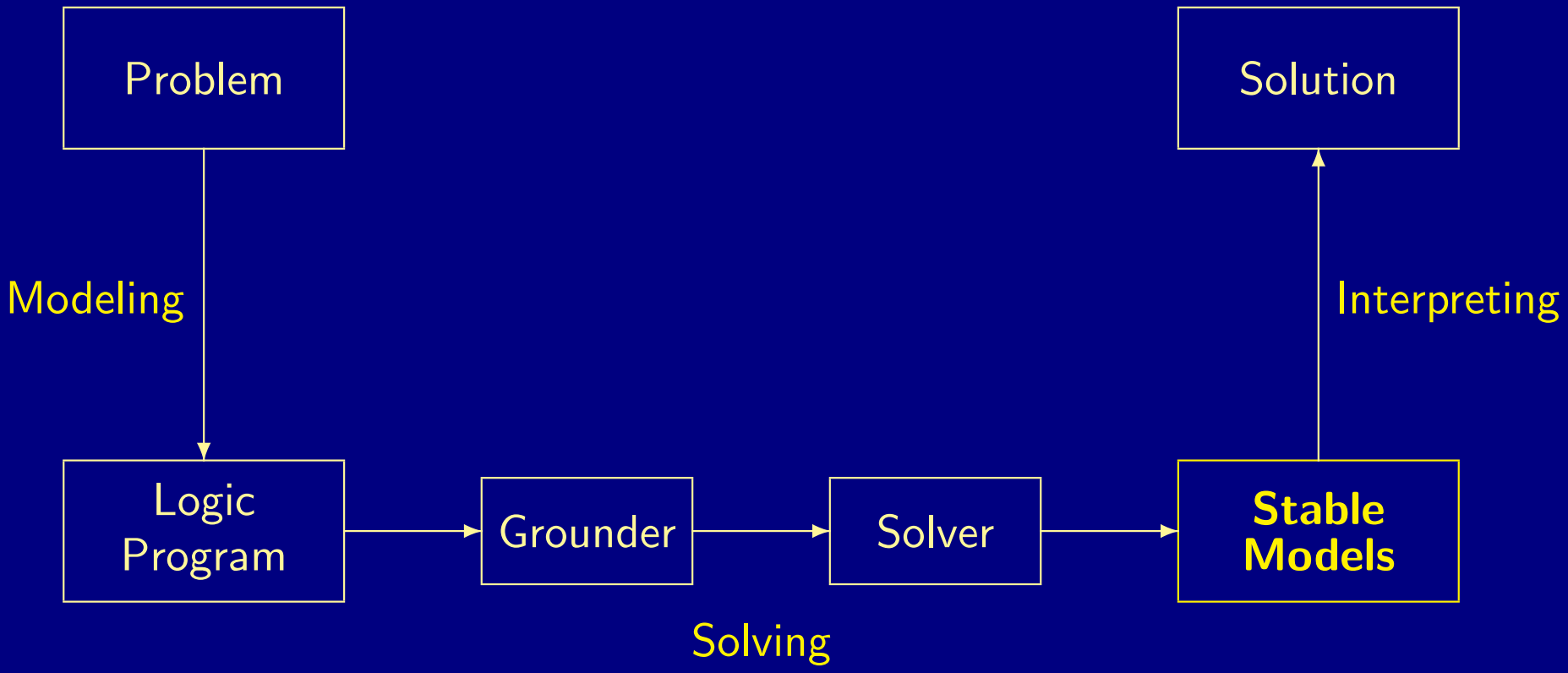
Potassco

ASP solving process



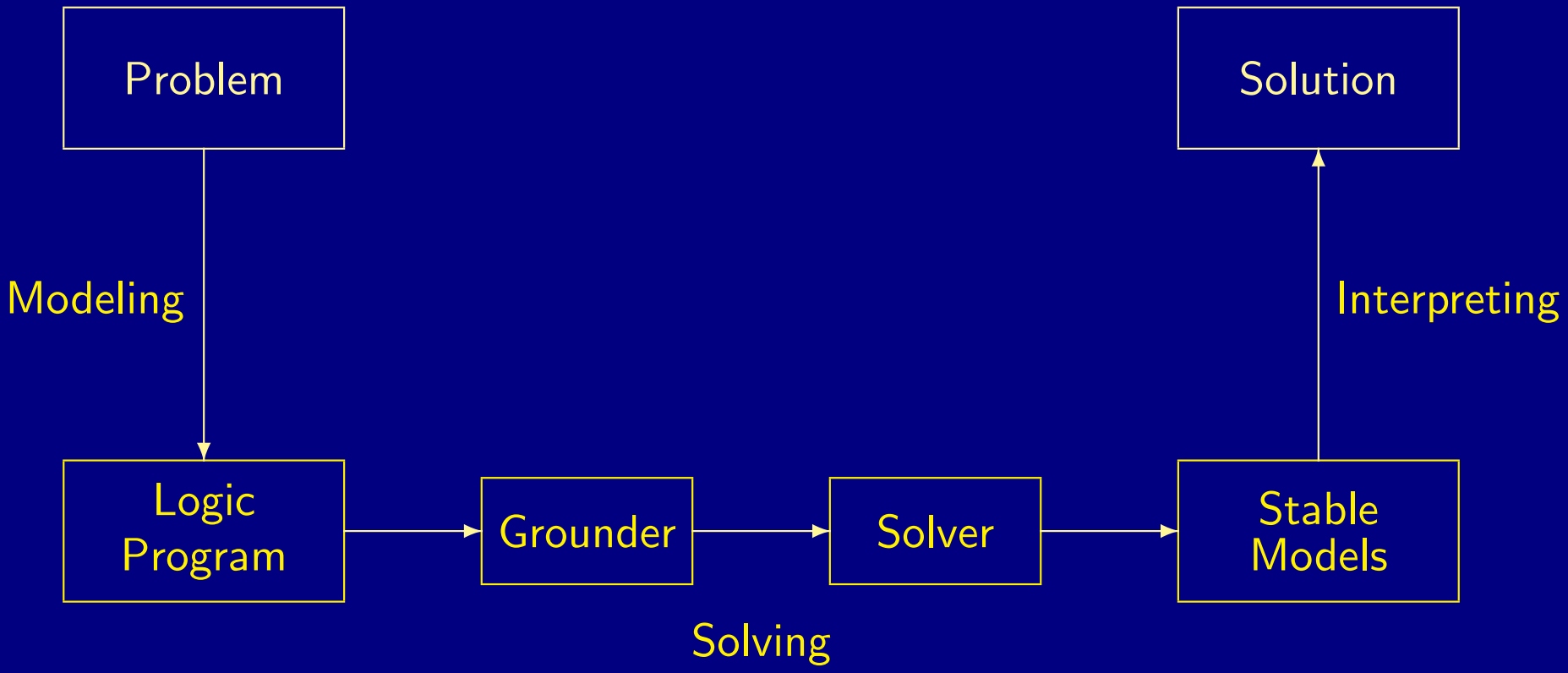
Potassco

ASP solving process



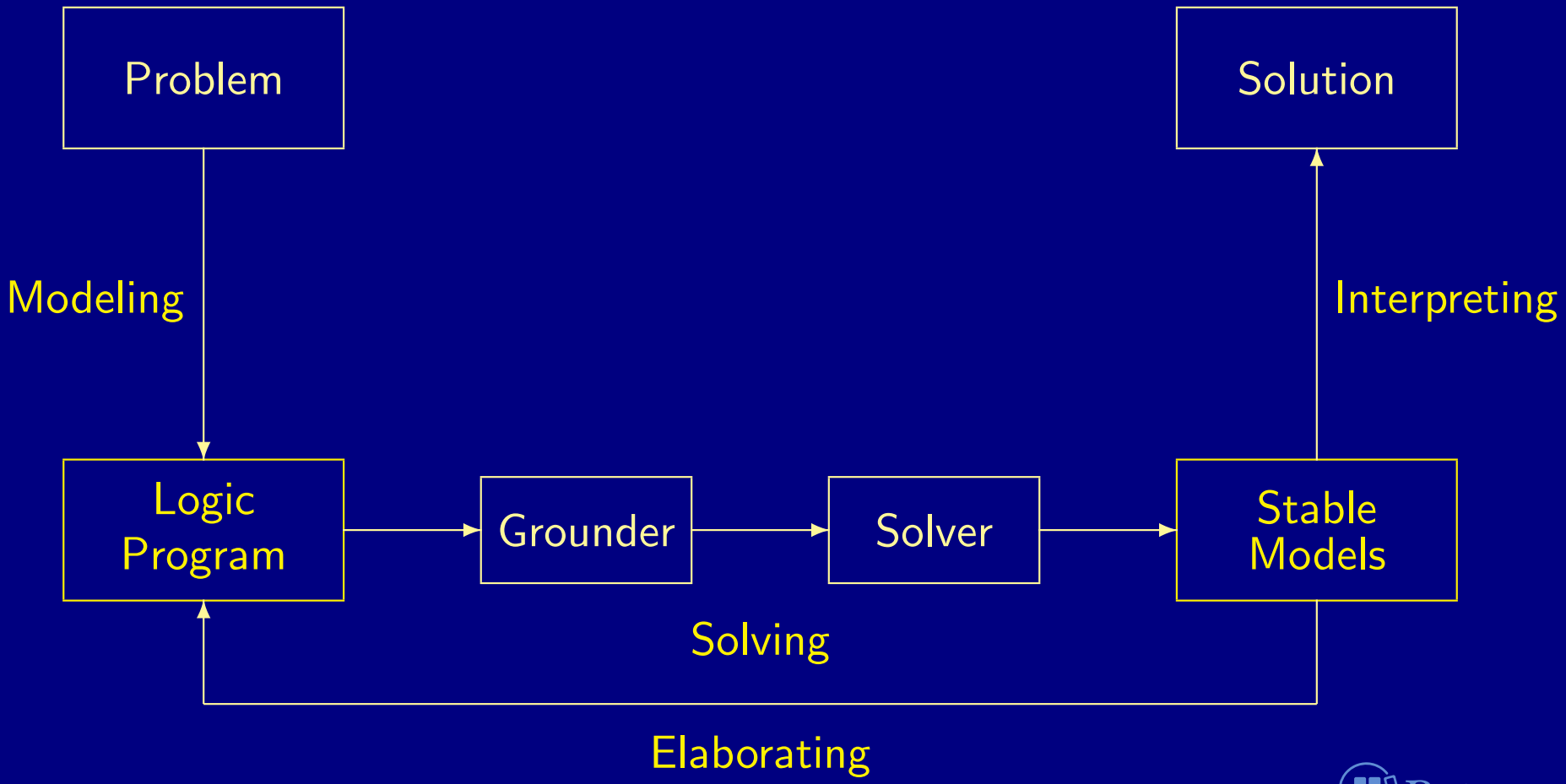
Potassco

ASP solving process



Potassco

ASP solving process



Potassco

Outline

Potassco

Guiding principle

- Elaboration Tolerance (McCarthy, 1998)

  "A formalism is elaboration tolerant [if] *it is convenient to modify a set of facts expressed in the formalism to take into account new phenomena or changed circumstances."*

- Uniform problem representation

  For solving a problem instance **I** of a problem class **C**,
  - **I** is represented as a set of facts $\Pi_{\mathbf{I}}$,
  - **C** is represented as a set of rules $\Pi_{\mathbf{C}}$, and
  - $\Pi_{\mathbf{C}}$ can be used to solve all problem instances in **C**

Potassco

Guiding principle

- Elaboration Tolerance (McCarthy, 1998)

  "A formalism is elaboration tolerant [if] *it is convenient to modify a set of facts expressed in the formalism to take into account new phenomena or changed circumstances."*

- Uniform problem representation

  For solving a problem instance **I** of a problem class **C**,
  - **I** is represented as a set of facts $\Pi_{\mathbf{I}}$,
  - **C** is represented as a set of rules $\Pi_{\mathbf{C}}$, and
  - $\Pi_{\mathbf{C}}$ can be used to solve all problem instances in **C**

Potassco

Language constructs

- Variables
- Conditional literals
- Disjunction
- Integrity constraints
- Choice
- Aggregates
- Optimization

```
p(X) :- q(X)
p :- q(X) : r(X)
p(X) ; q(X) :- r(X)
:- q(X), p(X)
2 { p(X,Y) : q(X) } 7 :- r(Y)
s(Y) :- r(Y), 2 #sum{ X : p(X,Y), q(X) } 7
```

```
:~ q(X), p(X,C) [C]
#minimize { C : q(X), p(X,C) }
```

Potassco

Language constructs

- Variables
- Conditional literals
- Disjunction
- Integrity constraints
- Choice
- Aggregates
- Optimization

```
p(X) :- q(X)
p :- q(X) : r(X)
p(X) ; q(X) :- r(X)
:- q(X), p(X)
2 { p(X,Y) : q(X) } 7 :- r(Y)
s(Y) :- r(Y), 2 #sum{ X : p(X,Y), q(X) } 7
:~ q(X), p(X,C) [C]
#minimize { C : q(X), p(X,C) }
```

Potassco

Language constructs

- Variables
- Conditional literals
- Disjunction
- Integrity constraints
- Choice
- Aggregates
- Optimization

```
p(X) :- q(X)
p :- q(X) : r(X)
p(X) ; q(X) :- r(X)
:- q(X), p(X)
2 { p(X,Y) : q(X) } 7 :- r(Y)
s(Y) :- r(Y), 2 #sum{ X : p(X,Y), q(X) } 7
:~ q(X), p(X,C) [C]
#minimize { C : q(X), p(X,C) }
```

Potassco

Language constructs

- Variables
- Conditional literals
- Disjunction
- Integrity constraints
- Choice
- Aggregates
- Optimization

```
p(X) :- q(X)
p :- q(X) : r(X)
p(X) ; q(X) :- r(X)
:- q(X), p(X)
2 { p(X,Y) : q(X) } 7 :- r(Y)
s(Y) :- r(Y), 2 #sum{ X : p(X,Y), q(X) } 7
:~ q(X), p(X,C) [C]
#minimize { C : q(X), p(X,C) }
```

Potassco

Language constructs

- Variables
- Conditional literals
- Disjunction
- Integrity constraints
- Choice
- Aggregates
- Optimization

```
p(X) :- q(X)
p :- q(X) : r(X)
p(X) ; q(X) :- r(X)
:- q(X), p(X)
2 { p(X,Y) : q(X) } 7 :- r(Y)
s(Y) :- r(Y), 2 #sum{ X : p(X,Y), q(X) } 7
:~ q(X), p(X,C) [C]
#minimize { C : q(X), p(X,C) }
```

Potassco

Language constructs

- Variables
- Conditional literals
- Disjunction
- Integrity constraints
- Choice
- Aggregates
- Optimization

```
p(X) :- q(X)
p :- q(X) : r(X)
p(X) ; q(X) :- r(X)
:- q(X), p(X)
2 { p(X,Y) : q(X) } 7 :- r(Y)
s(Y) :- r(Y), 2 #sum{ X : p(X,Y), q(X) } 7
:~ q(X), p(X,C) [C]
#minimize { C : q(X), p(X,C) }
```

Potassco

Language constructs

- Variables `p(X) :- q(X)`
- Conditional literals `p :- q(X) : r(X)`
- Disjunction `p(X) ; q(X) :- r(X)`
- Integrity constraints `:- q(X), p(X)`
- Choice `2 { p(X,Y) : q(X) } 7 :- r(Y)`
- Aggregates `s(Y) :- r(Y), 2 #sum{ X : p(X,Y), q(X) } 7`
- Optimization
  - Weak constraints `:~ q(X), p(X,C) [C]`
  - Minimize statements `#minimize { C : q(X), p(X,C) }`

Potassco

Language constructs

- Variables `p(X) :- q(X)`
- Conditional literals `p :- q(X) : r(X)`
- Disjunction `p(X) ; q(X) :- r(X)`
- Integrity constraints `:- q(X), p(X)`
- Choice `2 { p(X,Y) : q(X) } 7 :- r(Y)`
- Aggregates `s(Y) :- r(Y), 2 #sum{ X : p(X,Y), q(X) } 7`
- Optimization
  - Weak constraints `:~ q(X), p(X,C) [C]`
  - Statements `#minimize { C : q(X), p(X,C) }`

Potassco

Language constructs

- Variables `p(X) :- q(X)`
- Conditional literals `p :- q(X) : r(X)`
- Disjunction `p(X) ; q(X) :- r(X)`
- Integrity constraints `:- q(X), p(X)`
- Choice `2 { p(X,Y) : q(X) } 7 :- r(Y)`
- Aggregates `s(Y) :- r(Y), 2 #sum{ X : p(X,Y), q(X) } 7`
- Optimization
  - Weak constraints `:~ q(X), p(X,C) [C]`
  - Statements `#minimize { C : q(X), p(X,C) }`

Potassco

Language constructs

- Variables `p(X) :- q(X)`
- Conditional literals `p :- q(X) : r(X)`
- Disjunction `p(X) ; q(X) :- r(X)`
- Integrity constraints `:- q(X), p(X)`
- Choice `2 { p(X,Y) : q(X) } 7 :- r(Y)`
- Aggregates `s(Y) :- r(Y), 2 #sum{ X : p(X,Y), q(X) } 7`
- Optimization
  - Weak constraints `:~ q(X), p(X,C) [C]`
  - Statements `#minimize { C : q(X), p(X,C) }`

Potassco

Language constructs

- Variables `p(X) :- q(X)`
- Conditional literals `p :- q(X) : r(X)`
- Disjunction `p(X) ; q(X) :- r(X)`
- Integrity constraints `:- q(X), p(X)`
- Choice `2 { p(X,Y) : q(X) } 7 :- r(Y)`
- Aggregates `s(Y) :- r(Y), 2 #sum{ X : p(X,Y), q(X) } 7`
- Multi-objective optimization
  - Weak constraints `:~ q(X), p(X,C) [C@42]`
  - Statements `#minimize { C@42 : q(X), p(X,C) }`

Potassco

Satisfiability testing

$(a \leftrightarrow b) \wedge c$

Satisfiability testing

$(a \leftrightarrow b) \wedge c$

```
{ a ; b ; c }.

 :- not a, b.
 :- a, not b.
 :- not c.
```

Satisfiability testing

$(a \leftrightarrow b) \wedge c$

```
{ a ; b ; c }.

 :- not a, b.
 :- a, not b.
 :- not c.
```

- Note $\{A\}$ is an abbreviation for $A \vee \neg A$

Potassco

(Lexico) Maximum satisfiability testing

$(a \leftrightarrow b) \wedge c$

```
{ a ; b ; c }.

 :- not a, b.
 :- a, not b.
 :- not c.
```

```
{ a ; b ; c }.

 :- not a, b.
 :~ a, not b. [10@2]
 :~ not c.    [100@1]
```

- Note $\{A\}$ is an abbreviation for $A \vee \neg A$

Potassco

Traveling salesperson

Basic encoding

```
{ cycle(X,Y) : edge(X,Y) } = 1 :- node(X).
{ cycle(X,Y) : edge(X,Y) } = 1 :- node(Y).

reached(X) :- X = #min { Y : node(Y) }.
reached(Y) :- cycle(X,Y), reached(X).

:- node(Y), not reached(Y).

#minimize { C,X,Y : cycle(X,Y), cost(X,Y,C) }.

node(X) :- edge(X,_).        node(X) :- edge(_,X).

edge(X,Y) :- cost(X,Y,_).

cost(1,2,6).  cost(1,3,2).  cost(1,4,9).
cost(2,4,4).  cost(2,5,5).  cost(2,6,6).  [...]
```

Potassco

Traveling salesperson

Basic encoding

```
{ cycle(X,Y) : edge(X,Y) } = 1 :- node(X).
{ cycle(X,Y) : edge(X,Y) } = 1 :- node(Y).

reached(X) :- X = #min { Y : node(Y) }.
reached(Y) :- cycle(X,Y), reached(X).

:- node(Y), not reached(Y).

#minimize { C,X,Y : cycle(X,Y), cost(X,Y,C) }.

node(X) :- edge(X,_).        node(X) :- edge(_,X).

edge(X,Y) :- cost(X,Y,_).

cost(1,2,6).  cost(1,3,2).  cost(1,4,9).
cost(2,4,4).  cost(2,5,5).  cost(2,6,6).  [...]
```

Potassco

Traveling salesperson

Basic encoding

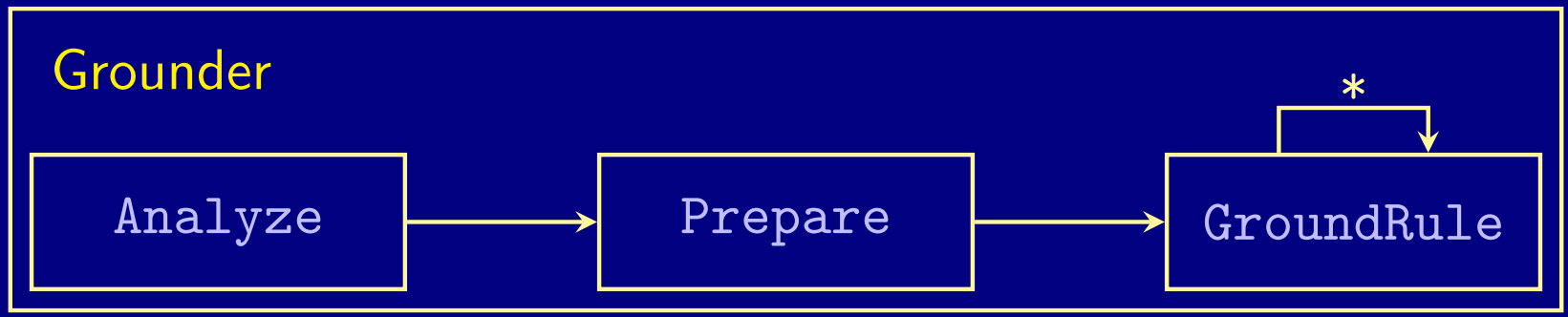
```
{ cycle(X,Y) : edge(X,Y) } = 1 :- node(X).
{ cycle(X,Y) : edge(X,Y) } = 1 :- node(Y).

reached(X) :- X = #min { Y : node(Y) }.
reached(Y) :- cycle(X,Y), reached(X).

:- node(Y), not reached(Y).

#minimize { C,X,Y : cycle(X,Y), cost(X,Y,C) }.

node(X) :- edge(X,_).        node(X) :- edge(_,X).

edge(X,Y) :- cost(X,Y,_).

cost(1,2,6).  cost(1,3,2).  cost(1,4,9).
cost(2,4,4).  cost(2,5,5).  cost(2,6,6).  [...]
```

Potassco

Outline

Potassco

Grounding Logic Programs

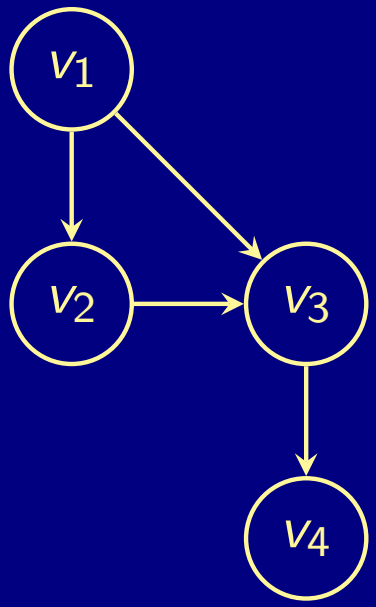


- Grounding algorithm uses three functions

`Analyze`	groups rules into components and determines recursive predicates
`Prepare`	rewrites rules based on recursive predicates
`GroundRule`	instantiates rules iteratively following semi-naive database evaluation

Potassco

Reachability Problem

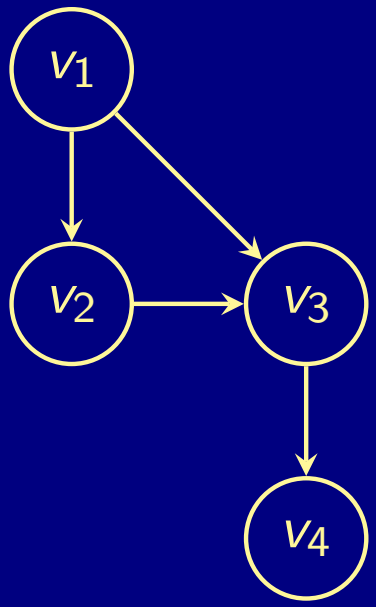


$\texttt{vertex}(\texttt{v}_1).$ $\texttt{edge}(\texttt{v}_1, \texttt{v}_2).$
$\texttt{vertex}(\texttt{v}_2).$ $\texttt{edge}(\texttt{v}_1, \texttt{v}_3).$
$\texttt{vertex}(\texttt{v}_3).$ $\texttt{edge}(\texttt{v}_2, \texttt{v}_3).$
$\texttt{vertex}(\texttt{v}_4).$ $\texttt{edge}(\texttt{v}_3, \texttt{v}_4).$

$$\texttt{reach}(\texttt{X}, \texttt{Y}) \texttt{ :- } \texttt{edge}(\texttt{X}, \texttt{Y}). \quad (1)$$
$$\texttt{reach}(\texttt{X}, \texttt{Y}) \texttt{ :- } \texttt{reach}(\texttt{X}, \texttt{Z}), \texttt{edge}(\texttt{Z}, \texttt{Y}). \quad (2)$$

Potassco

Reachability Problem

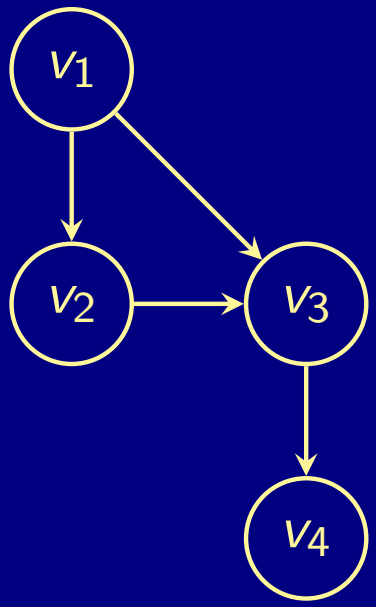


$\texttt{vertex}(\texttt{v}_1).$ $\texttt{edge}(\texttt{v}_1, \texttt{v}_2).$
$\texttt{vertex}(\texttt{v}_2).$ $\texttt{edge}(\texttt{v}_1, \texttt{v}_3).$
$\texttt{vertex}(\texttt{v}_3).$ $\texttt{edge}(\texttt{v}_2, \texttt{v}_3).$
$\texttt{vertex}(\texttt{v}_4).$ $\texttt{edge}(\texttt{v}_3, \texttt{v}_4).$

$$\texttt{reach}(\texttt{X}, \texttt{Y}) \texttt{ :- } \texttt{edge}(\texttt{X}, \texttt{Y}). \quad (1)$$
$$\texttt{reach}(\texttt{X}, \texttt{Y}) \texttt{ :- } \texttt{reach}(\texttt{X}, \texttt{Z}), \texttt{edge}(\texttt{Z}, \texttt{Y}). \quad (2)$$

Potassco

Reachability Problem



$\texttt{vertex}(\texttt{v}_1).$ $\texttt{edge}(\texttt{v}_1, \texttt{v}_2).$
$\texttt{vertex}(\texttt{v}_2).$ $\texttt{edge}(\texttt{v}_1, \texttt{v}_3).$
$\texttt{vertex}(\texttt{v}_3).$ $\texttt{edge}(\texttt{v}_2, \texttt{v}_3).$
$\texttt{vertex}(\texttt{v}_4).$ $\texttt{edge}(\texttt{v}_3, \texttt{v}_4).$

$\texttt{reach}(\texttt{X}, \texttt{Y}) \texttt{ :- } \texttt{edge}(\texttt{X}, \texttt{Y}).$ $\rightsquigarrow$ 16 rules (1)

$\texttt{reach}(\texttt{X}, \texttt{Y}) \texttt{ :- } \texttt{reach}(\texttt{X}, \texttt{Z}), \texttt{edge}(\texttt{Z}, \texttt{Y}).$ $\rightsquigarrow$ 64 rules (2)

Potassco

Grounding

$0 \quad \texttt{edge}(\texttt{v}_1, \texttt{v}_2), \texttt{edge}(\texttt{v}_1, \texttt{v}_3), \texttt{edge}(\texttt{v}_2, \texttt{v}_3), \texttt{edge}(\texttt{v}_3, \texttt{v}_4), \ldots$

Potassco

Grounding

0	$\texttt{edge}(\texttt{v}_1,\texttt{v}_2),\texttt{edge}(\texttt{v}_1,\texttt{v}_3),\texttt{edge}(\texttt{v}_2,\texttt{v}_3),\texttt{edge}(\texttt{v}_3,\texttt{v}_4),\ldots$	
1	$\texttt{reach}(\texttt{v}_1,\texttt{v}_2)$:- $\texttt{edge}(\texttt{v}_1,\texttt{v}_2)$	(1)
	$\texttt{reach}(\texttt{v}_1,\texttt{v}_3)$:- $\texttt{edge}(\texttt{v}_1,\texttt{v}_3)$	(1)
	$\texttt{reach}(\texttt{v}_2,\texttt{v}_3)$:- $\texttt{edge}(\texttt{v}_2,\texttt{v}_3)$	(1)
	$\texttt{reach}(\texttt{v}_3,\texttt{v}_4)$:- $\texttt{edge}(\texttt{v}_3,\texttt{v}_4)$	(1)

Potassco

Grounding

0	$\texttt{edge}(\texttt{v}_1,\texttt{v}_2),\texttt{edge}(\texttt{v}_1,\texttt{v}_3),\texttt{edge}(\texttt{v}_2,\texttt{v}_3),\texttt{edge}(\texttt{v}_3,\texttt{v}_4),\ldots$	
1	$\texttt{reach}(\texttt{v}_1,\texttt{v}_2)$:- $\texttt{edge}(\texttt{v}_1,\texttt{v}_2)$	(1)
	$\texttt{reach}(\texttt{v}_1,\texttt{v}_3)$:- $\texttt{edge}(\texttt{v}_1,\texttt{v}_3)$	(1)
	$\texttt{reach}(\texttt{v}_2,\texttt{v}_3)$:- $\texttt{edge}(\texttt{v}_2,\texttt{v}_3)$	(1)
	$\texttt{reach}(\texttt{v}_3,\texttt{v}_4)$:- $\texttt{edge}(\texttt{v}_3,\texttt{v}_4)$	(1)
2	$\texttt{reach}(\texttt{v}_1,\texttt{v}_3)$:- $\texttt{edge}(\texttt{v}_2,\texttt{v}_3), \texttt{reach}(\texttt{v}_1,\texttt{v}_2)$	(2)
	$\texttt{reach}(\texttt{v}_1,\texttt{v}_4)$:- $\texttt{edge}(\texttt{v}_3,\texttt{v}_4), \texttt{reach}(\texttt{v}_1,\texttt{v}_3)$	(2)
	$\texttt{reach}(\texttt{v}_2,\texttt{v}_4)$:- $\texttt{edge}(\texttt{v}_3,\texttt{v}_4), \texttt{reach}(\texttt{v}_2,\texttt{v}_3)$	(2)

Potassco

Grounding

0	$\texttt{edge}(\texttt{v}_1,\texttt{v}_2), \texttt{edge}(\texttt{v}_1,\texttt{v}_3), \texttt{edge}(\texttt{v}_2,\texttt{v}_3), \texttt{edge}(\texttt{v}_3,\texttt{v}_4), \ldots$	
1	$\texttt{reach}(\texttt{v}_1,\texttt{v}_2)$:- $\texttt{edge}(\texttt{v}_1,\texttt{v}_2)$	(1)
	$\texttt{reach}(\texttt{v}_1,\texttt{v}_3)$:- $\texttt{edge}(\texttt{v}_1,\texttt{v}_3)$	(1)
	$\texttt{reach}(\texttt{v}_2,\texttt{v}_3)$:- $\texttt{edge}(\texttt{v}_2,\texttt{v}_3)$	(1)
	$\texttt{reach}(\texttt{v}_3,\texttt{v}_4)$:- $\texttt{edge}(\texttt{v}_3,\texttt{v}_4)$	(1)
2	$\texttt{reach}(\texttt{v}_1,\texttt{v}_3)$:- $\texttt{edge}(\texttt{v}_2,\texttt{v}_3), \texttt{reach}(\texttt{v}_1,\texttt{v}_2)$	(2)
	$\texttt{reach}(\texttt{v}_1,\texttt{v}_4)$:- $\texttt{edge}(\texttt{v}_3,\texttt{v}_4), \texttt{reach}(\texttt{v}_1,\texttt{v}_3)$	(2)
	$\texttt{reach}(\texttt{v}_2,\texttt{v}_4)$:- $\texttt{edge}(\texttt{v}_3,\texttt{v}_4), \texttt{reach}(\texttt{v}_2,\texttt{v}_3)$	(2)
3	$\texttt{reach}(\texttt{v}_1,\texttt{v}_4)$:- $\texttt{edge}(\texttt{v}_3,\texttt{v}_4), \texttt{reach}(\texttt{v}_1,\texttt{v}_3)$	(2)

Potassco

Simplifying while Grounding

0	$\texttt{edge}(\texttt{v}_1,\texttt{v}_2), \texttt{edge}(\texttt{v}_1,\texttt{v}_3), \texttt{edge}(\texttt{v}_2,\texttt{v}_3), \texttt{edge}(\texttt{v}_3,\texttt{v}_4), \ldots$	
1	$\texttt{reach}(\texttt{v}_1,\texttt{v}_2)$:- $\texttt{edge}(\texttt{v}_1,\texttt{v}_2)$	(1)
	$\texttt{reach}(\texttt{v}_1,\texttt{v}_3)$:- $\texttt{edge}(\texttt{v}_1,\texttt{v}_3)$	(1)
	$\texttt{reach}(\texttt{v}_2,\texttt{v}_3)$:- $\texttt{edge}(\texttt{v}_2,\texttt{v}_3)$	(1)
	$\texttt{reach}(\texttt{v}_3,\texttt{v}_4)$:- $\texttt{edge}(\texttt{v}_3,\texttt{v}_4)$	(1)
2	$\texttt{reach}(\texttt{v}_1,\texttt{v}_3)$:- $\texttt{edge}(\texttt{v}_2,\texttt{v}_3), \texttt{reach}(\texttt{v}_1,\texttt{v}_2)$	(2)
	$\texttt{reach}(\texttt{v}_1,\texttt{v}_4)$:- $\texttt{edge}(\texttt{v}_3,\texttt{v}_4), \texttt{reach}(\texttt{v}_1,\texttt{v}_3)$	(2)
	$\texttt{reach}(\texttt{v}_2,\texttt{v}_4)$:- $\texttt{edge}(\texttt{v}_3,\texttt{v}_4), \texttt{reach}(\texttt{v}_2,\texttt{v}_3)$	(2)
3	$\texttt{reach}(\texttt{v}_1,\texttt{v}_4)$:- $\texttt{edge}(\texttt{v}_3,\texttt{v}_4), \texttt{reach}(\texttt{v}_1,\texttt{v}_3)$	(2)

Potassco

Simplifying while Grounding

0	$\texttt{edge}(\texttt{v}_1,\texttt{v}_2),\texttt{edge}(\texttt{v}_1,\texttt{v}_3),\texttt{edge}(\texttt{v}_2,\texttt{v}_3),\texttt{edge}(\texttt{v}_3,\texttt{v}_4),\ldots$	
1	$\texttt{reach}(\texttt{v}_1,\texttt{v}_2)$	(1)
	$\texttt{reach}(\texttt{v}_1,\texttt{v}_3)$	(1)
	$\texttt{reach}(\texttt{v}_2,\texttt{v}_3)$	(1)
	$\texttt{reach}(\texttt{v}_3,\texttt{v}_4)$	(1)
2	$\texttt{reach}(\texttt{v}_1,\texttt{v}_3)$:- $\texttt{edge}(\texttt{v}_2,\texttt{v}_3),\ \texttt{reach}(\texttt{v}_1,\texttt{v}_2)$	(2)
	$\texttt{reach}(\texttt{v}_1,\texttt{v}_4)$:- $\texttt{edge}(\texttt{v}_3,\texttt{v}_4),\ \texttt{reach}(\texttt{v}_1,\texttt{v}_3)$	(2)
	$\texttt{reach}(\texttt{v}_2,\texttt{v}_4)$:- $\texttt{edge}(\texttt{v}_3,\texttt{v}_4),\ \texttt{reach}(\texttt{v}_2,\texttt{v}_3)$	(2)
3	$\texttt{reach}(\texttt{v}_1,\texttt{v}_4)$:- $\texttt{edge}(\texttt{v}_3,\texttt{v}_4),\ \texttt{reach}(\texttt{v}_1,\texttt{v}_3)$	(2)

Potassco

Simplifying while Grounding

0	$\texttt{edge}(\texttt{v}_1,\texttt{v}_2), \texttt{edge}(\texttt{v}_1,\texttt{v}_3), \texttt{edge}(\texttt{v}_2,\texttt{v}_3), \texttt{edge}(\texttt{v}_3,\texttt{v}_4), \ldots$	
1	$\texttt{reach}(\texttt{v}_1,\texttt{v}_2)$	(1)
	$\texttt{reach}(\texttt{v}_1,\texttt{v}_3)$	(1)
	$\texttt{reach}(\texttt{v}_2,\texttt{v}_3)$	(1)
	$\texttt{reach}(\texttt{v}_3,\texttt{v}_4)$	(1)
2	$\texttt{reach}(\texttt{v}_1,\texttt{v}_3)$:- $\texttt{reach}(\texttt{v}_1,\texttt{v}_2)$	(2)
	$\texttt{reach}(\texttt{v}_1,\texttt{v}_4)$:- $\texttt{reach}(\texttt{v}_1,\texttt{v}_3)$	(2)
	$\texttt{reach}(\texttt{v}_2,\texttt{v}_4)$:- $\texttt{reach}(\texttt{v}_2,\texttt{v}_3)$	(2)
3	$\texttt{reach}(\texttt{v}_1,\texttt{v}_4)$:- $\texttt{edge}(\texttt{v}_3,\texttt{v}_4), \texttt{reach}(\texttt{v}_1,\texttt{v}_3)$	(2)

Potassco

Simplifying while Grounding

0	$\texttt{edge}(\texttt{v}_1,\texttt{v}_2), \texttt{edge}(\texttt{v}_1,\texttt{v}_3), \texttt{edge}(\texttt{v}_2,\texttt{v}_3), \texttt{edge}(\texttt{v}_3,\texttt{v}_4), \ldots$	
1	$\texttt{reach}(\texttt{v}_1,\texttt{v}_2)$	(1)
	$\texttt{reach}(\texttt{v}_1,\texttt{v}_3)$	(1)
	$\texttt{reach}(\texttt{v}_2,\texttt{v}_3)$	(1)
	$\texttt{reach}(\texttt{v}_3,\texttt{v}_4)$	(1)
2	$\texttt{reach}(\texttt{v}_1,\texttt{v}_3)$	(2)
	$\texttt{reach}(\texttt{v}_1,\texttt{v}_4)$	(2)
	$\texttt{reach}(\texttt{v}_2,\texttt{v}_4)$	(2)
3	$\texttt{reach}(\texttt{v}_1,\texttt{v}_4)$:- $\texttt{edge}(\texttt{v}_3,\texttt{v}_4), \texttt{reach}(\texttt{v}_1,\texttt{v}_3)$	(2)

Potassco

Simplifying while Grounding

0	$\texttt{edge}(\texttt{v}_1,\texttt{v}_2), \texttt{edge}(\texttt{v}_1,\texttt{v}_3), \texttt{edge}(\texttt{v}_2,\texttt{v}_3), \texttt{edge}(\texttt{v}_3,\texttt{v}_4), \ldots$	
1	$\texttt{reach}(\texttt{v}_1,\texttt{v}_2)$	(1)
	$\texttt{reach}(\texttt{v}_1,\texttt{v}_3)$	(1)
	$\texttt{reach}(\texttt{v}_2,\texttt{v}_3)$	(1)
	$\texttt{reach}(\texttt{v}_3,\texttt{v}_4)$	(1)
2	$\texttt{reach}(\texttt{v}_1,\texttt{v}_3)$	(2)
	$\texttt{reach}(\texttt{v}_1,\texttt{v}_4)$	(2)
	$\texttt{reach}(\texttt{v}_2,\texttt{v}_4)$	(2)
3	$\texttt{reach}(\texttt{v}_1,\texttt{v}_4) \texttt{ :- } \texttt{reach}(\texttt{v}_1,\texttt{v}_3)$	(2)

Potassco

Simplifying while Grounding

0	$\texttt{edge}(\texttt{v}_1,\texttt{v}_2), \texttt{edge}(\texttt{v}_1,\texttt{v}_3), \texttt{edge}(\texttt{v}_2,\texttt{v}_3), \texttt{edge}(\texttt{v}_3,\texttt{v}_4), \ldots$	
1	$\texttt{reach}(\texttt{v}_1,\texttt{v}_2)$	(1)
	$\texttt{reach}(\texttt{v}_1,\texttt{v}_3)$	(1)
	$\texttt{reach}(\texttt{v}_2,\texttt{v}_3)$	(1)
	$\texttt{reach}(\texttt{v}_3,\texttt{v}_4)$	(1)
2	$\texttt{reach}(\texttt{v}_1,\texttt{v}_3)$	(2)
	$\texttt{reach}(\texttt{v}_1,\texttt{v}_4)$	(2)
	$\texttt{reach}(\texttt{v}_2,\texttt{v}_4)$	(2)
3	$\texttt{reach}(\texttt{v}_1,\texttt{v}_4)$	(2)

Potassco

Simplifying while Grounding

0	$\texttt{edge}(\texttt{v}_1,\texttt{v}_2), \texttt{edge}(\texttt{v}_1,\texttt{v}_3), \texttt{edge}(\texttt{v}_2,\texttt{v}_3), \texttt{edge}(\texttt{v}_3,\texttt{v}_4), \ldots$	
1	$\texttt{reach}(\texttt{v}_1,\texttt{v}_2)$	(1)
	$\texttt{reach}(\texttt{v}_1,\texttt{v}_3)$	(1)
	$\texttt{reach}(\texttt{v}_2,\texttt{v}_3)$	(1)
	$\texttt{reach}(\texttt{v}_3,\texttt{v}_4)$	(1)
2	$\texttt{reach}(\texttt{v}_1,\texttt{v}_3)$	(2)
	$\texttt{reach}(\texttt{v}_1,\texttt{v}_4)$	(2)
	$\texttt{reach}(\texttt{v}_2,\texttt{v}_4)$	(2)
3	$\texttt{reach}(\texttt{v}_1,\texttt{v}_4)$	(2)

- Similar simplifications are done with negative information

Potassco

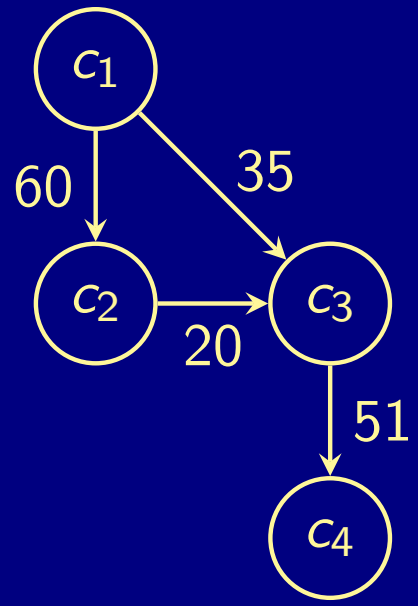
Simplifying while Grounding

0	$\texttt{edge}(\texttt{v}_1,\texttt{v}_2), \texttt{edge}(\texttt{v}_1,\texttt{v}_3), \texttt{edge}(\texttt{v}_2,\texttt{v}_3), \texttt{edge}(\texttt{v}_3,\texttt{v}_4), \ldots$	
1	$\texttt{reach}(\texttt{v}_1,\texttt{v}_2)$	(1)
	$\texttt{reach}(\texttt{v}_1,\texttt{v}_3)$	(1)
	$\texttt{reach}(\texttt{v}_2,\texttt{v}_3)$	(1)
	$\texttt{reach}(\texttt{v}_3,\texttt{v}_4)$	(1)
2	$\texttt{reach}(\texttt{v}_1,\texttt{v}_3)$	(2)
	$\texttt{reach}(\texttt{v}_1,\texttt{v}_4)$	(2)
	$\texttt{reach}(\texttt{v}_2,\texttt{v}_4)$	(2)
3	$\texttt{reach}(\texttt{v}_1,\texttt{v}_4)$	(2)

- Similar simplifications are done with negative information
- Grounding is sufficient for solving this reachability problem

Potassco

Company Controls Problem

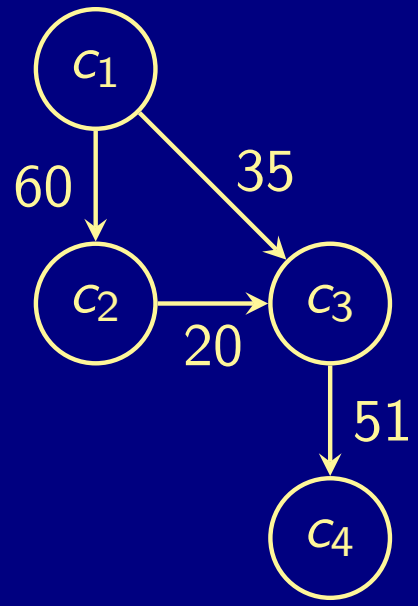


$\texttt{company}(\texttt{c}_1).$ $\texttt{owns}(\texttt{c}_1, \texttt{c}_2, 60).$
$\texttt{company}(\texttt{c}_2).$ $\texttt{owns}(\texttt{c}_2, \texttt{c}_3, 20).$
$\texttt{company}(\texttt{c}_3).$ $\texttt{owns}(\texttt{c}_1, \texttt{c}_3, 35).$
$\texttt{company}(\texttt{c}_4).$ $\texttt{owns}(\texttt{c}_3, \texttt{c}_4, 51).$

$$\texttt{controls}(\texttt{X}, \texttt{Y}) \texttt{ :-}$$
$$\#\texttt{sum}\,\{\texttt{S} : \texttt{owns}(\texttt{X}, \texttt{Y}, \texttt{S});$$
$$\texttt{S}, \texttt{Z} : \texttt{controls}(\texttt{X}, \texttt{Z}), \texttt{owns}(\texttt{Z}, \texttt{Y}, \texttt{S})\} > 50,$$
$$\texttt{company}(\texttt{X}), \texttt{company}(\texttt{Y}), \texttt{X} \neq \texttt{Y}.$$

Potassco

Company Controls Problem

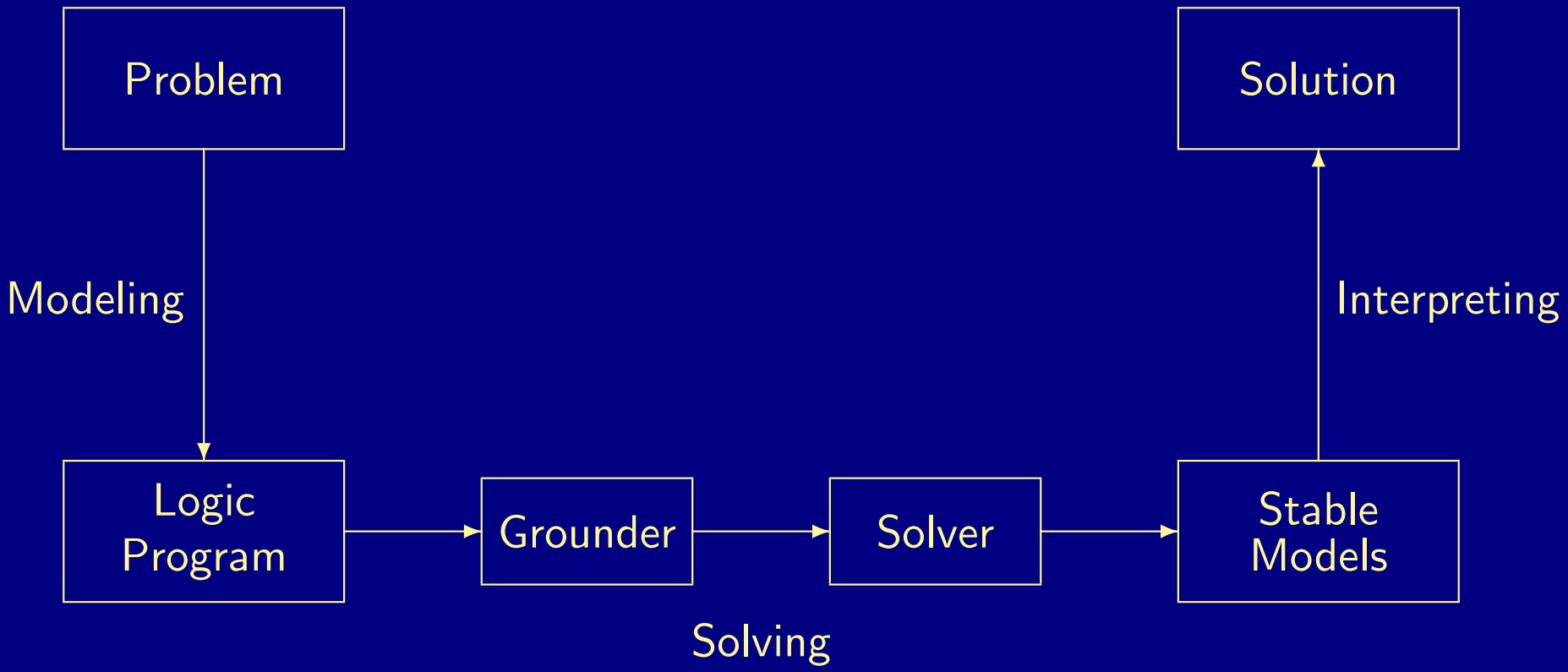


$\texttt{company}(\texttt{c}_1).$ $\texttt{owns}(\texttt{c}_1, \texttt{c}_2, 60).$
$\texttt{company}(\texttt{c}_2).$ $\texttt{owns}(\texttt{c}_2, \texttt{c}_3, 20).$
$\texttt{company}(\texttt{c}_3).$ $\texttt{owns}(\texttt{c}_1, \texttt{c}_3, 35).$
$\texttt{company}(\texttt{c}_4).$ $\texttt{owns}(\texttt{c}_3, \texttt{c}_4, 51).$

$$\texttt{controls}(\texttt{X}, \texttt{Y}) \texttt{ :-}$$
$$\#\texttt{sum}\,\{\texttt{S} : \texttt{owns}(\texttt{X}, \texttt{Y}, \texttt{S});$$
$$\texttt{S}, \texttt{Z} : \texttt{controls}(\texttt{X}, \texttt{Z}), \texttt{owns}(\texttt{Z}, \texttt{Y}, \texttt{S})\} > 50,$$
$$\texttt{company}(\texttt{X}), \texttt{company}(\texttt{Y}), \texttt{X} \neq \texttt{Y}.$$

Potassco

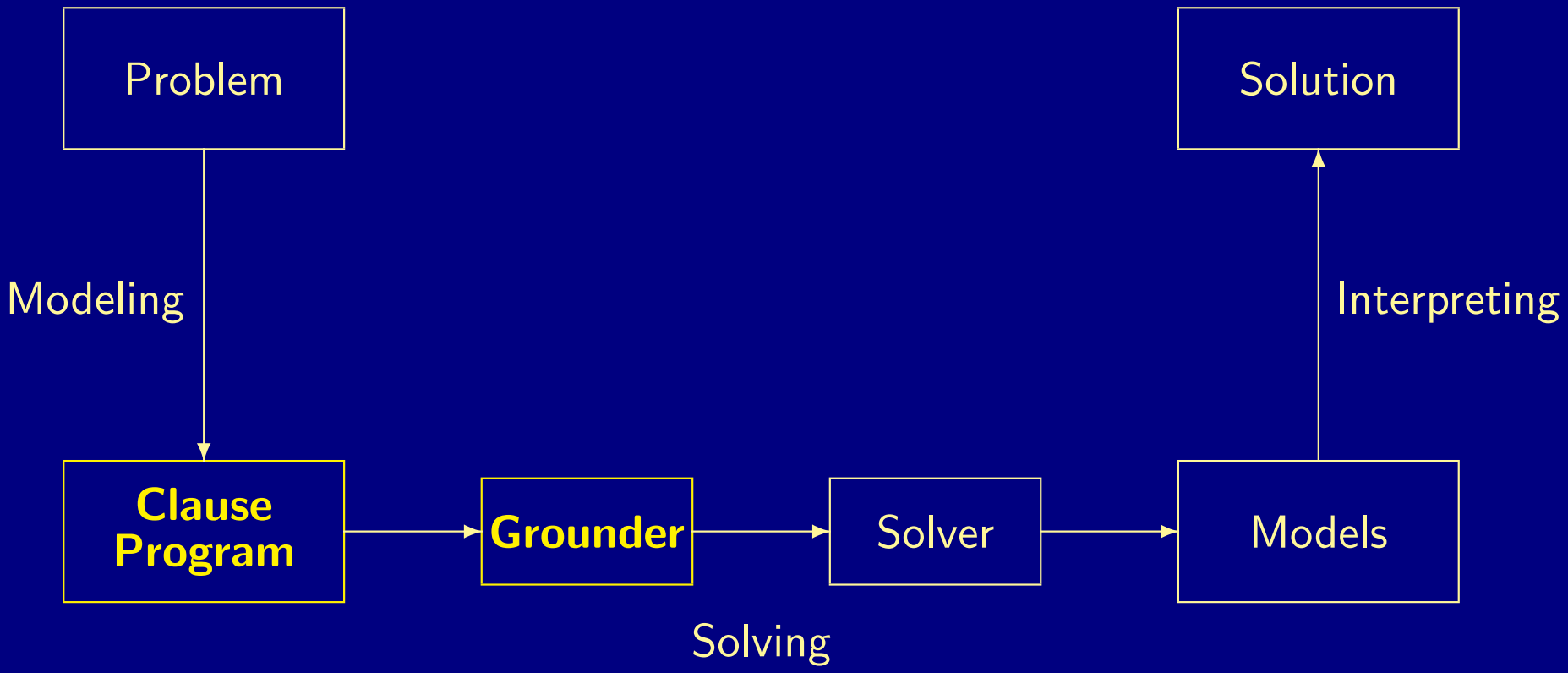
ASP Grounding



- `satgrnd` is available at `http://research.ics.aalto.fi/software/sat/satgrnd`

Potassco

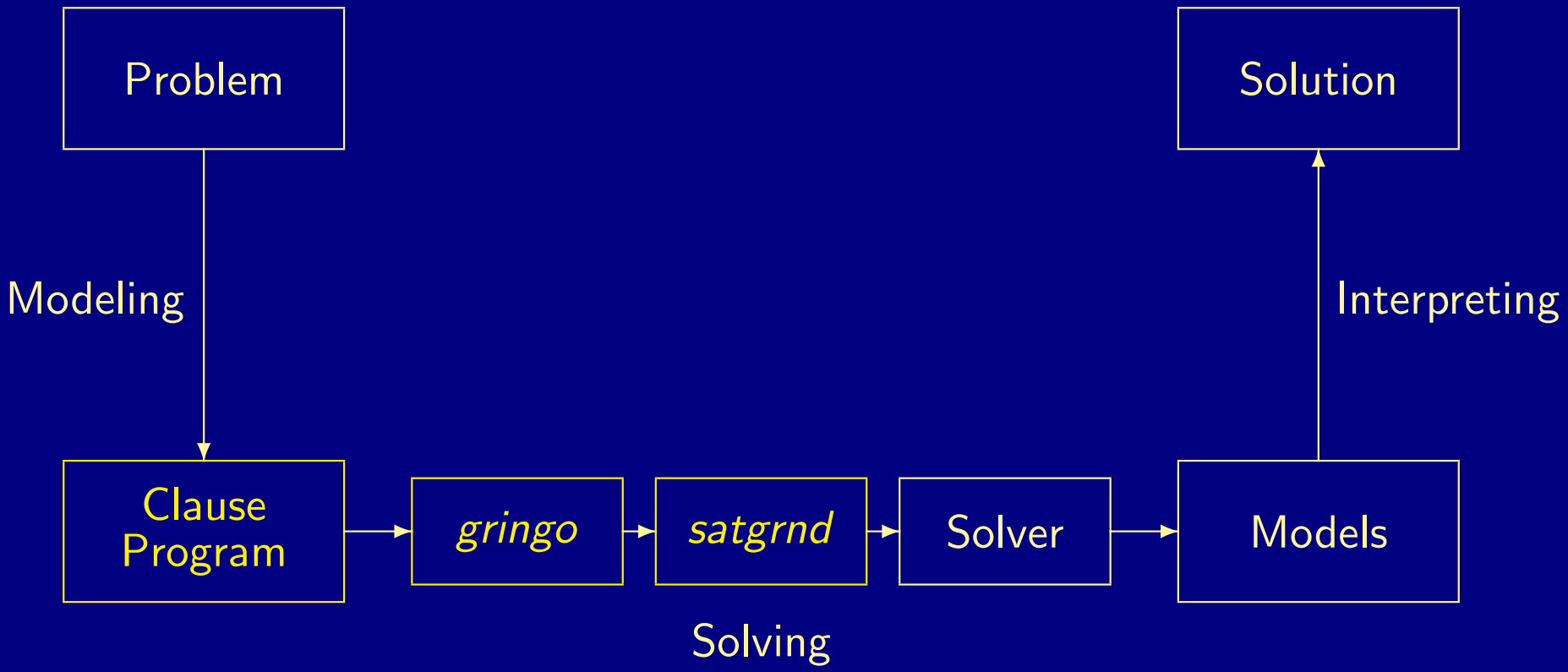
ASP Grounding for SAT



- satgrnd is available at http://research.ics.aalto.fi/software/sat/satgrnd

Potassco

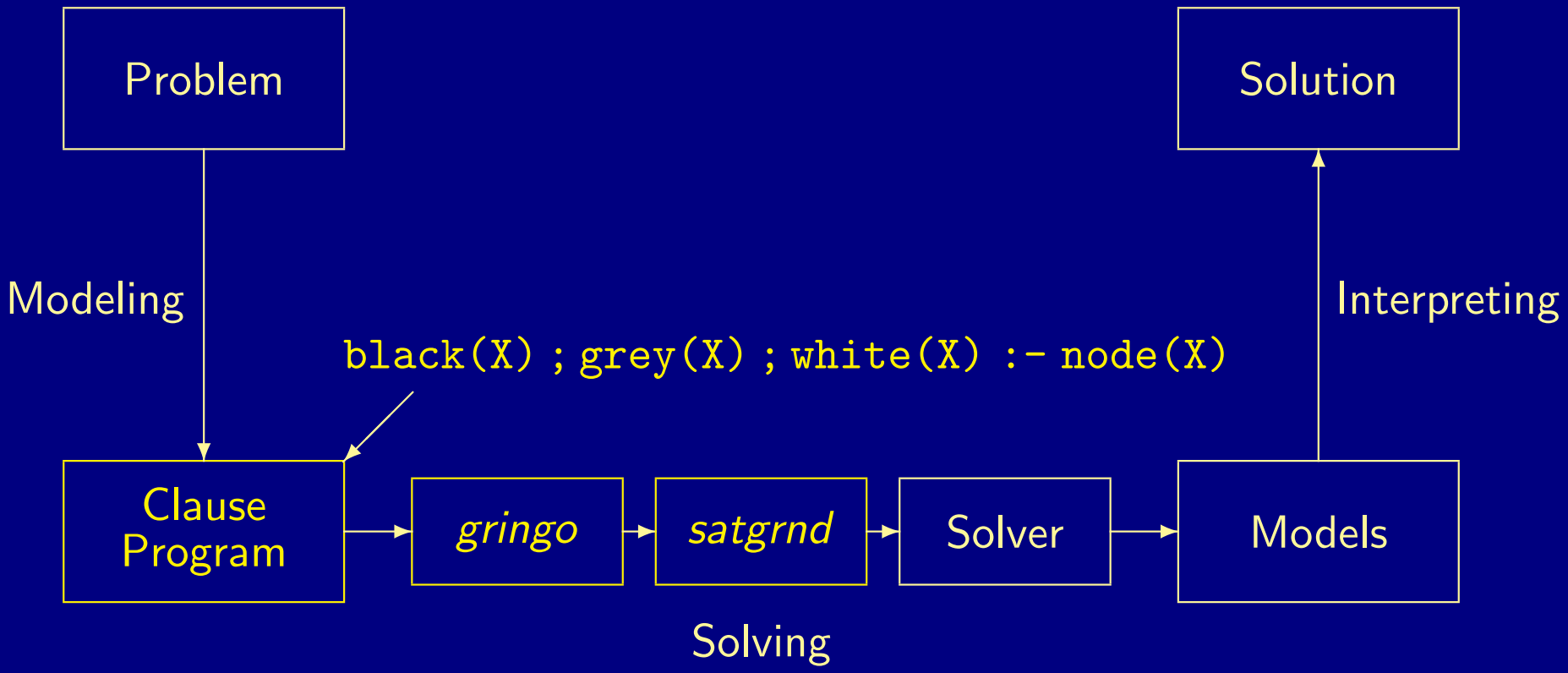
ASP Grounding for SAT



- satgrnd is available at http://research.ics.aalto.fi/software/sat/satgrnd

Potassco

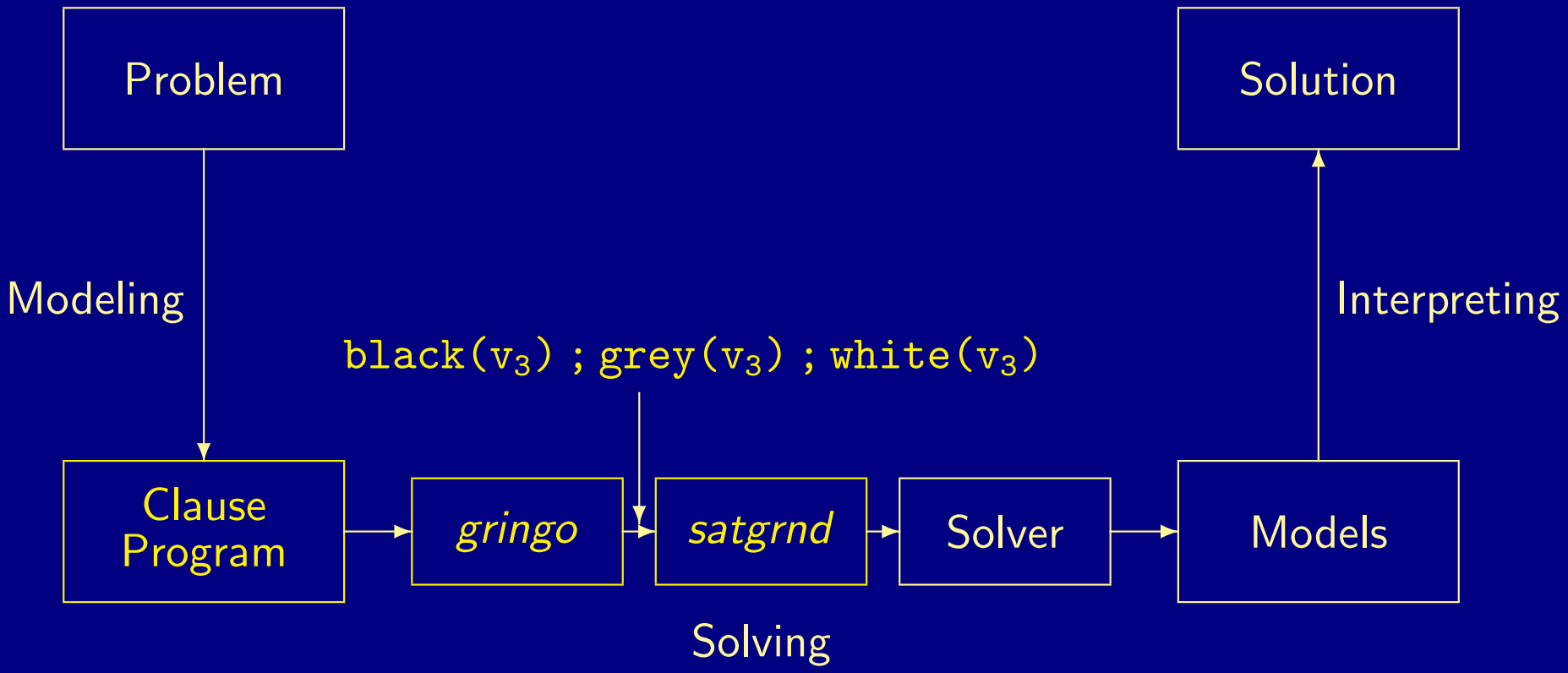
ASP Grounding for SAT



- satgrnd is available at http://research.ics.aalto.fi/software/sat/satgrnd

Potassco

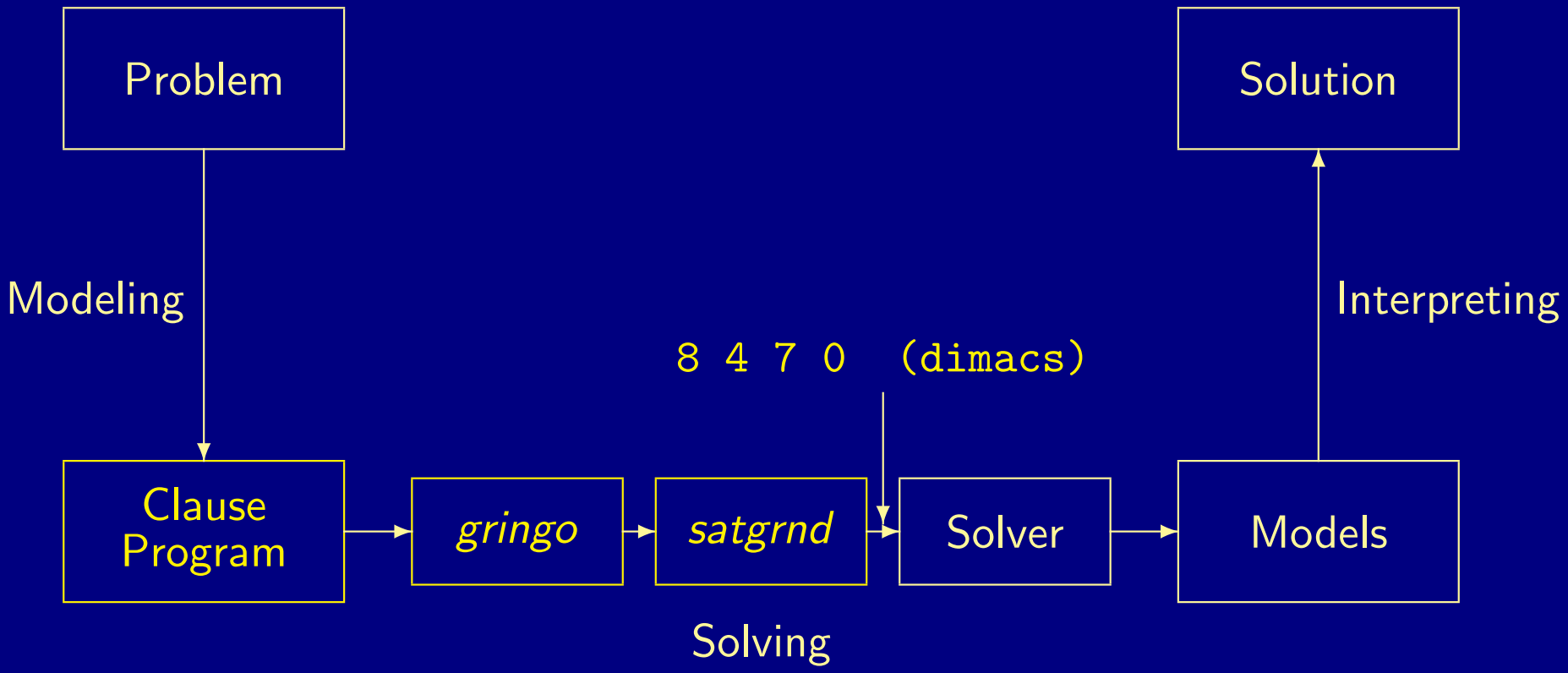
ASP Grounding for SAT



- satgrnd is available at http://research.ics.aalto.fi/software/sat/satgrnd

Potassco

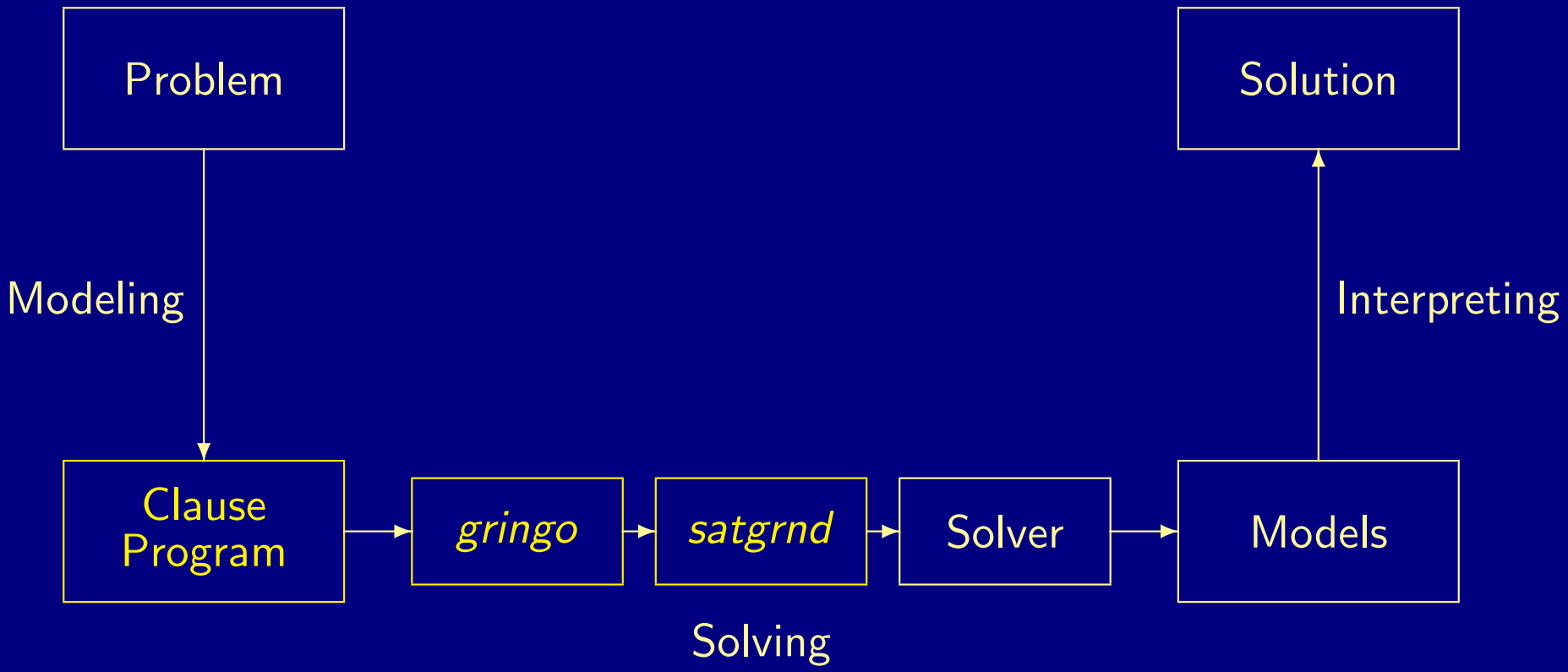
ASP Grounding for SAT



- satgrnd is available at http://research.ics.aalto.fi/software/sat/satgrnd

Potassco

ASP Grounding for SAT



- `satgrnd` is available at `http://research.ics.aalto.fi/software/sat/satgrnd`

Potassco

Outline

Potassco

The solver *clasp*

- Beyond deciding (stable) model existence, *clasp* allows for
  - Enumeration (without solution recording)
  - Projective enumeration (without solution recording)
  - Intersection and Union (linear solving process)
  - Multi-objective Optimization
  - and combinations thereof
- *clasp* allows for
  - ASP solving (*aspif* and *smodels* format)
  - SAT and MaxSAT solving (extended *dimacs* format)
  - PB solving (*opb* and *wbo* format)
- *clasp* pursues a coarse-grained, task-parallel approach to parallel search via shared memory multi-threading

Potassco

The solver *clasp*

- Beyond deciding (stable) model existence, *clasp* allows for
  - Enumeration (without solution recording)
  - Projective enumeration (without solution recording)
  - Intersection and Union (linear solving process)
  - Multi-objective Optimization
  - and combinations thereof
- *clasp* allows for
  - ASP solving (*aspif* and *smodels* format)
  - SAT and MaxSAT solving (extended *dimacs* format)
  - PB solving (*opb* and *wbo* format)
- *clasp* pursues a coarse-grained, task-parallel approach to parallel search via shared memory multi-threading

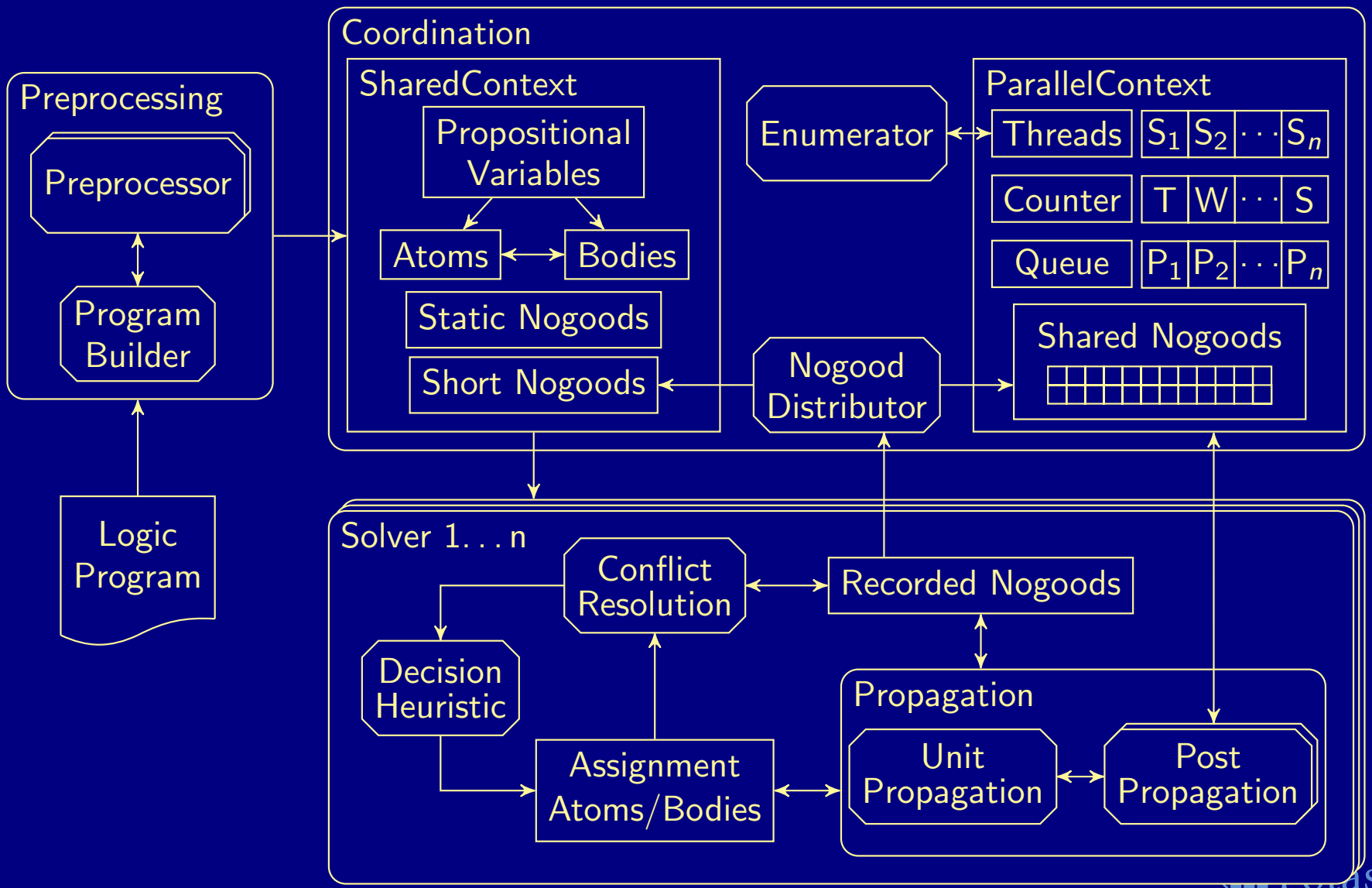
Potassco

The solver *clasp*

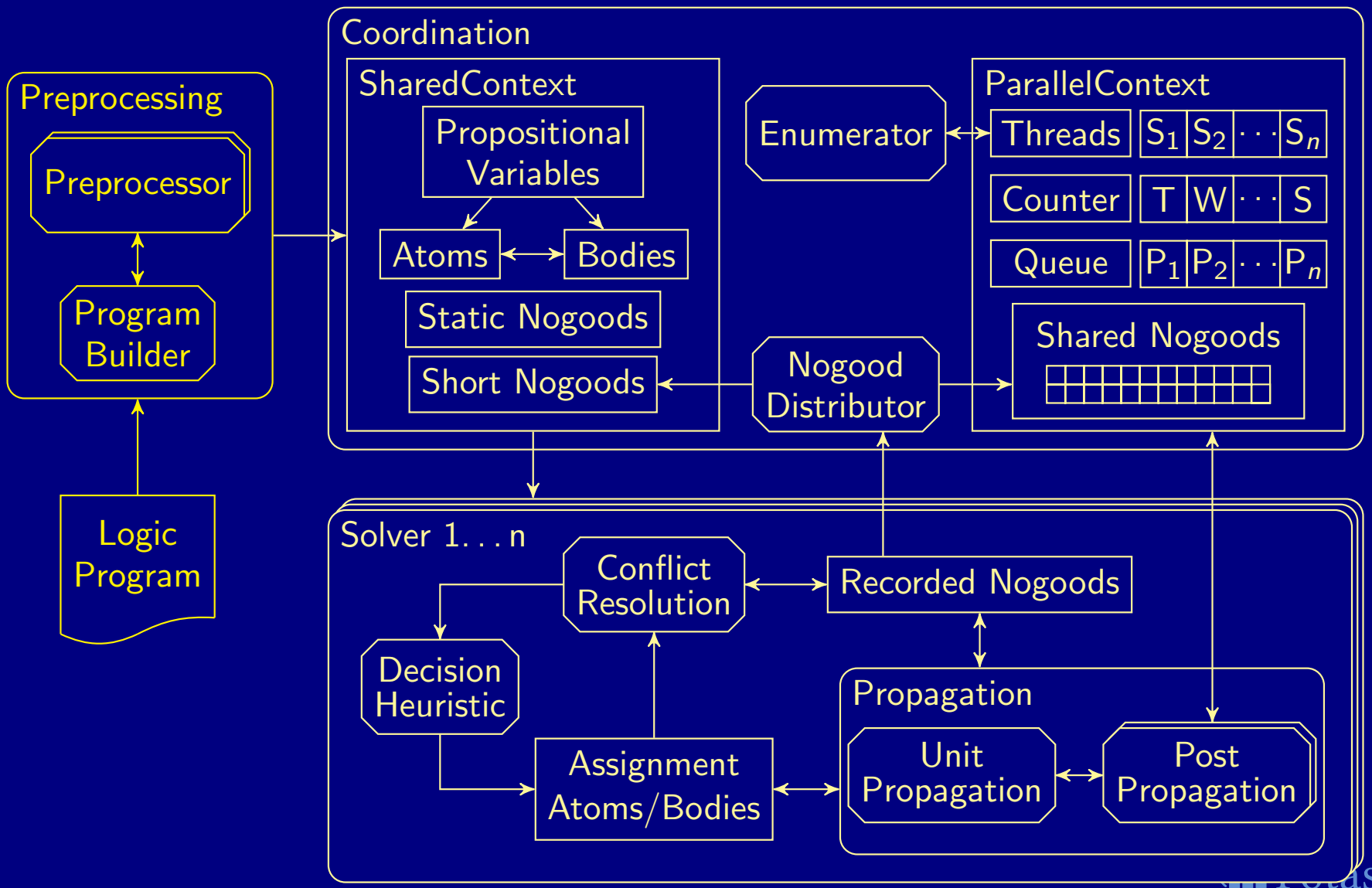
- Beyond deciding (stable) model existence, *clasp* allows for
  - Enumeration (without solution recording)
  - Projective enumeration (without solution recording)
  - Intersection and Union (linear solving process)
  - Multi-objective Optimization
  - and combinations thereof
- *clasp* allows for
  - ASP solving (*aspif* and *smodels* format)
  - SAT and MaxSAT solving (extended *dimacs* format)
  - PB solving (*opb* and *wbo* format)
- *clasp* pursues a coarse-grained, task-parallel approach to parallel search via shared memory multi-threading

Potassco

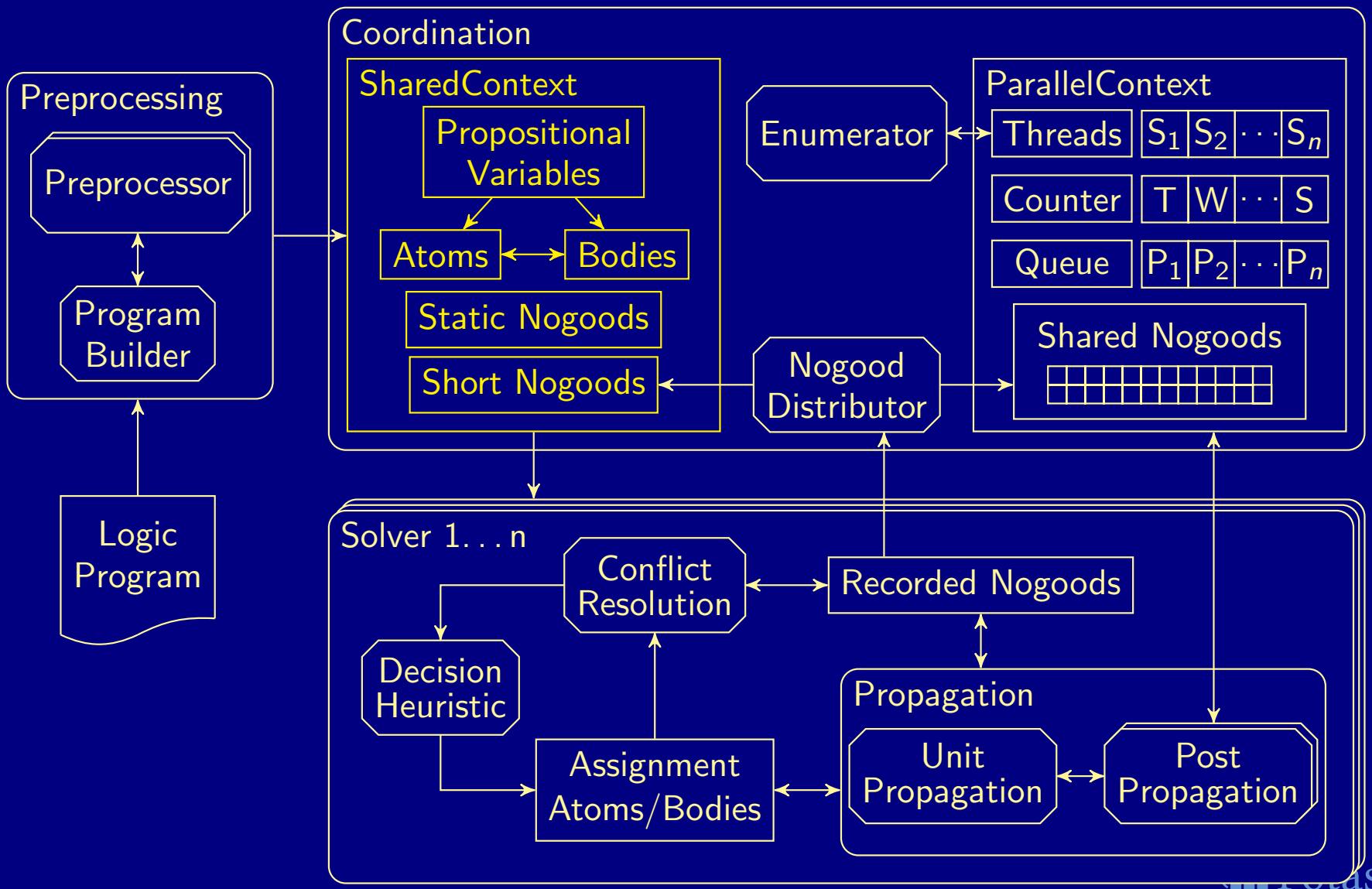
Multi-threaded architecture of *clasp*



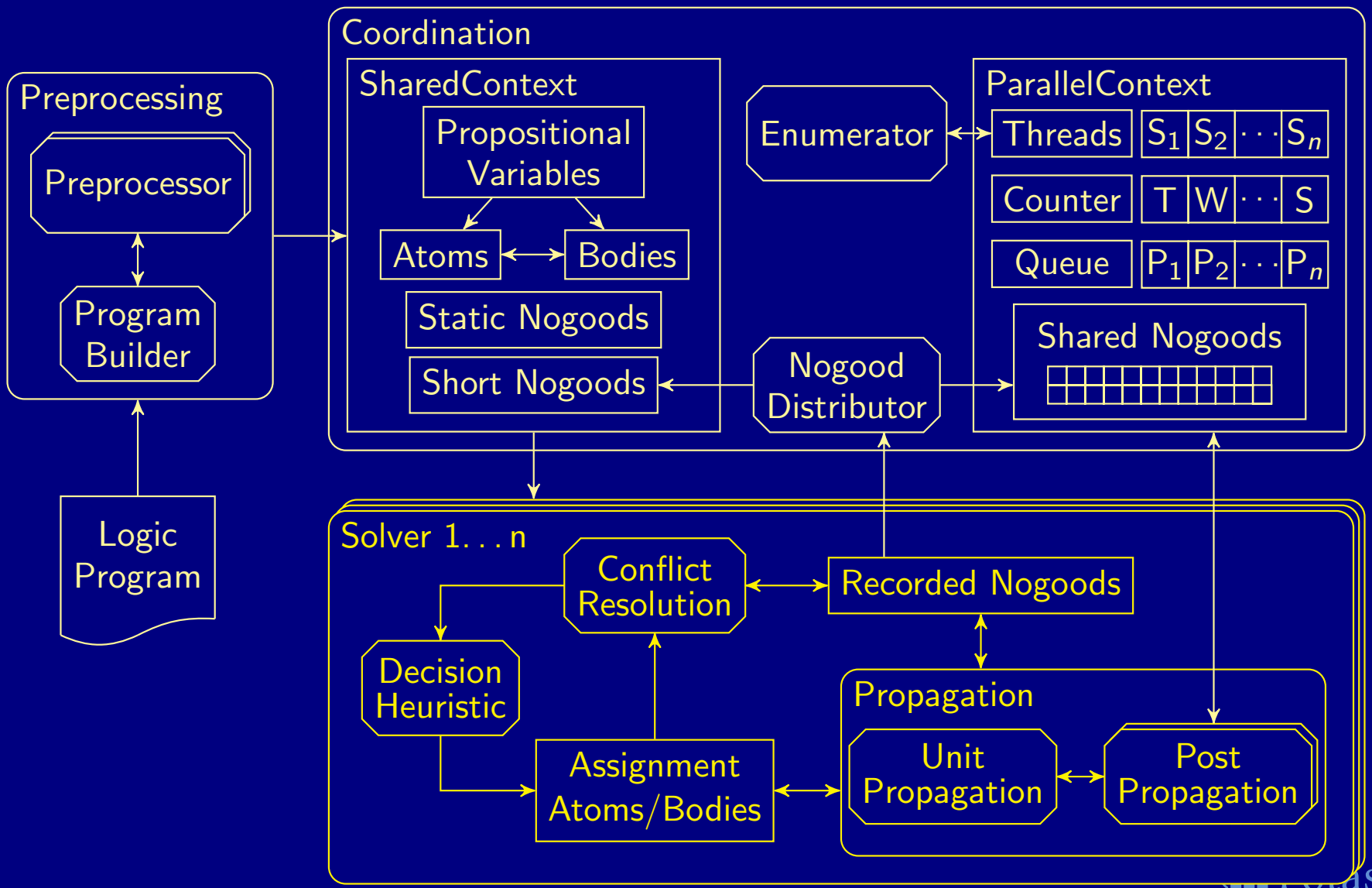
Multi-threaded architecture of *clasp*



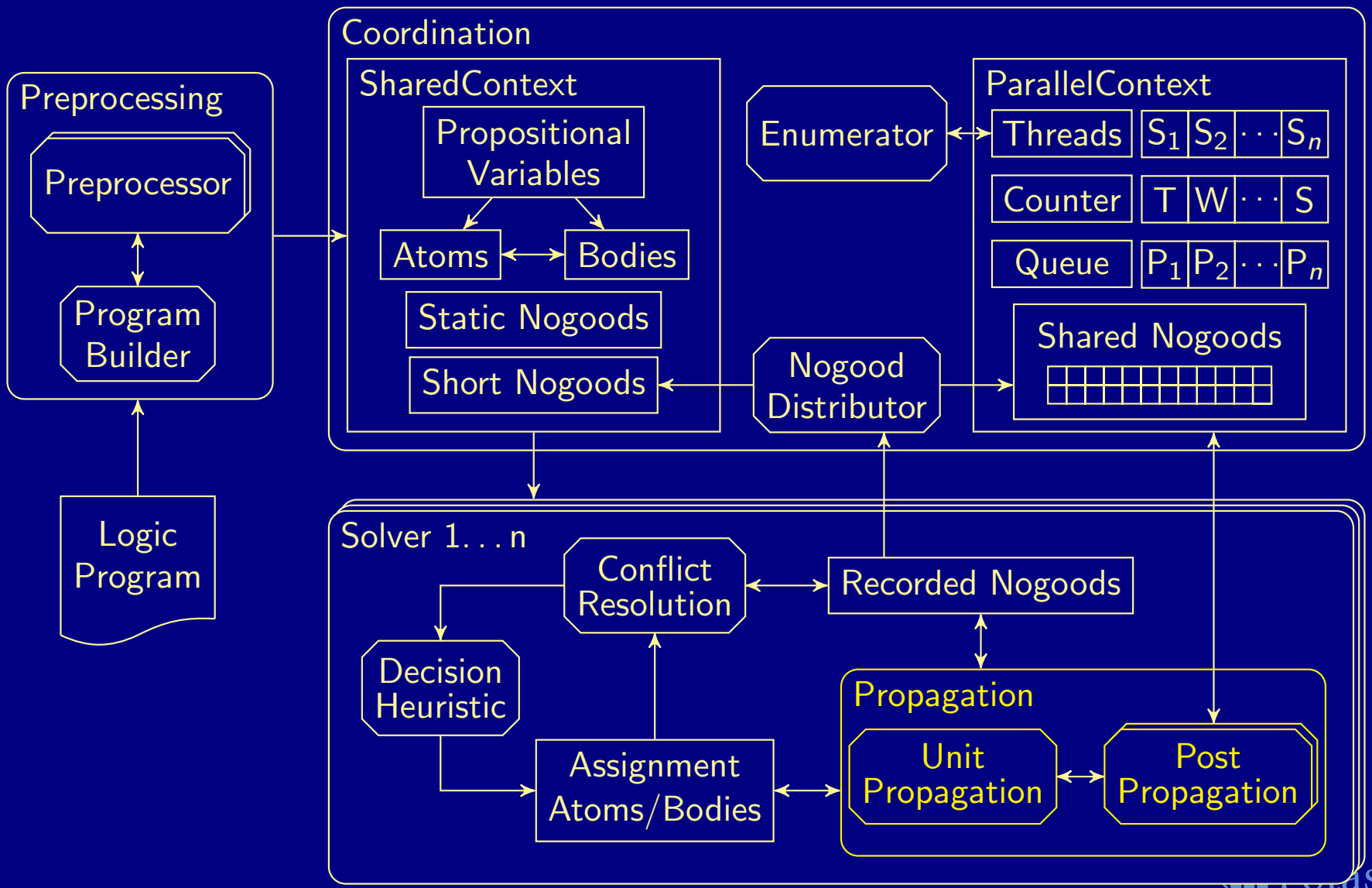
Multi-threaded architecture of *clasp*



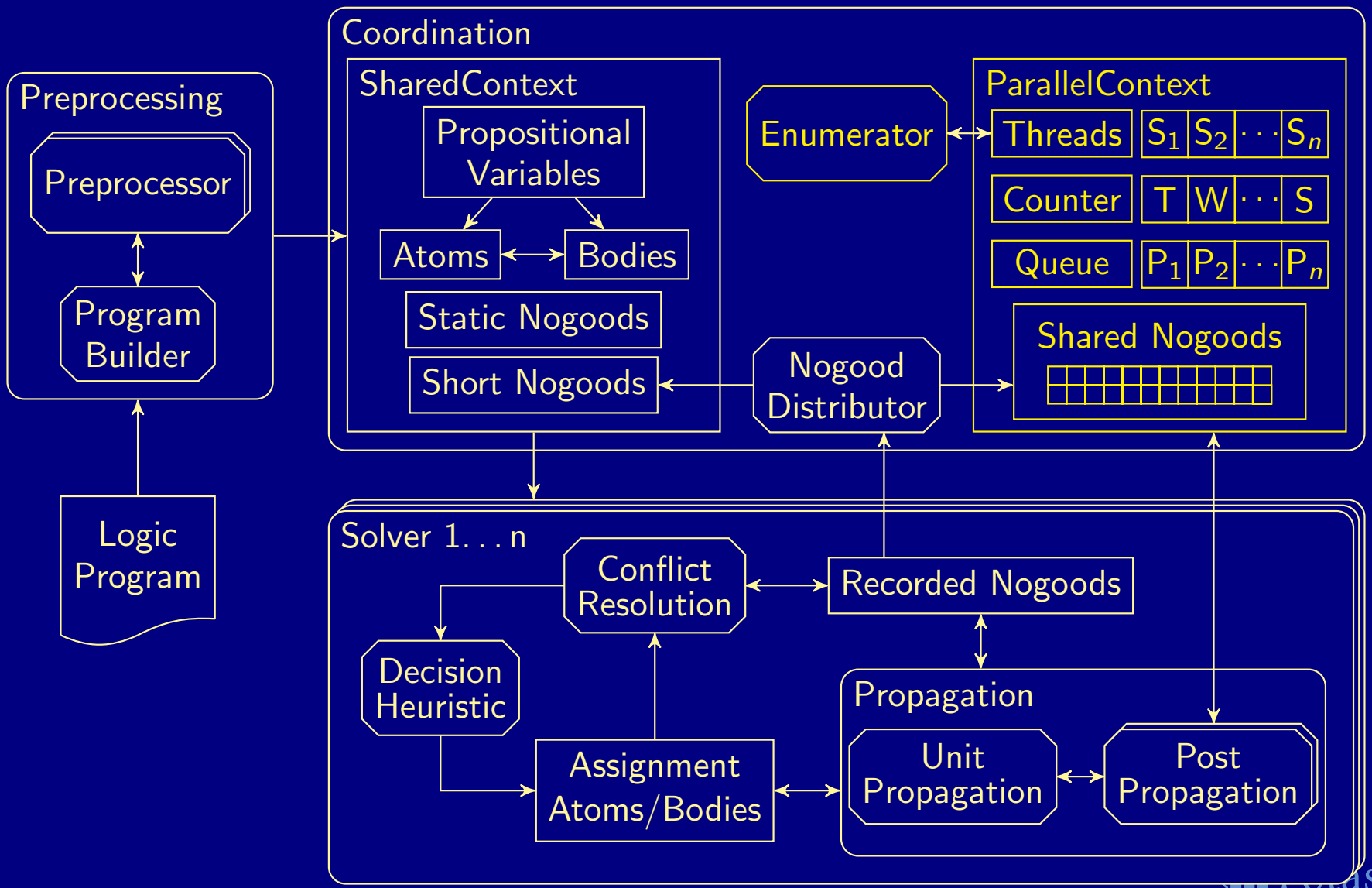
Multi-threaded architecture of *clasp*



Multi-threaded architecture of *clasp*



Multi-threaded architecture of *clasp*



Outline

Potassco

Outline

Potassco

Heuristic programming

- Heuristic directives

  $$\texttt{\#heuristic}\ a\ :\ l_1, \ldots, l_n.\ [k@p, m]$$

  where
  - a is an atom, and $l_1, \ldots, l_n$ are literals
  - k and p are integers
  - m is a heuristic modifier among `init`, `factor`, `level`, `sign`, `true`, or `false`
- Implementation bias on the `vsids` heuristic in *clasp*
- Examples

  Boosting convergence when optimizing

  ```
  #heuristic cycle(X,Y) : edge(X,Y). [1,false]
  ```

  Backward search when planning

  ```
  #heuristic occurs(A,T) : action(A), time(T). [T,factor]
  ```

Potassco

Heuristic programming

- Heuristic directives

  $$\texttt{\#heuristic}\ a\ :\ l_1, \ldots, l_n.\ [k@p, m]$$

  where
  - a is an atom, and $l_1, \ldots, l_n$ are literals
  - k and p are integers
  - m is a heuristic modifier among `init`, `factor`, `level`, `sign`, `true`, or `false`
- Implementation bias on the `vsids` heuristic in *clasp*
- Examples

  Boosting convergence when optimizing

  ```
  #heuristic cycle(X,Y) : edge(X,Y). [1,false]
  ```

  Backward search when planning

  ```
  #heuristic occurs(A,T) : action(A),time(T). [T,factor]
  ```

Potassco

Heuristic programming

- Heuristic directives

  $$\texttt{\#heuristic}\ a\ :\ l_1, \ldots, l_n.\ [k@p, m]$$

  where
  - a is an atom, and $l_1, \ldots, l_n$ are literals
  - k and p are integers
  - m is a heuristic modifier among `init`, `factor`, `level`, `sign`, `true`, or `false`
- Implementation bias on the `vsids` heuristic in *clasp*
- Examples
  - Boosting convergence when optimizing

    ```
    #heuristic cycle(X,Y) : edge(X,Y). [1,false]
    ```

  - Backward search when planning

    ```
    #heuristic occurs(A,T) : action(A),time(T). [T,factor]
    ```

Potassco

Heuristic programming

- Heuristic directives

  $$\texttt{\#heuristic}\ a\ :\ l_1, \ldots, l_n.\ [k@p, m]$$

  where
  - a is an atom, and $l_1, \ldots, l_n$ are literals
  - k and p are integers
  - m is a heuristic modifier among `init`, `factor`, `level`, `sign`, `true`, or `false`
- Implementation bias on the `vsids` heuristic in *clasp*
- Examples
  - Boosting convergence when optimizing

    ```
    #heuristic cycle(X,Y) : edge(X,Y). [1,false]
    ```

  - Backward search when planning

    ```
    #heuristic occurs(A,T) : action(A),time(T). [T,factor]
    ```

Potassco

Heuristic programming

- Heuristic directives

  $$\texttt{\#heuristic}\ a\ :\ l_1, \ldots, l_n.\ [k@p, m]$$

  where
  - a is an atom, and $l_1, \ldots, l_n$ are literals
  - k and p are integers
  - m is a heuristic modifier among `init`, `factor`, `level`, `sign`, `true`, or `false`
- Implementation bias on the `vsids` heuristic in *clasp*
- Examples
  - Boosting convergence when optimizing

    $$\texttt{\#heuristic cycle(X, Y) : edge(X, Y). [1, false]}$$

  - Backward search when planning

    $$\texttt{\#heuristic occurs(A, T) : action(A), time(T). [T, factor]}$$

Potassco

Outline

Potassco

Motivation

- Multi-shot solving
  is about solving continuously changing logic programs
  in an operative way

- Application areas

  Agents, Assisted Living, Robotics, Planning, Query-answering, etc

- Why

  ASP is non-monotonic
  ASP involves both grounding and solving

- How

  clingo 4 (*clingo* = *gringo* + *clasp*)

Potassco

Motivation

- Multi-shot solving
  is about solving continuously changing logic programs
  in an operative way
- Application areas
  Agents, Assisted Living, Robotics, Planning, Query-answering, etc
- Caveats
  - ASP is non-monotonic
  - ASP involves both grounding and solving
- Implementation *clingo* 4 (NB *clingo* = *gringo* + *clasp*)

Potassco

Motivation

- Multi-shot solving
  is about solving continuously changing logic programs
  in an operative way
- Application areas
  Agents, Assisted Living, Robotics, Planning, Query-answering, etc
- Caveats
  - ASP is non-monotonic
  - ASP involves both grounding and solving
- Implementation *clingo* 4 (NB *clingo* = *gringo* + *clasp*)

Potassco

Motivation

- Multi-shot solving
  is about solving continuously changing logic programs
  in an operative way
- Application areas
  Agents, Assisted Living, Robotics, Planning, Query-answering, etc
- Caveats
  - ASP is non-monotonic
  - ASP involves both grounding and solving
- Implementation *clingo* 4 (NB *clingo* = *gringo* + *clasp*)

Potassco

Motivation

- Multi-shot solving
  is about solving continuously changing logic programs
  in an operative way
- Application areas
  Agents, Assisted Living, Robotics, Planning, Query-answering, etc
- Caveats
  - ASP is non-monotonic
  - ASP involves both grounding and solving
- Implementation *clingo* 4 (NB *clingo* = *gringo* + *clasp*)

Potassco

Clingo = ASP + Control

- ASP

```
#program <name> [ (<parameters>) ]
          #program play(t).
#external <atom> [ :  <body> ]
          #external mark(X,Y,P,t) : field(X,Y), player(P).
```

- Control

Lua (www.lua.org)

```
prg:solve(), prg:ground(parts), ...
```

Python (www.python.org)

```
prg.solve(), prg.ground(parts), ...
```

C and Prolog embeddings are available soon

- Integration

in ASP: embedded scripting language (#script)
in Lua/Python: library import (import gringo)

Potassco

Clingo = ASP + Control

- ASP
  - `#program <name> [ (<parameters>) ]`
    - Example `#program play(t).`
  - `#external <atom> [ : <body> ]`
    - Example `#external mark(X,Y,P,t) : field(X,Y), player(P).`
- Control
  - Lua (`www.lua.org`)
    `prg:solve(), prg:ground(parts), ...`
  - Python (`www.python.org`)
    `prg.solve(), prg.ground(parts), ...`
  - C and Prolog embeddings are available soon
- Integration
  - in ASP: embedded scripting language (`#script`)
  - in Lua/Python: library import (`import gringo`)

Potassco

Clingo = ASP + Control

- ASP
  - `#program <name> [ (<parameters>) ]`
    - Example `#program play(t).`
  - `#external <atom> [ : <body> ]`
    - Example `#external mark(X,Y,P,t) : field(X,Y), player(P).`
- Control
  - Lua (`www.lua.org`)
    `prg:solve(), prg:ground(parts), ...`
  - Python (`www.python.org`)
    `prg.solve(), prg.ground(parts), ...`
  - C and Prolog embeddings are available soon
- Integration
  - in ASP: embedded scripting language (`#script`)
  - in Lua/Python: library import (`import gringo`)

Potassco

Clingo = ASP + Control

- ASP
  - `#program <name> [ (<parameters>) ]`
    - Example `#program play(t).`
  - `#external <atom> [ : <body> ]`
    - Example `#external mark(X,Y,P,t) : field(X,Y), player(P).`
- Control
  - Lua (`www.lua.org`)
    - Example `prg:solve(), prg:ground(parts), ...`
  - Python (`www.python.org`)
    - Example `prg.solve(), prg.ground(parts), ...`
  - C and Prolog embeddings are available soon
- Integration

  in ASP: embedded scripting language (`#script`)
  in Lua/Python: library import (`import gringo`)

Potassco

Clingo = ASP + Control

- ASP
  - `#program <name> [ (<parameters>) ]`
    - Example `#program play(t).`
  - `#external <atom> [ : <body> ]`
    - Example `#external mark(X,Y,P,t) : field(X,Y), player(P).`
- Control
  - Lua (`www.lua.org`)
    - Example `prg:solve(), prg:ground(parts), ...`
  - Python (`www.python.org`)
    - Example `prg.solve(), prg.ground(parts), ...`
  - C and Prolog embeddings are available soon
- Integration

  in ASP: embedded scripting language (`#script`)
  in Lua/Python: library import (`import gringo`)

Potassco

Clingo = ASP + Control

- ASP
  - `#program <name> [ (<parameters>) ]`
    - Example `#program play(t).`
  - `#external <atom> [ : <body> ]`
    - Example `#external mark(X,Y,P,t) : field(X,Y), player(P).`
- Control
  - Lua (`www.lua.org`)
    - Example `prg:solve(), prg:ground(parts), ...`
  - Python (`www.python.org`)
    - Example `prg.solve(), prg.ground(parts), ...`
  - C and Prolog embeddings are available soon
- Integration

  in ASP: embedded scripting language (`#script`)
  in Lua/Python: library import (`import gringo`)

Potassco

Clingo = ASP + Control

- ASP
  - `#program <name> [ (<parameters>) ]`
    - Example `#program play(t).`
  - `#external <atom> [ : <body> ]`
    - Example `#external mark(X,Y,P,t) : field(X,Y), player(P).`
- Control
  - Lua (`www.lua.org`)
    - Example `prg:solve(), prg:ground(parts), ...`
  - Python (`www.python.org`)
    - Example `prg.solve(), prg.ground(parts), ...`
  - C and Prolog embeddings are available soon
- Integration
  - in ASP: embedded scripting language (`#script`)
  - in Lua/Python: library import (`import gringo`)

Potassco

Vanilla *Clingo*

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

Potassco

Vanilla *Clingo*

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

Vanilla *Clingo*

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

Incremental solving

```
#program base.

p(0).

#program step (t).

p(t) :- p(t-1).

#program check (t).
#external plug(t).

:- not p(42), plug(t).
```

Potassco

Incremental solving

```
#program base.

p(0).

#program step (t).

p(t) :- p(t-1).

#program check (t).
#external plug(t).

:- not p(42), plug(t).
```

Potassco

Incremental solving

```
#program base.

p(0).

#program step (t).

p(t) :- p(t-1).

#program check (t).
#external plug(t).

:- not p(42), plug(t).
```

Potassco

Incremental solving

```
#program base.

p(0).

#program step (t).

p(t) :- p(t-1).

#program check (t).
#external plug(t).

:- not p(42), plug(t).
```

Potassco

Controlling incremental solving

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

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

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

Potassco

Outline

Potassco

Motivation

- Preference handling
  involves the combination of qualitative and quantitative preferences
- ASP systems provide optimization statements representing (lexicographically ordered) objective functions using summation
- Goal a framework for handling preferences among the (stable) models of logic programs
  - capturing existing approaches and
  - allowing for an easy implementation of new ones
- Implementation *asprin*

Potassco

Motivation

- Preference handling
  involves the combination of qualitative and quantitative preferences
- ASP systems provide optimization statements representing (lexicographically ordered) objective functions using summation
- Goal a framework for handling preferences among the (stable) models of logic programs
  - capturing existing approaches and
  - allowing for an easy implementation of new ones
- Implementation *asprin*

Potassco

Motivation

- Preference handling
  involves the combination of qualitative and quantitative preferences
- ASP systems provide optimization statements representing (lexicographically ordered) objective functions using summation
- Goal a framework for handling preferences among the (stable) models of logic programs
  - capturing existing approaches and
  - allowing for an easy implementation of new ones
- Implementation *asprin*

Potassco

Example

Your vacation (logic) program ...

... talking about *sauna*, *dive*, *hike*, *bungee*, *hot*, etc

$\#preference(bucks, less(weight))\{40 : sauna, 70 : dive\}$

$\#preference(fun, superset)\{sauna, dive, hike, \neg bungee\}$

$\#preference(temps, aso)\{dive > sauna \,\|\, hot, sauna > dive \,\|\, \neg hot\}$

$\#preference(all, pareto)\{name(bucks), name(fun), name(temps)\}$

$\#optimize(all)$

Potassco

Example

Your vacation (logic) program ...

... talking about *sauna*, *dive*, *hike*, *bungee*, *hot*, etc

$$\#preference(bucks, less(weight))\{40 : sauna, 70 : dive\}$$
$$\#preference(fun, superset)\{sauna, dive, hike, \neg bungee\}$$
$$\#preference(temps, aso)\{dive > sauna \,\|\, hot, sauna > dive \,\|\, \neg hot\}$$
$$\#preference(all, pareto)\{name(bucks), name(fun), name(temps)\}$$
$$\#optimize(all)$$

Potassco

Example

Your vacation (logic) program ...

... talking about *sauna*, *dive*, *hike*, *bungee*, *hot*, etc

$\#preference(bucks, less(weight))\{40 : sauna, 70 : dive\}$

$\#preference(fun, superset)\{sauna, dive, hike, \neg bungee\}$

$\#preference(temps, aso)\{dive > sauna \,\|\, hot, sauna > dive \,\|\, \neg hot\}$

$\#preference(all, pareto)\{name(bucks), name(fun), name(temps)\}$

$\#optimize(all)$

Potassco

Example

Your vacation (logic) program ...

... talking about *sauna*, *dive*, *hike*, *bungee*, *hot*, etc

$\#preference(bucks, less(weight))\{40 : sauna, 70 : dive\}$

$\#preference(fun, superset)\{sauna, dive, hike, \neg bungee\}$

$\#preference(temps, aso)\{dive > sauna \,\|\, hot, sauna > dive \,\|\, \neg hot\}$

$\#preference(all, pareto)\{name(bucks), name(fun), name(temps)\}$

$\#optimize(all)$

Potassco

Example

Your vacation (logic) program ...

... talking about *sauna*, *dive*, *hike*, *bungee*, *hot*, etc

$\#preference(bucks, less(weight))\{40 : sauna, 70 : dive\}$

$\#preference(fun, superset)\{sauna, dive, hike, \neg bungee\}$

$\#preference(temps, aso)\{dive > sauna \,\|\, hot, sauna > dive \,\|\, \neg hot\}$

$\#preference(all, pareto)\{name(bucks), name(fun), name(temps)\}$

$\#optimize(all)$

Potassco

Example

Your vacation (logic) program ...

... talking about *sauna*, *dive*, *hike*, *bungee*, *hot*, etc

$\#preference(bucks, less(weight))\{40 : sauna, 70 : dive\}$

$\#preference(fun, superset)\{sauna, dive, hike, \neg bungee\}$

$\#preference(temps, aso)\{dive > sauna \,\|\, hot, sauna > dive \,\|\, \neg hot\}$

$\#preference(all, pareto)\{name(bucks), name(fun), name(temps)\}$

$\#optimize(all)$

Potassco

Example

Your vacation (logic) program ...

... talking about *sauna*, *dive*, *hike*, *bungee*, *hot*, etc

$\#preference(bucks, less(weight))\{40 : sauna, 70 : dive\}$

$\#preference(fun, superset)\{sauna, dive, hike, \neg bungee\}$

$\#preference(temps, aso)\{dive > sauna \,\|\, hot, sauna > dive \,\|\, \neg hot\}$

$\#preference(all, pareto)\{name(bucks), name(fun), name(temps)\}$

$\#optimize(all)$

Potassco

Outline

Potassco

Motivation

- Confession ASP is not a silver bullet
- ASP modulo theories
  is about integrating dedicated reasoning procedure
- Application areas
  Agents, Assisted Living, Robotics, Planning, Scheduling, Bio- and Cheminformatics
- Caveats
  - ASP is non-monotonic
  - ASP involves both grounding and solving
- Implementation *clingo* 5 (and *clingcon* for CSP)

Potassco

Motivation

- Confession ASP is not a silver bullet
- ASP modulo theories
  is about integrating dedicated reasoning procedure
- Application areas
  Agents, Assisted Living, Robotics, Planning, Scheduling,
  Bio- and Cheminformatics
- Caveats
  - ASP is non-monotonic
  - ASP involves both grounding and solving
- Implementation *clingo* 5 (and *clingcon* for CSP)

Potassco

Motivation

- Confession ASP is not a silver bullet
- ASP modulo theories
  is about integrating dedicated reasoning procedure
- Application areas
  Agents, Assisted Living, Robotics, Planning, Scheduling,
  Bio- and Cheminformatics
- Caveats
  - ASP is non-monotonic
  - ASP involves both grounding and solving
- Implementation *clingo* 5 (and *clingcon* for CSP)

Potassco

Motivation

- Confession ASP is not a silver bullet
- ASP modulo theories
  is about integrating dedicated reasoning procedure
- Application areas
  Agents, Assisted Living, Robotics, Planning, Scheduling,
  Bio- and Cheminformatics
- Caveats
  - ASP is non-monotonic
  - ASP involves both grounding and solving
- Implementation *clingo* 5 (and *clingcon* for CSP)

Potassco

Motivation

- Confession ASP is not a silver bullet
- ASP modulo theories
  is about integrating dedicated reasoning procedure
- Application areas
  Agents, Assisted Living, Robotics, Planning, Scheduling,
  Bio- and Cheminformatics
- Caveats
  - ASP is non-monotonic
  - ASP involves both grounding and solving
- Implementation *clingo* 5 (and *clingcon* for CSP)

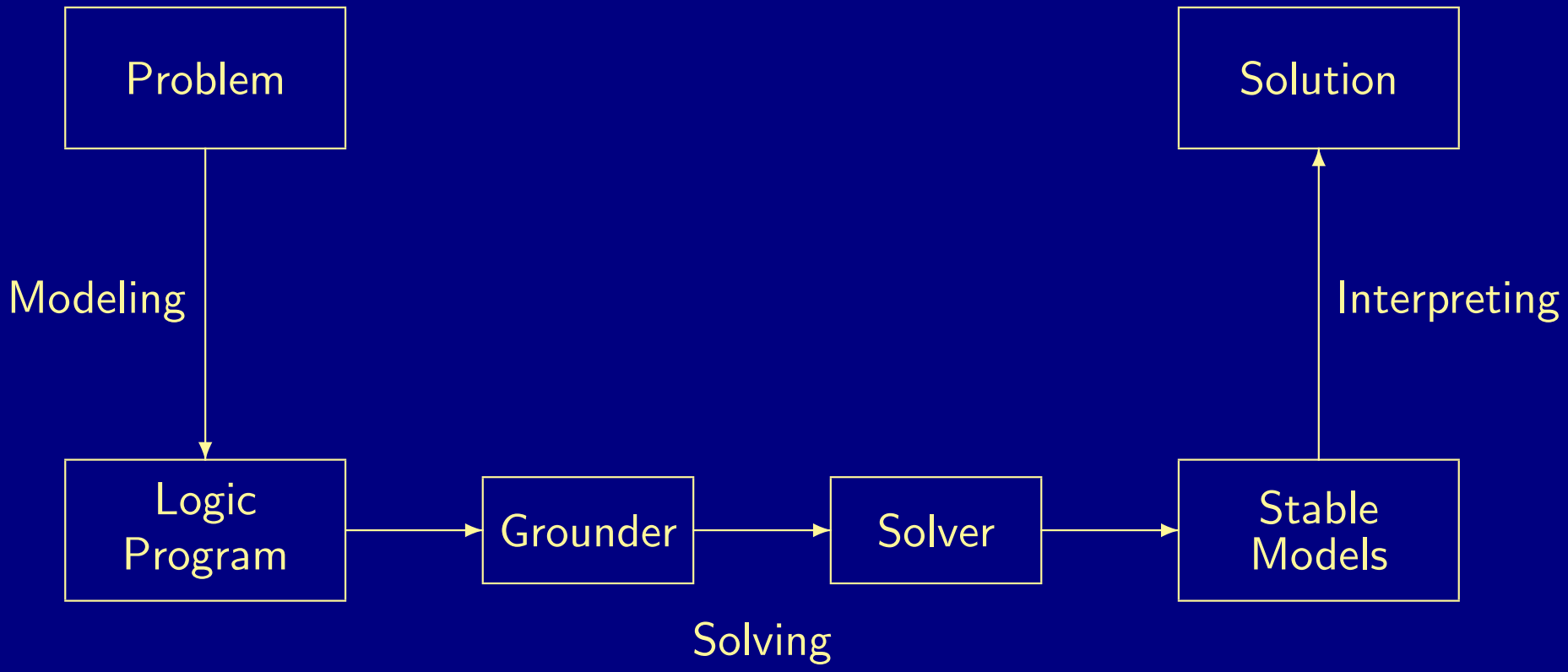
Potassco

Motivation

- Confession ASP is not a silver bullet
- ASP modulo theories
  is about integrating dedicated reasoning procedure
- Application areas
  Agents, Assisted Living, Robotics, Planning, Scheduling,
  Bio- and Cheminformatics
- Caveats
  - ASP is non-monotonic
  - ASP involves both grounding and solving
- Implementation *clingo* 5 (and *clingcon* for CSP)

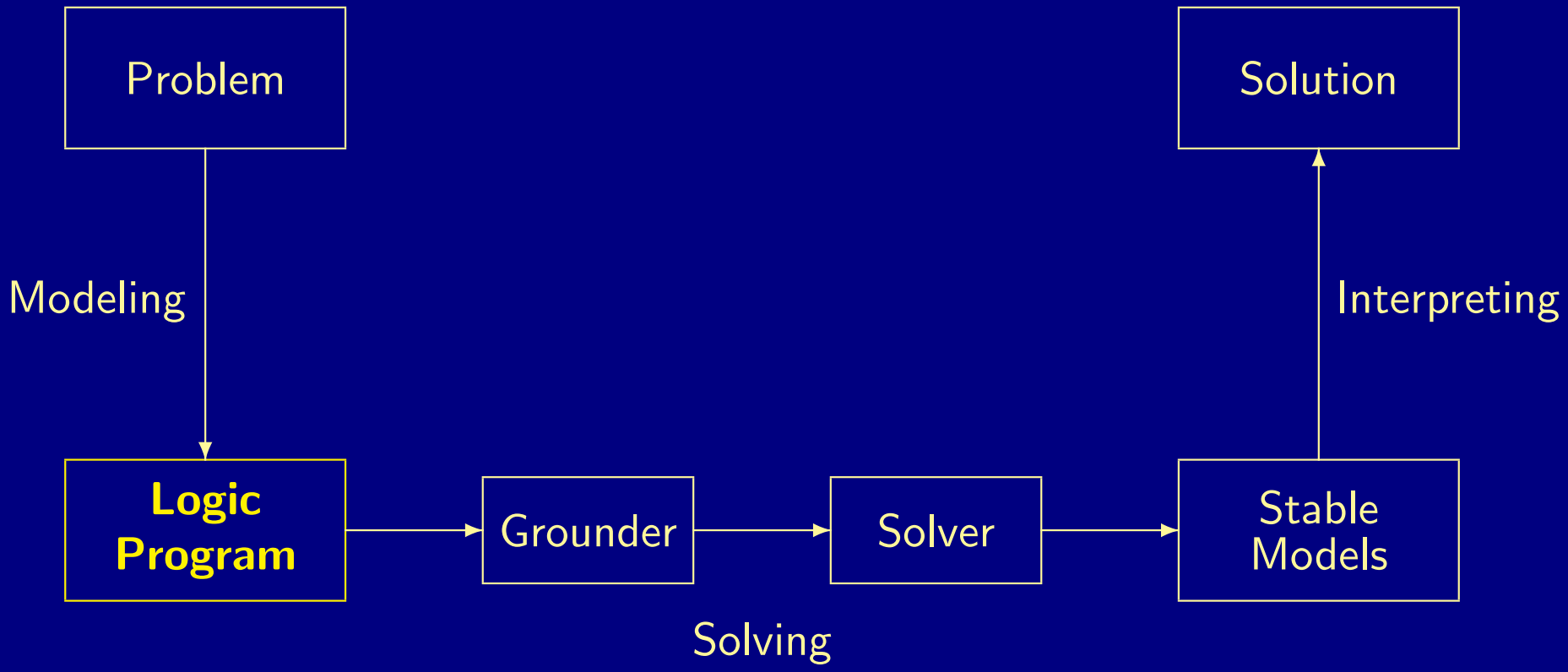
Potassco

ASP solving process modulo theories



Potassco

ASP solving process modulo theories



Potassco

Linear constraints

```
#theory csp {
    linear_term {
      + : 5, unary;
      - : 5, unary;
      * : 4, binary, left;
      + : 3, binary, left;
      - : 3, binary, left
    };

    dom_term {
      +  : 5, unary;
      -  : 5, unary;
      .. : 1, binary, left
    };

    &dom/0 : dom_term, {=}, linear_term, any;
    &sum/0 : linear_term, {<=,=,>=,<,>,!=}, linear_term, any;
    &show/0 : show_term, directive;
    &distinct/0 : linear_term, any;
    &minimize/0 : minimize_term, directive
}.
```

```
show_term {
  / : 1, binary, left
};

minimize_term {
  + : 5, unary;
  - : 5, unary;
  * : 4, binary, left;
  + : 3, binary, left;
  - : 3, binary, left;
  @ : 0, binary, left
};
```

Potassco

send+more=money

```
#include "csp.lp".

digit(1,3,s).   digit(2,3,m).   digit(sum,4,m).
digit(1,2,e).   digit(2,2,o).   digit(sum,3,o).
digit(1,1,n).   digit(2,1,r).   digit(sum,2,n).
digit(1,0,d).   digit(2,0,e).   digit(sum,1,e).
                                digit(sum,0,y).

base(10).
exp(E) :- digit(_,E,_).

power(1,0).
power(B*P,E) :- base(B), power(P,E-1), exp(E), E>0.

summand(N) :- digit(N,_,_), N!= sum.
high(D) :- digit(N,E,D), not digit(N,E+1,_).

&dom {0..9} = X :- digit(_,_,X).

&sum { M*D : digit(N,E,D),   power(M,E), summand(N);
      -M*D : digit(sum,E,D), power(M,E)       } = 0.

&sum { D } > 0 :- high(D).

&distinct { D : digit(_,_,D) }.

&show { D : digit(_,_,D) }.
```

Potassco

send+more=money

```
#include "csp.lp".

digit(1,3,s).   digit(2,3,m).   digit(sum,4,m).
digit(1,2,e).   digit(2,2,o).   digit(sum,3,o).
digit(1,1,n).   digit(2,1,r).   digit(sum,2,n).
digit(1,0,d).   digit(2,0,e).   digit(sum,1,e).
                                digit(sum,0,y).
base(10).
exp(E) :- digit(_,E,_).

power(1,0).
power(B*P,E) :- base(B), power(P,E-1), exp(E), E>0.

summand(N) :- digit(N,_,_), N!= sum.
high(D) :- digit(N,E,D), not digit(N,E+1,_).

&dom {0..9} = X :- digit(_,_,X).

&sum { M*D : digit(N,E,D),   power(M,E), summand(N);
      -M*D : digit(sum,E,D), power(M,E)       } = 0.

&sum { D } > 0 :- high(D).

&distinct { D : digit(_,_,D) }.

&show { D : digit(_,_,D) }.
```

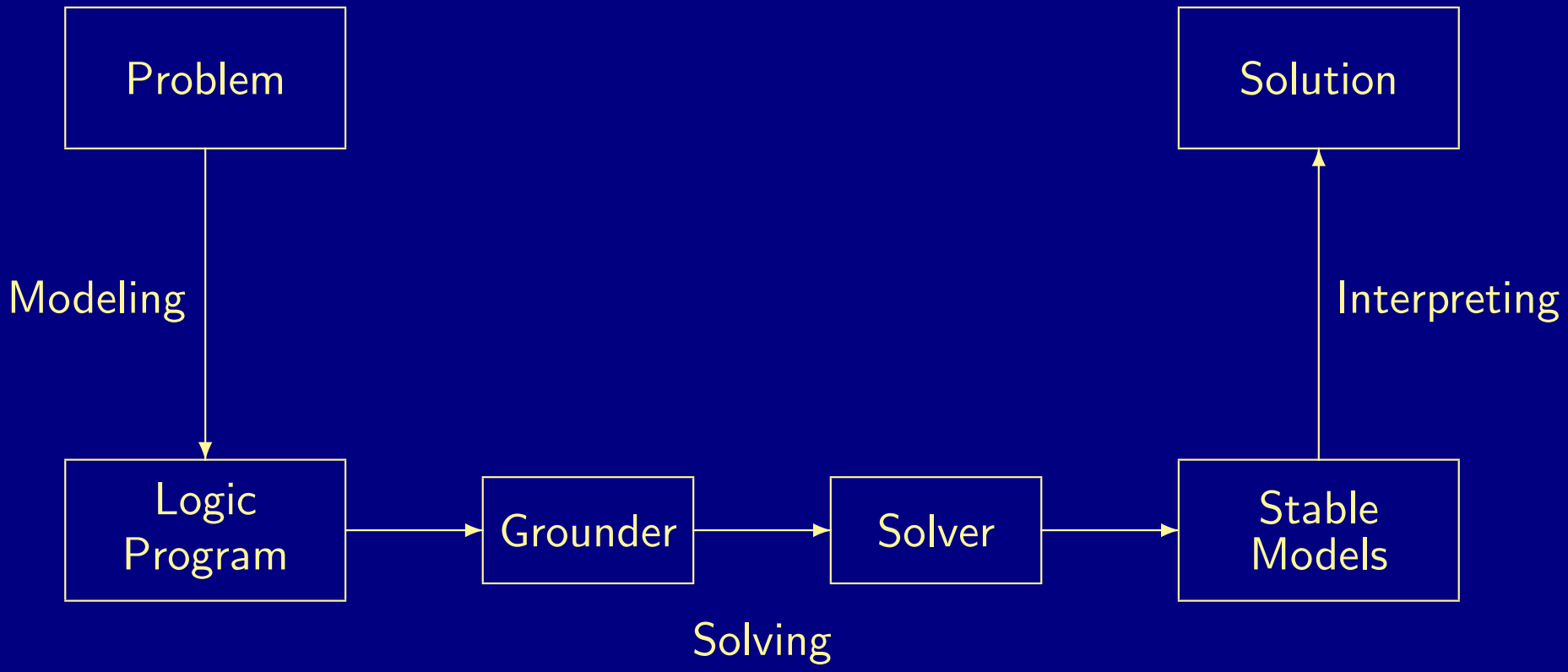
Potassco

send+more=money

```
#include "csp.lp".

digit(1,3,s).   digit(2,3,m).   digit(sum,4,m).
digit(1,2,e).   digit(2,2,o).   digit(sum,3,o).
digit(1,1,n).   digit(2,1,r).   digit(sum,2,n).
digit(1,0,d).   digit(2,0,e).   digit(sum,1,e).
                                digit(sum,0,y).
base(10).
exp(E) :- digit(_,E,_).

power(1,0).
power(B*P,E) :- base(B), power(P,E-1), exp(E), E>0.

summand(N) :- digit(N,_,_), N!= sum.
high(D) :- digit(N,E,D), not digit(N,E+1,_).

&dom {0..9} = X :- digit(_,_,X).

&sum { M*D : digit(N,E,D),   power(M,E), summand(N);
      -M*D : digit(sum,E,D), power(M,E)       } = 0.

&sum { D } > 0 :- high(D).

&distinct { D : digit(_,_,D) }.

&show { D : digit(_,_,D) }.
```

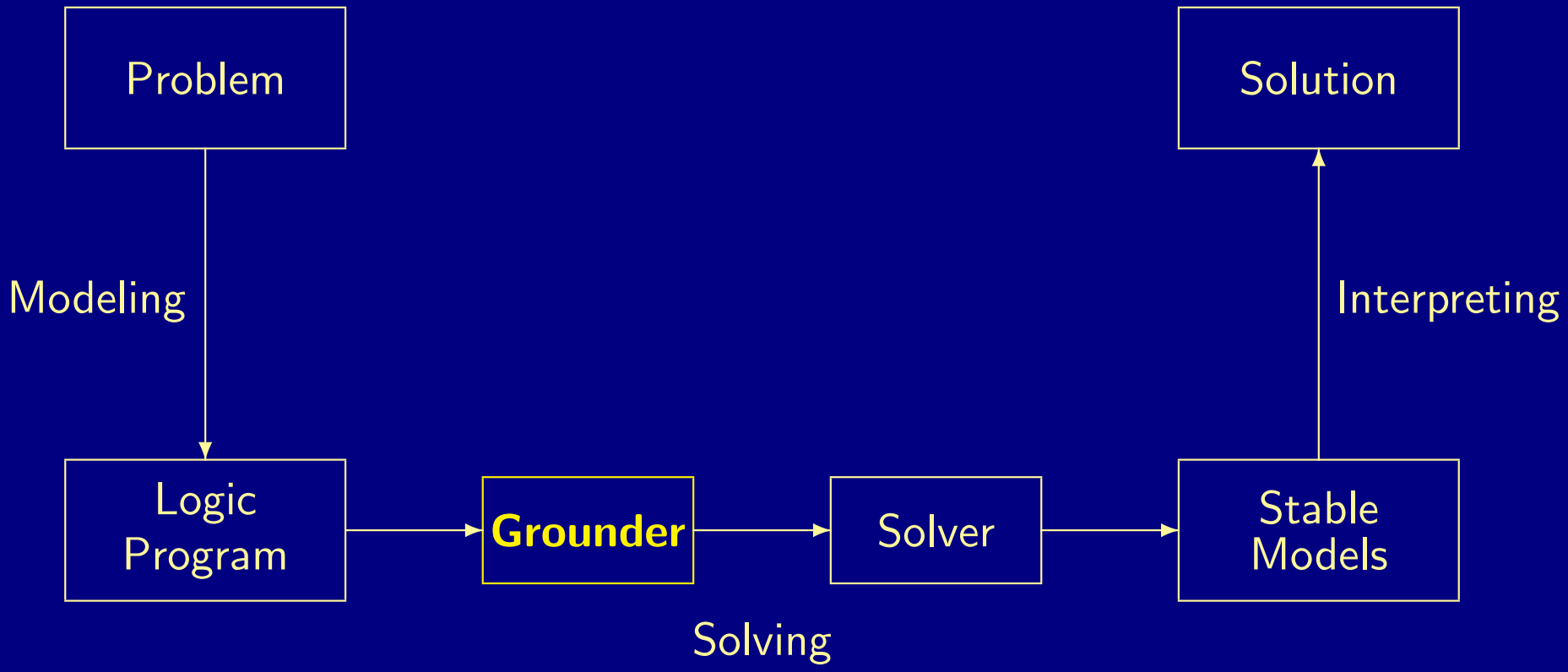
Potassco

ASP solving process modulo theories



Potassco

ASP solving process modulo theories



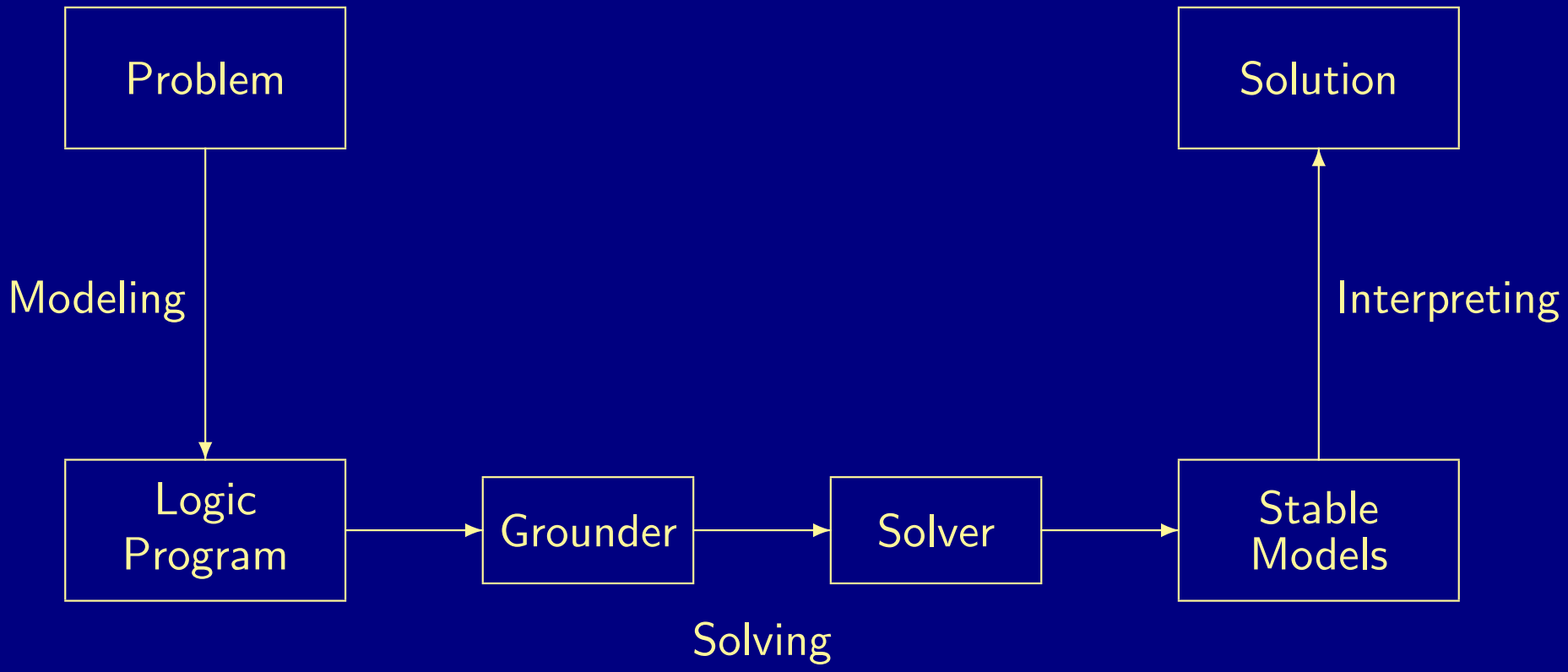
Potassco

send+more=money

```
digit(1,3,s).   digit(2,3,m).   digit(sum,4,m).
digit(1,2,e).   digit(2,2,o).   digit(sum,3,o).
digit(1,1,n).   digit(2,1,r).   digit(sum,2,n).
digit(1,0,d).   digit(2,0,e).   digit(sum,1,e).
                                digit(sum,0,y).

base(10).
exp(0).  exp(1).  exp(2).  exp(3).  exp(4).

power(1,0).
power(10,1).  power(100,2).  power(1000,3).  power(10000,4).

summand(1).  summand(2).
high(s).  high(m).

&dom{0..9}=s. &dom{0..9}=m. &dom{0..9}=e. &dom{0..9}=o. &dom{0..9}=n. &dom{0..9}=r. &do

&sum{           1000*s;  100*e;  10*n;  1*d;
                1000*m;  100*o;  10*r;  1*e;
     -10000*m; -1000*o; -100*n; -10*e; -1*y  } = 0.

&sum{s} > 0.  &sum{m} > 0.

&distinct{s; m; e; o; n; r; d; y}.

&show{s; m; e; o; n; r; d; y}.
```

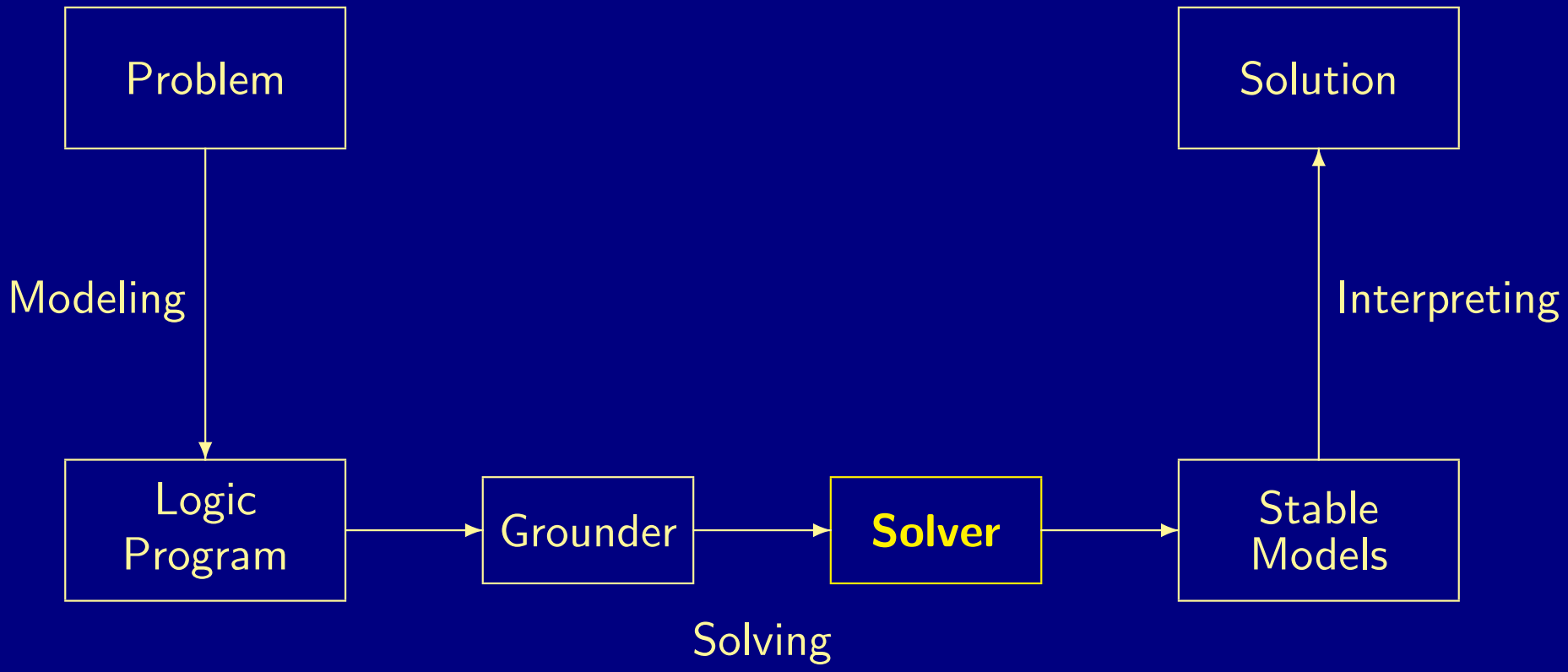
Potassco

ASP solving process modulo theories



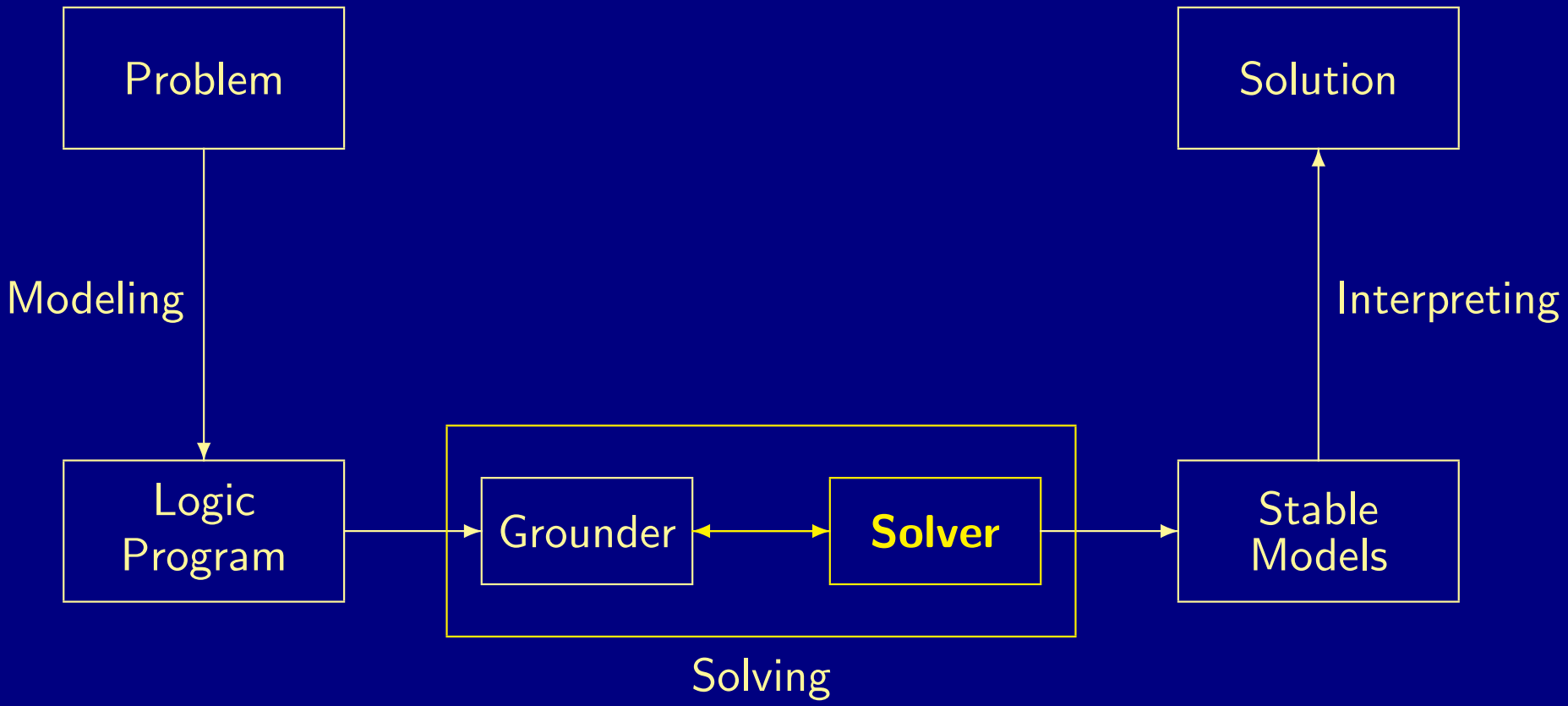
Potassco

ASP solving process modulo theories



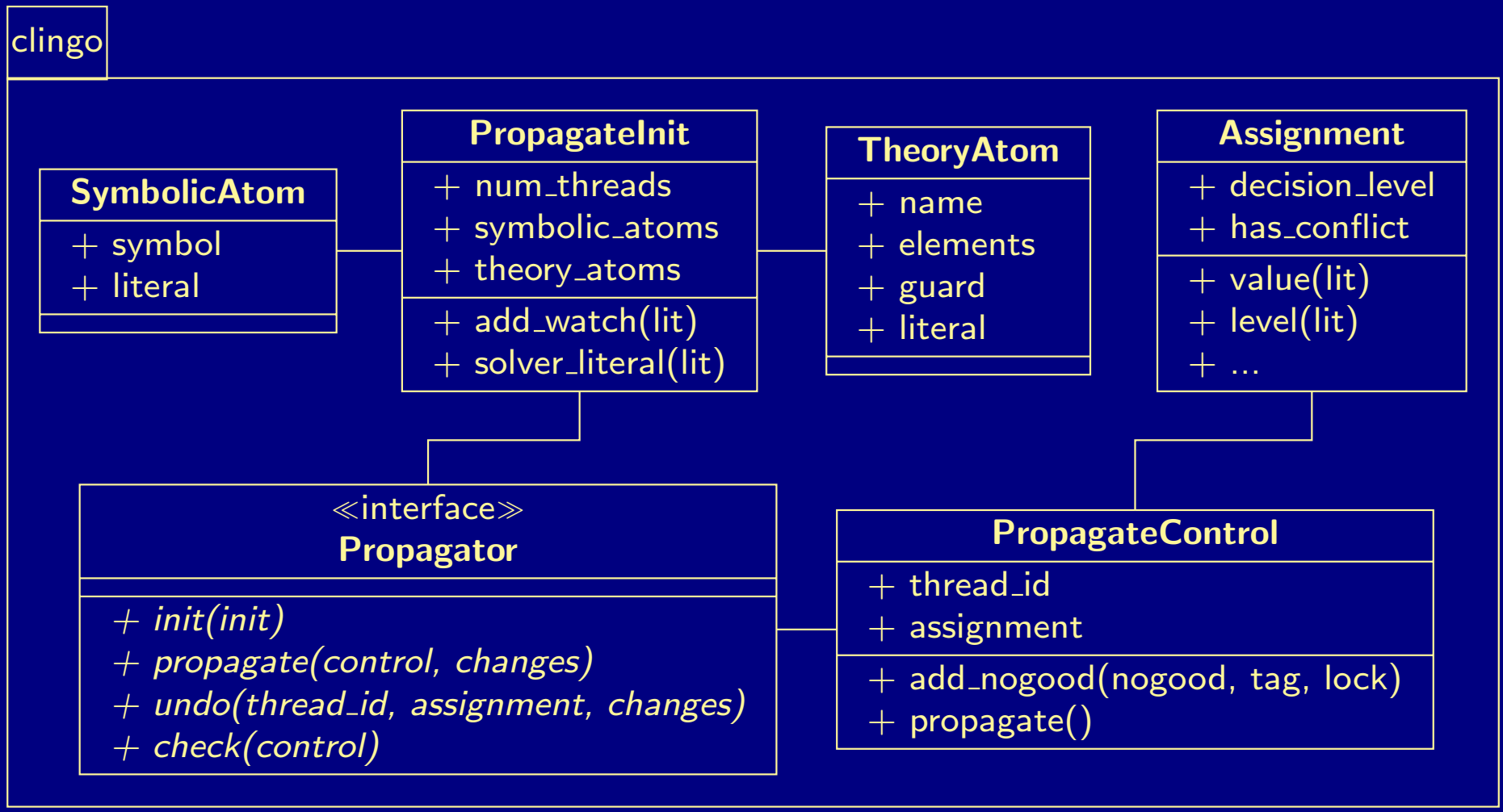
Potassco

ASP solving process modulo theories



Potassco

Propagator interface



Potassco

The *dot* propagator

```
#script (python)

import sys
import time

class Propagator:
    def init(self, init):
        self.sleep = .1
        for atom in init.symbolic_atoms:
            init.add_watch(init.solver_literal(atom.literal))

    def propagate(self, ctl, changes):
        for l in changes:
            sys.stdout.write(".")
            sys.stdout.flush()
            time.sleep(self.sleep)
        return True

    def undo(self, solver_id, assign, undo):
        for l in undo:
            sys.stdout.write("\b \b")
            sys.stdout.flush()
            time.sleep(self.sleep)

def main(prg):
    prg.register_propagator(Propagator())
    prg.ground([("base", [])])
    prg.solve()
    sys.stdout.write("\n")

#end.
```

Potassco

Outline

Potassco

Take home message

Potassco

Take home message

ASP = DB+LP+KR+SAT

Potassco

Take home message

$$\textbf{ASP} = \textbf{DB}+\textbf{LP}+\textbf{KR}+\textbf{SMT}^{n}$$

Potassco

Take home message

ASP = DB+LP+KR+SMT^n

http://potassco.sourceforge.net

Potassco

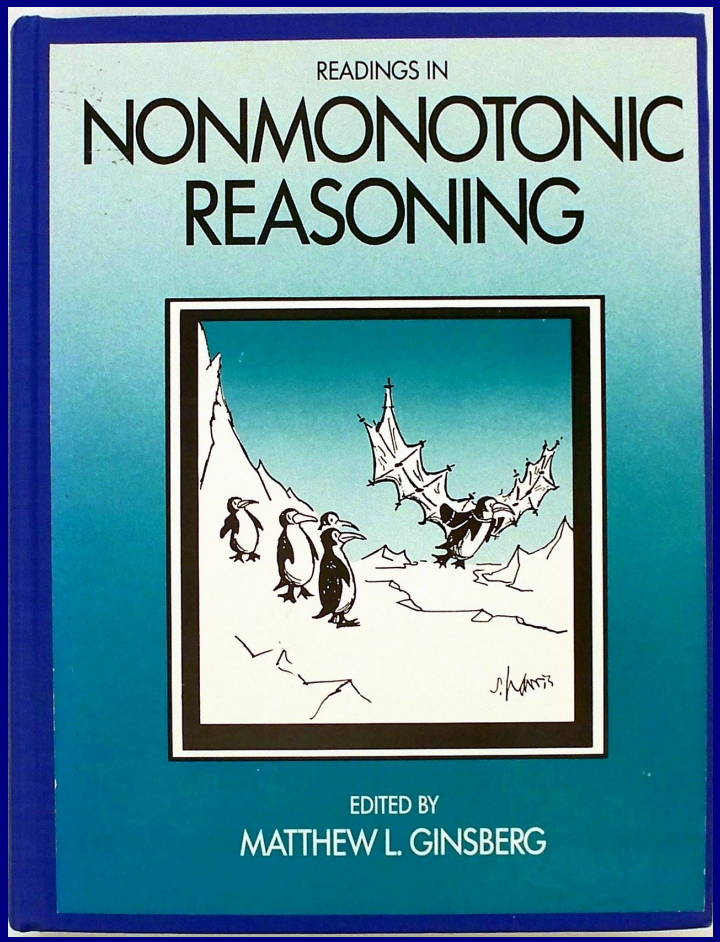
Epilogue

After all, it's all about Tweety!

Potassco

Epilogue

After all, it's all about Tweety!

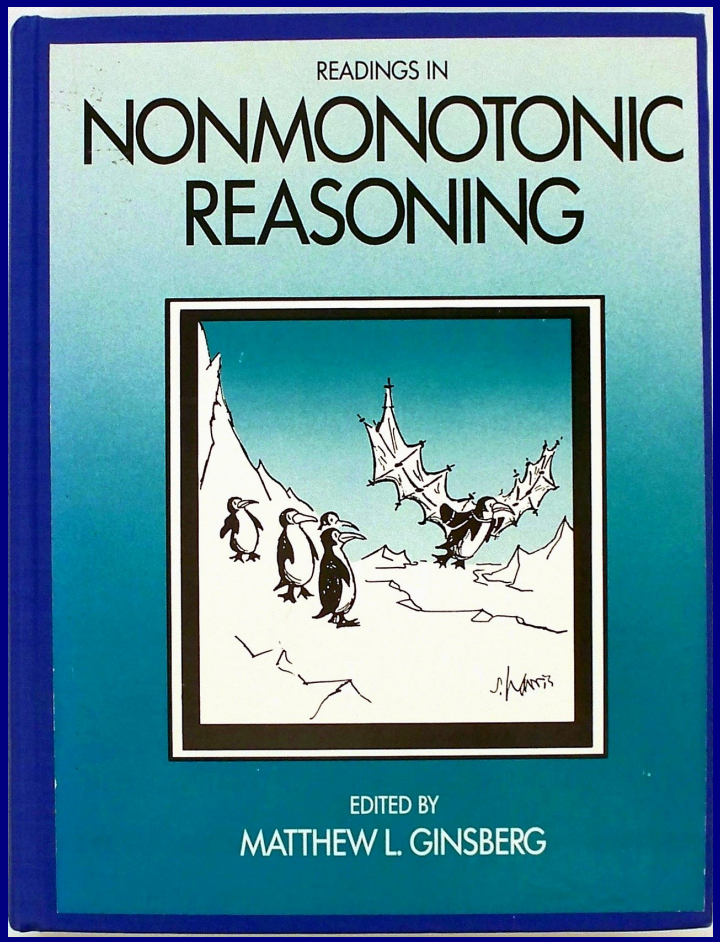


'87

Potassco

Epilogue

After all, it's all about Tweety!



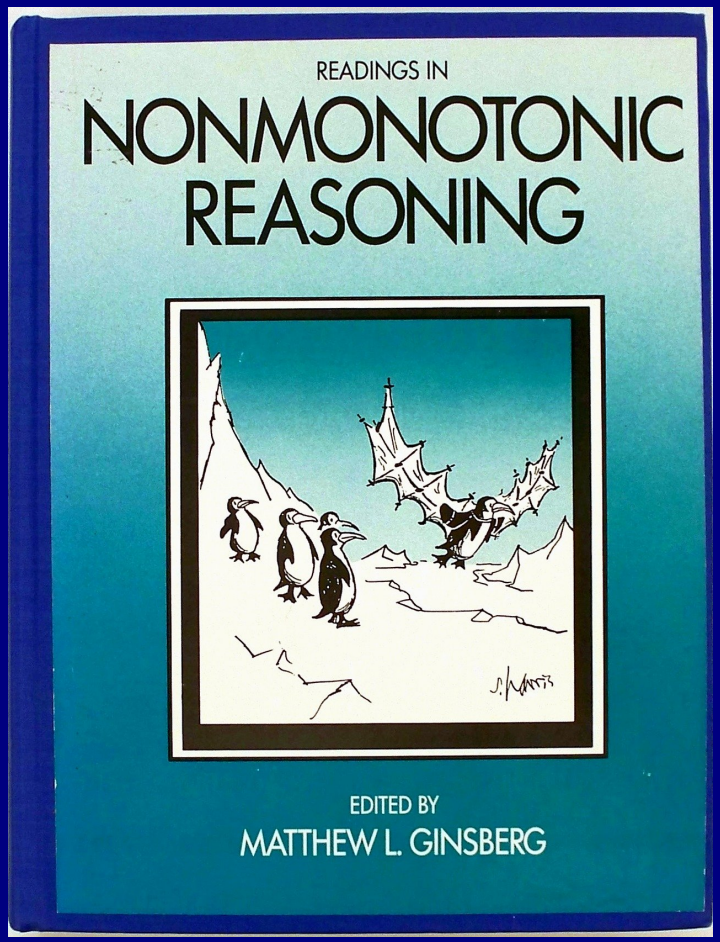
'87

'16

Potassco

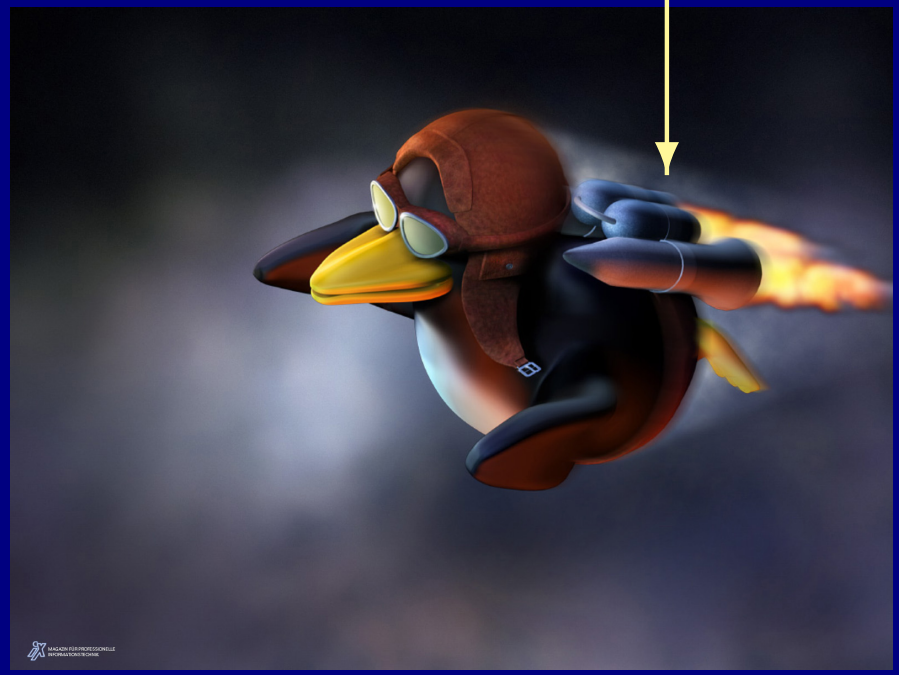
Epilogue

After all, it's all about Tweety!



'87

SAT



'16