Bridging the Gap: Foundedness, Defaults, and Expressivity in Constraint Answer Set Programming

Torsten Schaub

University of Potsdam & Potassco Solutions GmbH

Bridging the Gap: Foundedness, Defaults, and Expressivity in Constraint Answer Set Programming

Torsten Schaub[1]

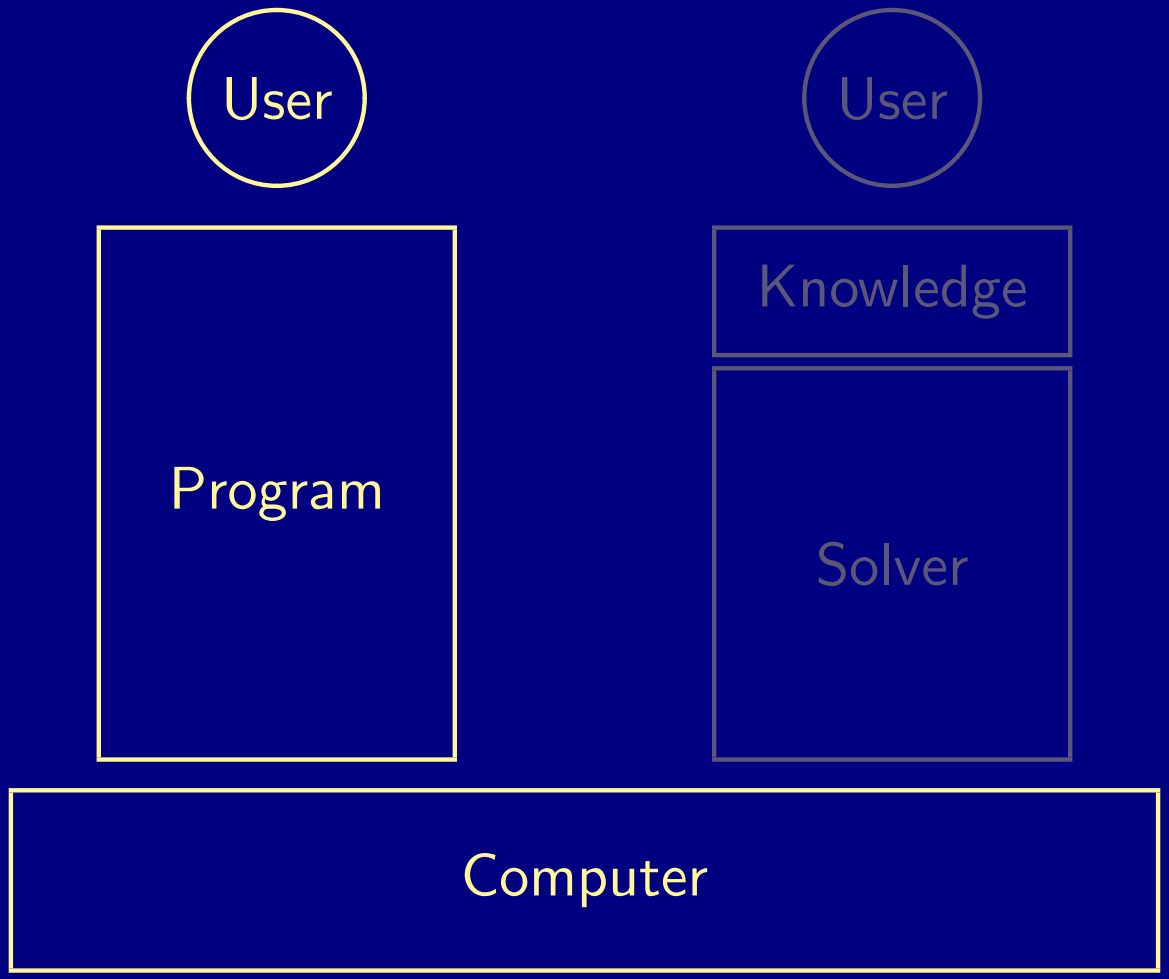
University of Potsdam & Potassco Solutions GmbH

[1] Joint work with Pedro Cabalar, Jorge Fandinno, Nicolas Rühling, Sebastian Schellhorn, and Philipp Wanko

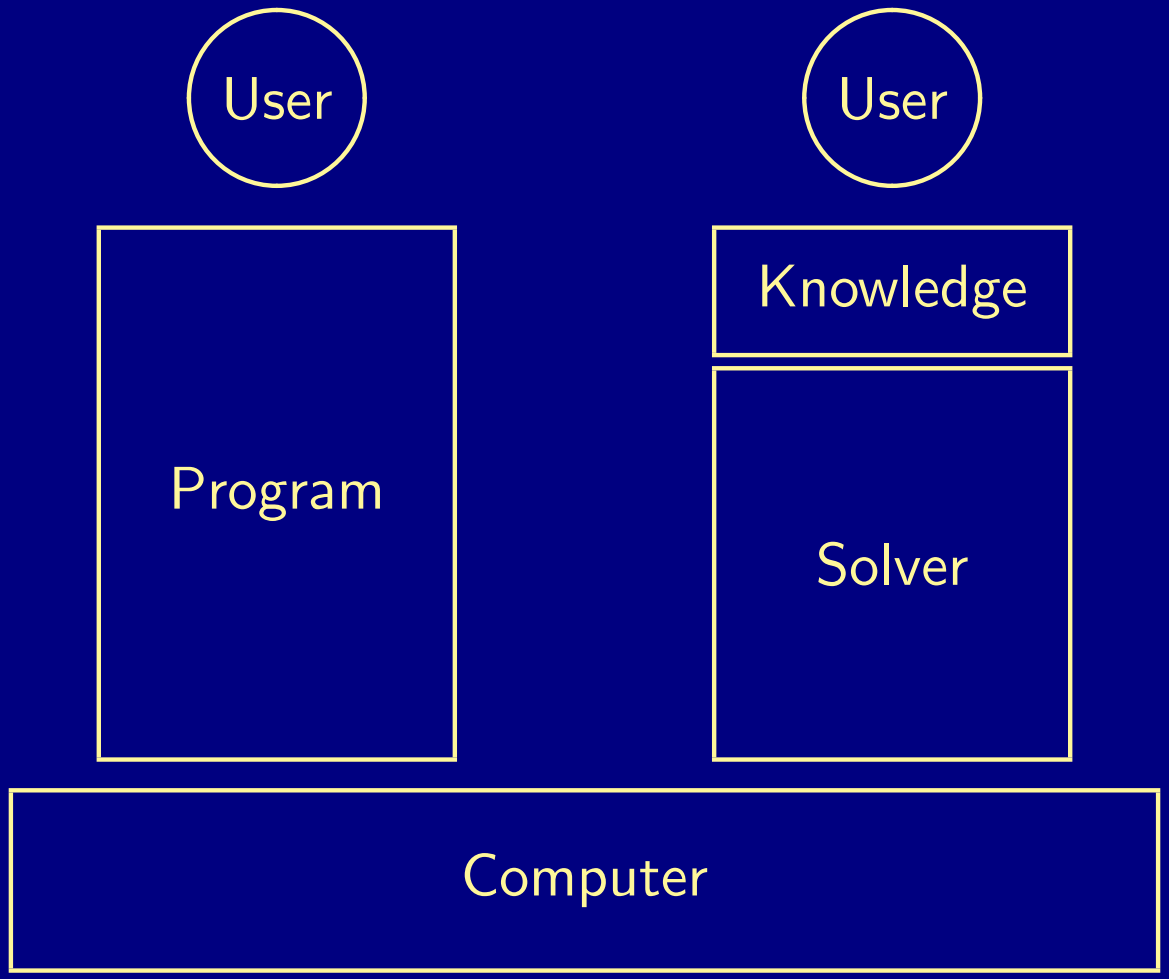
Outline

Outline

Traditional Software



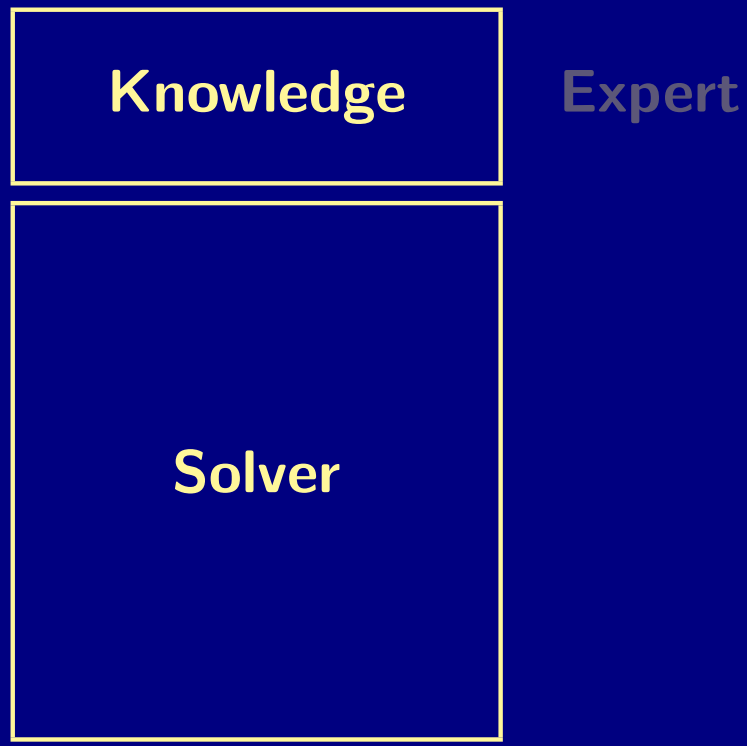
Knowledge-driven Software



What is the benefit?

+ Transparency
+ Flexibility
+ Maintainability
+ Reliability

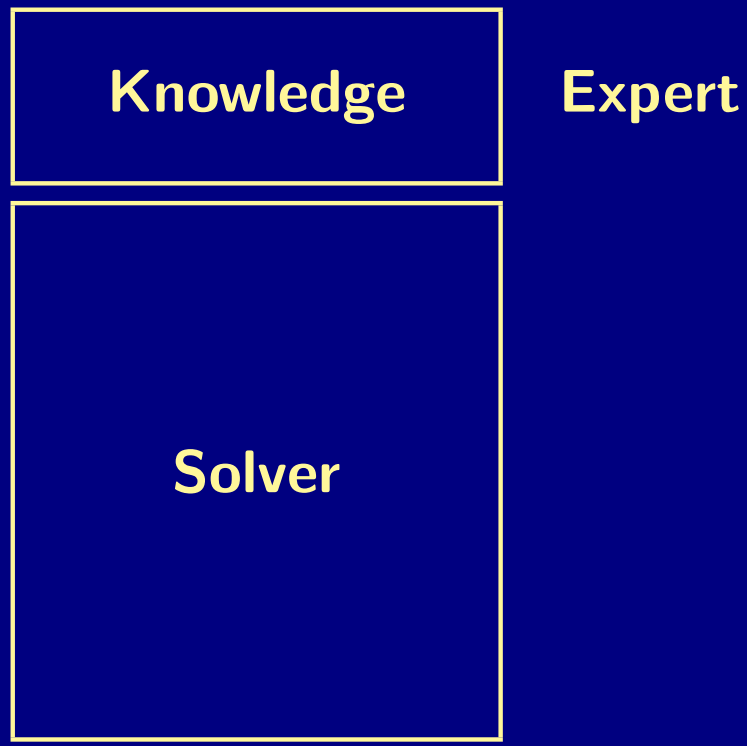
+ Generality
+ Efficiency
+ Optimality
+ Availability



What is the benefit?

+ Transparency
+ Flexibility
+ Maintainability
+ Reliability

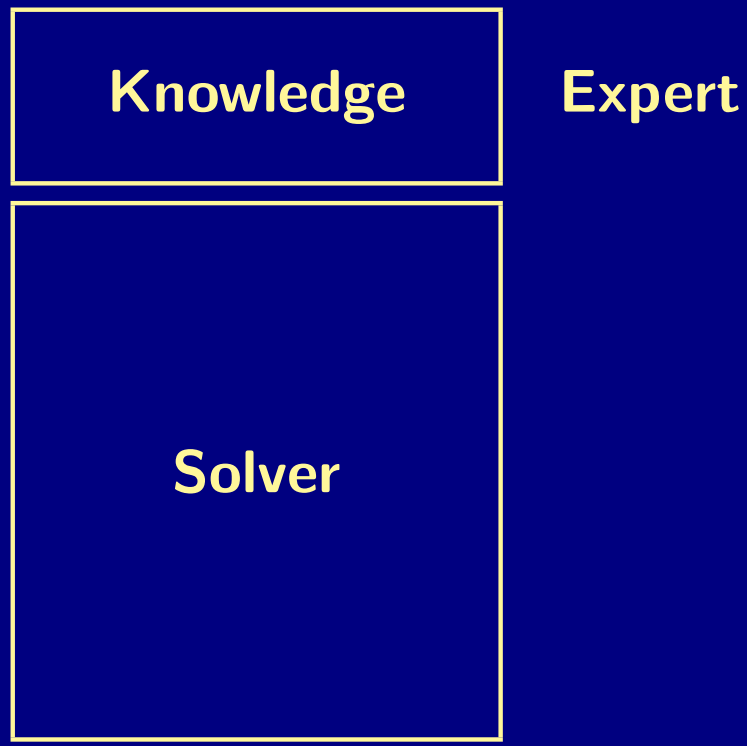
+ Generality
+ Efficiency
+ Optimality
+ Availability



What is the benefit?

+ Transparency
+ Flexibility
+ Maintainability
+ Reliability

+ Generality
+ Efficiency
+ Optimality
+ Availability



Industrial impact

Within SIEMENS, constraint technologies have been successfully used for solving configuration problems for more than 25 years. [...] approximately 80 percent of the maintenance costs and more than 60 percent of the development costs for the knowledge representation and reasoning tasks were saved.

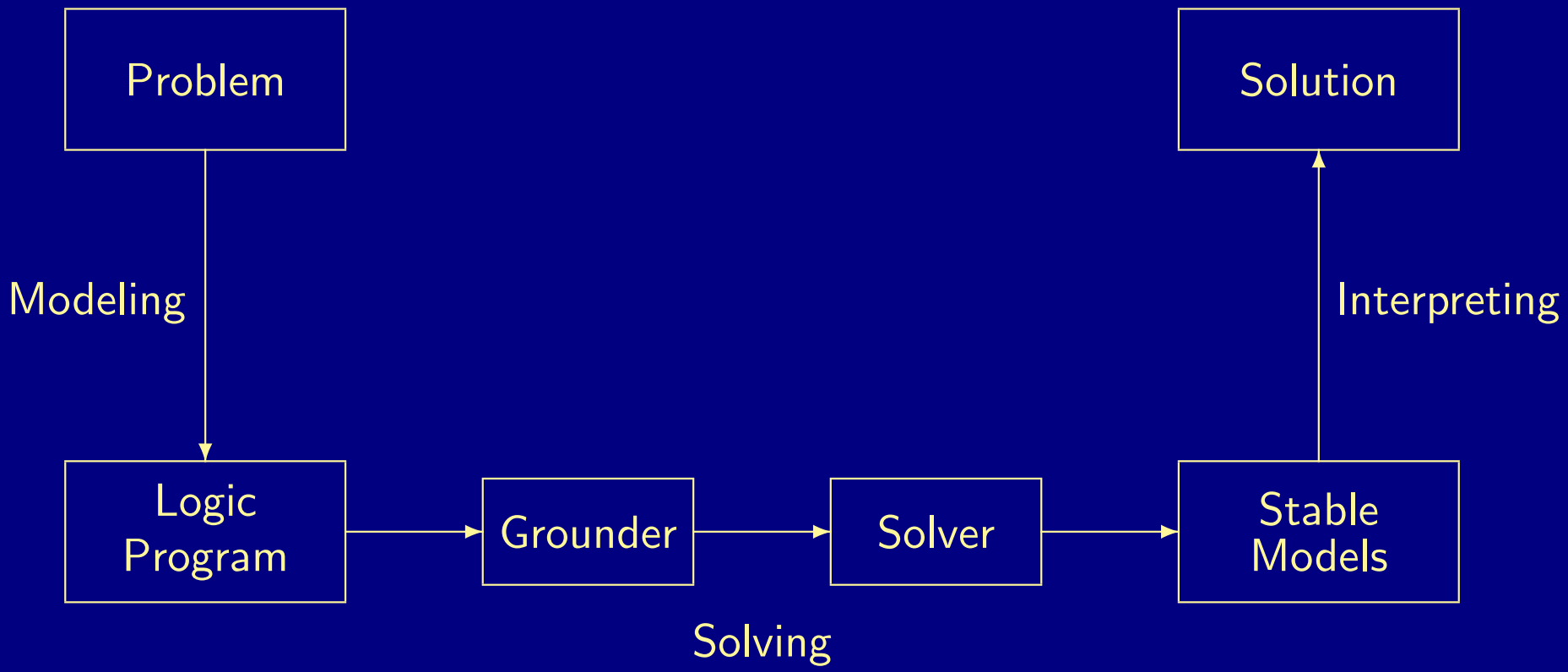
In: A. Falkner et al. Twenty-Five Years of Successful Application of Constraint Technologies at Siemens. AI Magazine. 37(4):67-80, 2016.

Industrial impact

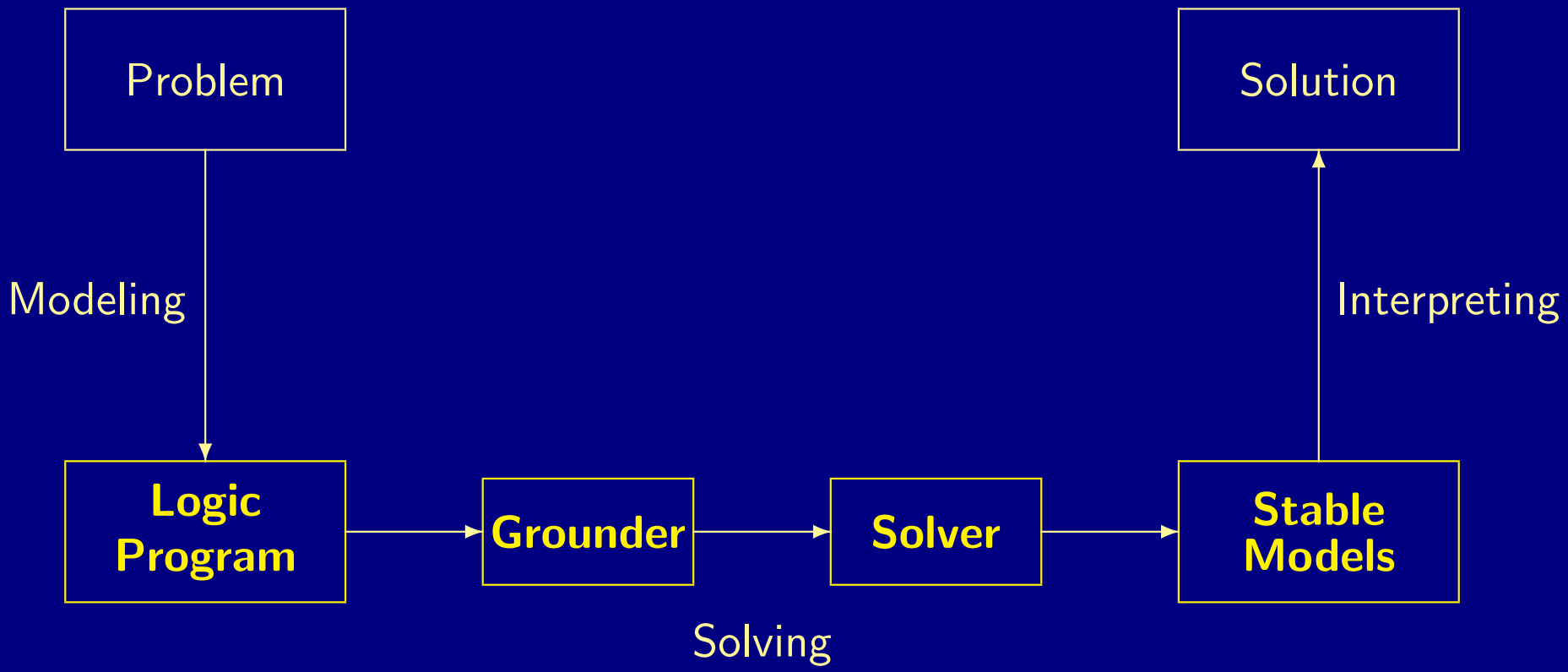
Within SIEMENS, constraint technologies have been successfully used for solving configuration problems for more than 25 years. [...] approximately 80 percent of the maintenance costs and more than 60 percent of the development costs for the knowledge representation and reasoning tasks were saved.

In: A. Falkner et al. Twenty-Five Years of Successful Application of Constraint Technologies at Siemens. AI Magazine. 37(4):67-80, 2016.

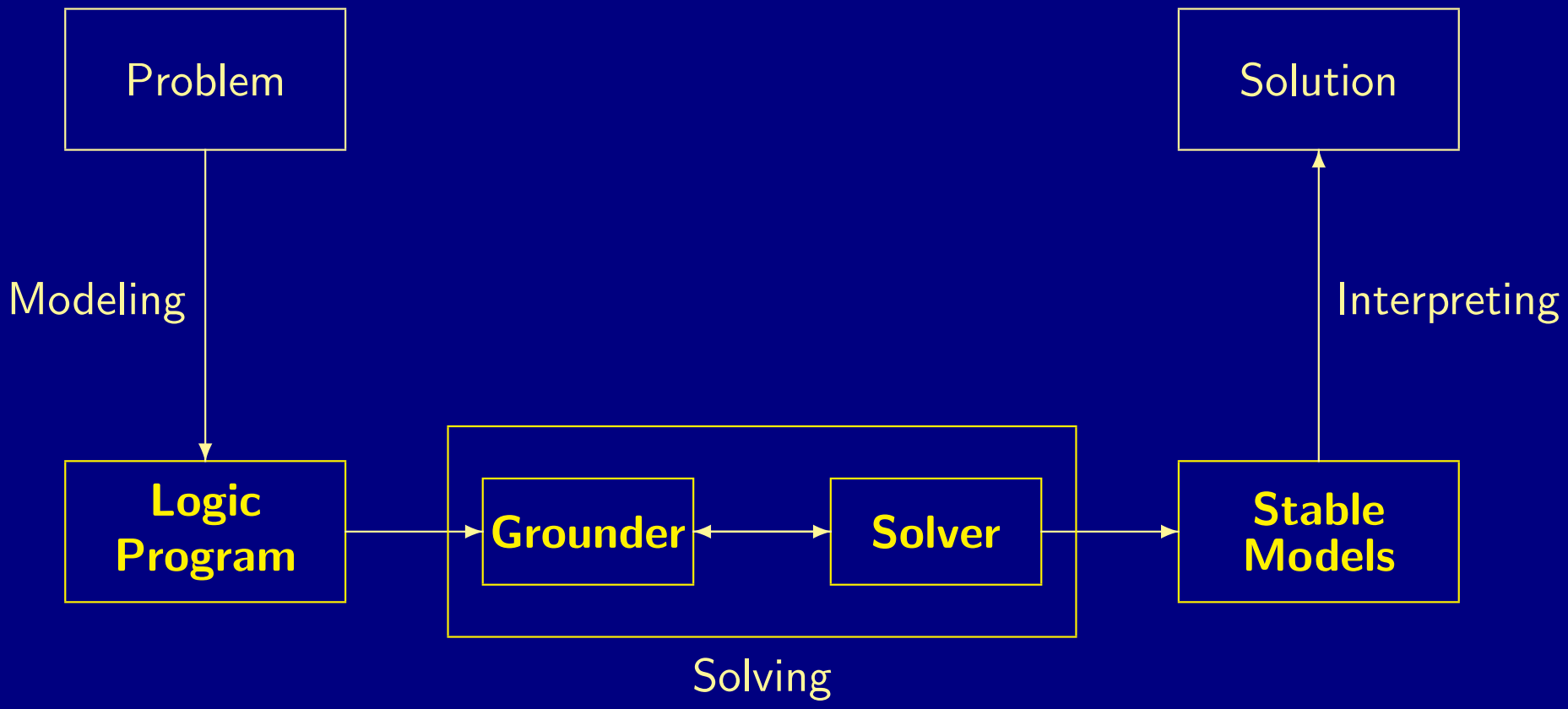
ASP solving process



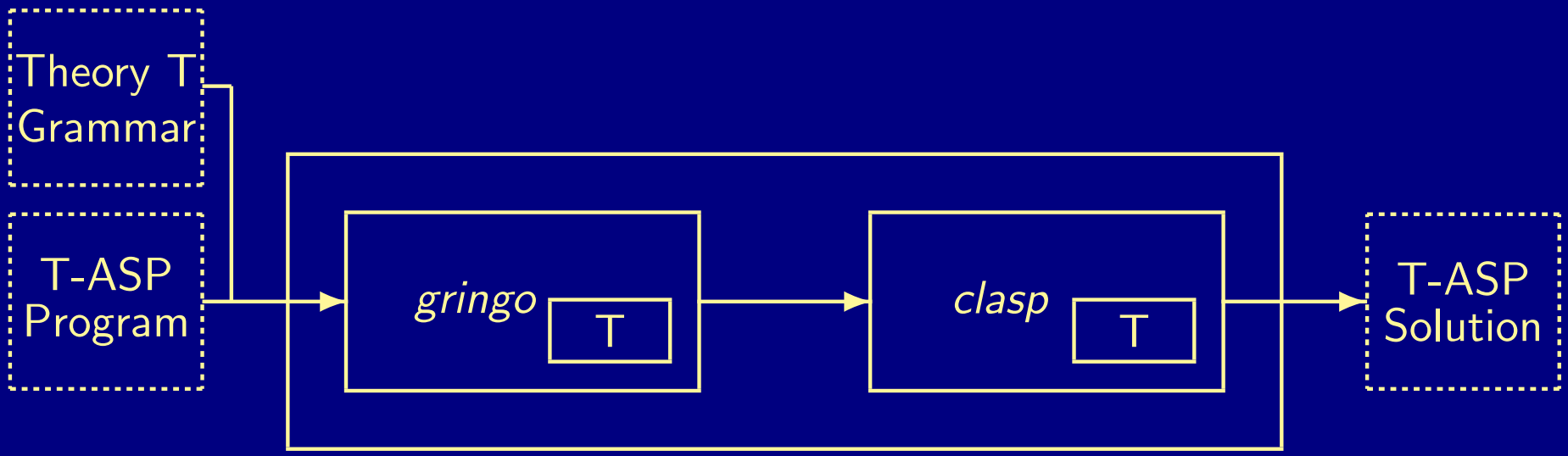
ASP solving process modulo theories



ASP solving process modulo theories

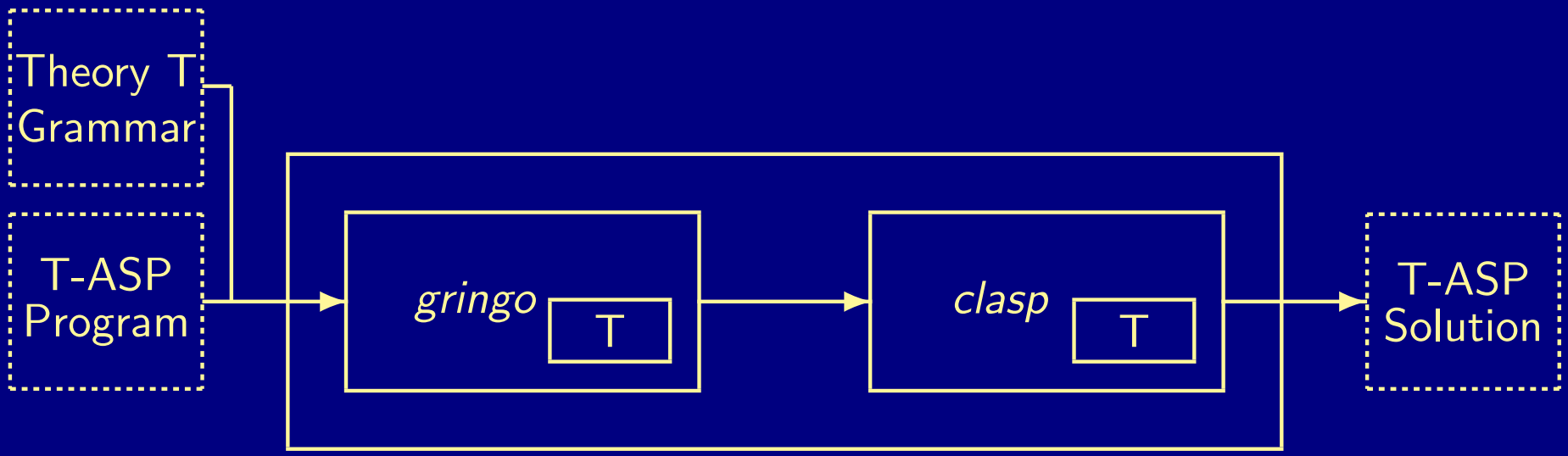


clingo's approach



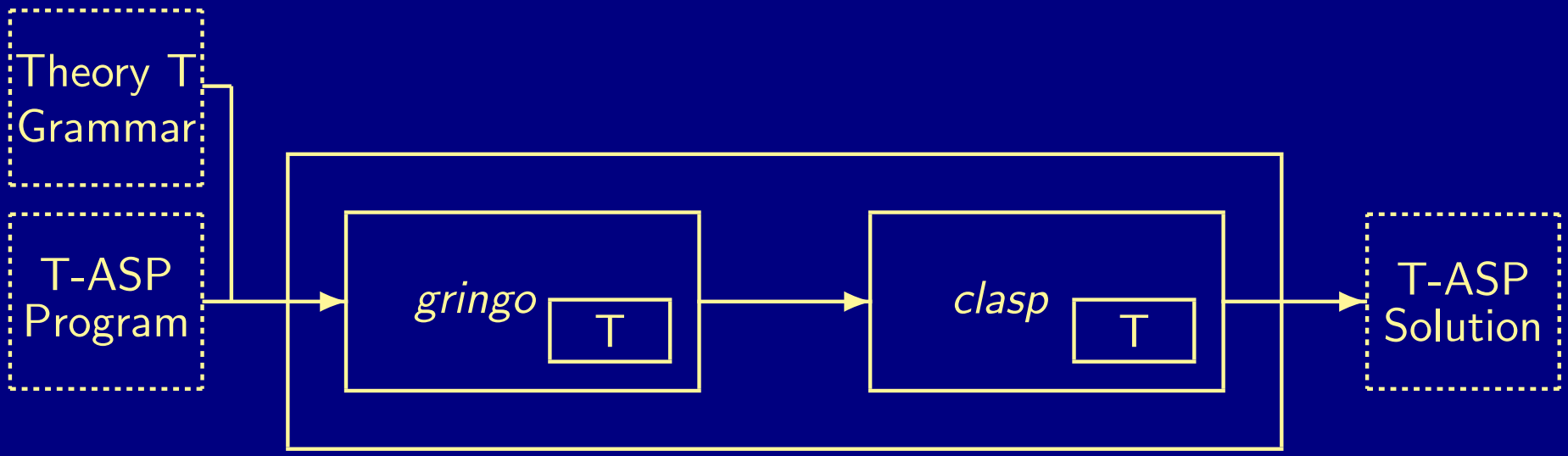
- Challenge Logic programs with elusive theory atoms
- Example The atom "`&sum{x;-y}<=4`" stands for difference constraint $x - y \leq 4$

clingo's approach



- Challenge Logic programs with elusive theory atoms
- Example The atom "`&sum{x;-y}<=4`" stands for difference constraint $x - y \leq 4$

clingo's approach



- Challenge Logic programs with elusive theory atoms
- Example The atom "`&sum{x;-y}<=4`" stands for difference constraint $x - y \leq 4$

CASP = ASP + CP

Constraint Answer Set Programming

- *clingo* extensions with linear constraints
  - *clingcon* — linear constraints over integers
  - *clingo*[DL] — difference constraints
  - *clingo*[LP] — linear constraints over floats
  - *clingo*[LPX] — linear constraints over rationals

$\hookrightarrow$ numeric variables are not grounded

CASP = ASP + CP

Constraint Answer Set Programming

- *clingo* extensions with linear constraints
  - *clingcon* — linear constraints over integers
  - *clingo*[DL] — difference constraints
  - *clingo*[LP] — linear constraints over floats
  - *clingo*[LPX] — linear constraints over rationals

$\hookrightarrow$ numeric variables are not grounded

¿ What about defaults ? Undefined variables ?
¡ Nope, numeric variables are monotonic !

CASP = ASP + CP

Constraint Answer Set Programming

- *clingo* extensions with linear constraints
  - *clingcon* — linear constraints over integers
  - *clingo*[DL] — difference constraints
  - *clingo*[LP] — linear constraints over floats
  - *clingo*[LPX] — linear constraints over rationals
- *clingo* extensions with linear constraints, non-monotonically
  - *lc2casp* — linear constraints over integers
  - *flingo* — linear constraints over integers
    - values of variables must be derived and may be undefined
    - defaults, assignments, definedness check
    - non-strict and strict aggregates

CASP = ASP + CP

Constraint Answer Set Programming

- *clingo* extensions with linear constraints
  - *clingcon* — linear constraints over integers
  - *clingo*[DL] — difference constraints
  - *clingo*[LP] — linear constraints over floats
  - *clingo*[LPX] — linear constraints over rationals
- *clingo* extensions with linear constraints, non-monotonically
  - *lc2casp* — linear constraints over integers
  - *flingo* — linear constraints over integers
    - values of variables must be derived and may be undefined
    - defaults, assignments, definedness check
    - non-strict and strict aggregates

¿ What does this mean ?

CASP = ASP + CP

Constraint Answer Set Programming

- *clingo* extensions with linear constraints
  - *clingcon* — linear constraints over integers
  - *clingo*[DL] — difference constraints
  - *clingo*[LP] — linear constraints over floats
  - *clingo*[LPX] — linear constraints over rationals
- *clingo* extensions with linear constraints, non-monotonically
  - *lc2casp* — linear constraints over integers
  - *flingo* — linear constraints over integers
    - values of variables must be derived and may be undefined
    - defaults, assignments, definedness check
    - non-strict and strict aggregates

¿ What does this mean ?

¡ For semantics, start here, go there, and find an equilibrium !

Outline

Stable model

- A logic program P is a set of rules, r, of the form

$$a \leftarrow b_1, \ldots, b_m, \neg c_1, \ldots, \neg c_n$$

- The reduct, P^X, of a program P relative to a set X of atoms is

$$P^X = \{a \leftarrow b_1, \ldots, b_m \mid r \in P,\ \{c_1, \ldots, c_n\} \cap X = \emptyset\}$$

- $Cn(P)$ stands for the smallest model of a positive program P

- A set X of atoms is a stable model of a program P if $Cn(P^X) = X$

Stable model

- A logic program P is a set of rules, r, of the form

$$a \leftarrow b_1, \ldots, b_m, \neg c_1, \ldots, \neg c_n$$

- The reduct, P^X, of a program P relative to a set X of atoms is

$$P^X = \{a \leftarrow b_1, \ldots, b_m \mid r \in P,\ \{c_1, \ldots, c_n\} \cap X = \emptyset\}$$

- $Cn(P)$ stands for the smallest model of a positive program P

- A set X of atoms is a stable model of a program P if $Cn(P^X) = X$

Stable model

- A logic program P is a set of rules, r, of the form

$$a \leftarrow b_1, \ldots, b_m, \neg c_1, \ldots, \neg c_n$$

- The reduct, P^X, of a program P relative to a set X of atoms is

$$P^X = \{a \leftarrow b_1, \ldots, b_m \mid r \in P,\ \{c_1, \ldots, c_n\} \cap X = \emptyset\}$$

- $Cn(P)$ stands for the smallest model of a positive program P

- A set X of atoms is a stable model of a program P if $Cn(P^X) = X$

Stable Model

continued

Stable Model

continued

Twelve Definitions of a Stable Model

Vladimir Lifschitz

Department of Computer Sciences, University of Texas at Austin, USA
vl@cs.utexas.edu

Abstract. This is a review of some of the definitions of the concept of a stable model that have been proposed in the literature. These definitions are equivalent to each other, at least when applied to traditional Prolog-style programs, but there are reasons why each of them is valuable and interesting. A new characterization of stable models can suggest an alternative picture of the intuitive meaning of logic programs; or it can lead to new algorithms for generating stable models; or it can work better than others when we turn to generalizations of the traditional syntax that are important from the perspective of answer set programming; or it can be more convenient for use in proofs; or it can be interesting simply because it demonstrates a relationship between seemingly unrelated ideas.

1 Introduction

This is a review of some of the definitions, or characterizations, of the concept of a stable model that have been proposed in the literature. These definitions are equivalent to each other when applied to "traditional rules"—with an atom in the head and a list of atoms, some possibly preceded with the negation as failure symbol, in the body:

$$A_0 \leftarrow A_1, \dots, A_m, not\ A_{m+1}, not\ A_n. \quad (1)$$

But there are reasons why each of them is valuable and interesting. A new characterization of stable models can suggest an alternative picture of the intuitive meaning of logic programs; or it can lead to new algorithms for generating stable models; or it can work better when we turn to generalizations of the traditional syntax that are important from the perspective of answer set programming (ASP); or it can be more convenient for use in proofs, such as proofs of correctness of ASP programs; or, quite simply, it can intellectually excite us by demonstrating a relationship between seemingly unrelated ideas.

We concentrate here primarily on programs consisting of finitely many rules of type (1), although generalizations of this syntactic form are mentioned several times in the second half of the paper. Some work on the stable model semantics (for instance, [Gelfond and Lifschitz, 1990], [Lifschitz *et al.*, 1999], [Niemelä and Simons, 2000], [Balduccini and Gelfond, 2003]) is not discussed here simply because it is about extending, rather than modifying, the definitions proposed earlier; this kind of work does not tell us much new about stable models of traditional programs.

Stable Model

continued

Twelve Definitions of a Stable Model

Vladimir Lifschitz

Department of Computer Sciences, University of Texas at Austin, USA
vl@cs.utexas.edu

Abstract. This is a review of some of the definitions of the concept of a stable model that have been proposed in the literature. These definitions are equivalent to each other, at least when applied to traditional Prolog-style programs, but there are reasons why each of them is valuable and interesting. A new characterization of stable models can suggest an alternative picture of the intuitive meaning of logic programs; or it can lead to new algorithms for generating stable models; or it can work better than others when we turn to generalizations of the traditional syntax that are important from the perspective of answer set programming; or it can be more convenient for use in proofs; or it can be interesting simply because it demonstrates a relationship between seemingly unrelated ideas.

1 Introduction

This is a review of some of the definitions, or characterizations, of the concept of a stable model that have been proposed in the literature. These definitions are equivalent to each other when applied to "traditional rules"—with an atom in the head and a list of atoms, some possibly preceded with the negation as failure symbol, in the body:

$$A_0 \leftarrow A_1, \ldots, A_m, not\ A_{m+1}, not\ A_n. \quad (1)$$

But there are reasons why each of them is valuable and interesting. A new characterization of stable models can suggest an alternative picture of the intuitive meaning of logic programs; or it can lead to new algorithms for generating stable models; or it can work better when we turn to generalizations of the traditional syntax that are important from the perspective of answer set programming (ASP); or it can be more convenient for use in proofs, such as proofs of correctness of ASP programs; or, quite simply, it can intellectually excite us by demonstrating a relationship between seemingly unrelated ideas.

We concentrate here primarily on programs consisting of finitely many rules of type (1), although generalizations of this syntactic form are mentioned several times in the second half of the paper. Some work on the stable model semantics (for instance, [Gelfond and Lifschitz, 1990], [Lifschitz *et al.*, 1999], [Niemelä and Simons, 2000], [Balduccini and Gelfond, 2003]) is not discussed here simply because it is about extending, rather than modifying, the definitions proposed earlier; this kind of work does not tell us much new about stable models of traditional programs.

Thirteen Definitions of a Stable Model

Vladimir Lifschitz

Department of Computer Sciences, University of Texas at Austin, USA
vl@cs.utexas.edu

Abstract. Stable models of logic programs have been studied by many researchers, mainly because of their role in the foundations of answer set programming. This is a review of some of the definitions of the concept of a stable model that have been proposed in the literature. These definitions are equivalent to each other, at least when applied to traditional Prolog-style programs, but there are reasons why each of them is valuable and interesting. A new characterization of stable models can suggest an alternative picture of the intuitive meaning of logic programs; or it can lead to new algorithms for generating stable models; or it can work better than others when we turn to generalizations of the traditional syntax that are important from the perspective of answer set programming; or it can be more convenient for use in proofs; or it can be interesting simply because it demonstrates a relationship between seemingly unrelated ideas.

1 Introduction

Stable models of logic programs have been studied by many researchers, mainly because of their role in the foundations of answer set programming (ASP) [30, 37]. This programming paradigm provides a declarative approach to solving combinatorial search problems, and it has found applications in several areas of science and technology [21]. In ASP, a search problem is reduced to computing stable models, and programs for generating stable models ("answer set solvers") are used to perform search.

This paper is a review of some of the definitions, or characterizations, of the concept of a stable model that have been proposed in the literature. These definitions are equivalent to each other when applied to "traditional rules" – with an atom in the head and a list of atoms, some possibly preceded with the negation as failure symbol, in the body:

$$A_0 \leftarrow A_1, \ldots, A_m, not\ A_{m+1}, not\ A_n. \quad (1)$$

But there are reasons why each of them is valuable and interesting. A new characterization of stable models can suggest an alternative picture of the intuitive meaning of logic programs; or it can lead to new algorithms for generating stable models; or it can work better when we turn to generalizations of the traditional syntax that are important from the perspective of answer set programming; or it can be more convenient for use in proofs, such as proofs of correctness of ASP

Start here

Logic à la George Boole

- Alphabet A set $\mathcal{A}$ of atoms
- Formula $\varphi ::= \bot \mid a \in \mathcal{A} \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi \rightarrow \varphi$

  Definition $\neg\varphi = \varphi \rightarrow \bot$
- Interpretation A set I of atoms with $I \subseteq \mathcal{A}$
- Intuition I represents true atoms, $\mathcal{A} \setminus I$ represents false atoms
- Satisfaction
  1. $I \models a$ if $a \in I$
  2. $I \models \varphi \wedge \psi$ if $I \models \varphi$ and $I \models \psi$
  3. $I \models \varphi \vee \psi$ if $I \models \varphi$ or $I \models \psi$
  4. $I \models \varphi \rightarrow \psi$ if $I \not\models \varphi$ or $I \models \psi$

  $I \models \neg\varphi$ if $I \not\models \varphi$
- Model An interpretation I such that $I \models \varphi$

Logic à la George Boole

- Alphabet A set $\mathcal{A}$ of atoms
- Formula $\varphi ::= \bot \mid a \in \mathcal{A} \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi \rightarrow \varphi$

  Definition $\neg\varphi = \varphi \rightarrow \bot$
- Interpretation A set I of atoms with $I \subseteq \mathcal{A}$
- Intuition I represents true atoms, $\mathcal{A} \setminus I$ represents false atoms
- Satisfaction
  1. $I \models a$ if $a \in I$
  2. $I \models \varphi \wedge \psi$ if $I \models \varphi$ and $I \models \psi$
  3. $I \models \varphi \vee \psi$ if $I \models \varphi$ or $I \models \psi$
  4. $I \models \varphi \rightarrow \psi$ if $I \not\models \varphi$ or $I \models \psi$
  5. $I \models \neg\varphi$ if $I \not\models \varphi$
- Model An interpretation I such that $I \models \varphi$

Logic à la George Boole

- Alphabet A set $\mathcal{A}$ of atoms
- Formula $\varphi ::= \bot \mid a \in \mathcal{A} \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi \rightarrow \varphi$

  Definition $\neg\varphi = \varphi \rightarrow \bot$
- Interpretation A set I of atoms with $I \subseteq \mathcal{A}$
- Intuition I represents true atoms, $\mathcal{A} \setminus I$ represents false atoms
- Satisfaction
  1. $I \models a$ if $a \in I$
  2. $I \models \varphi \wedge \psi$ if $I \models \varphi$ and $I \models \psi$
  3. $I \models \varphi \vee \psi$ if $I \models \varphi$ or $I \models \psi$
  4. $I \models \varphi \rightarrow \psi$ if $I \not\models \varphi$ or $I \models \psi$
  5. $I \models \neg\varphi$ if $I \not\models \varphi$
- Model An interpretation I such that $I \models \varphi$

Logic à la George Boole

- Alphabet A set $\mathcal{A}$ of atoms
- Formula $\varphi ::= \bot \mid a \in \mathcal{A} \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi \rightarrow \varphi$

  Definition $\neg\varphi = \varphi \rightarrow \bot$
- Interpretation A set I of atoms with $I \subseteq \mathcal{A}$
- Intuition I represents true atoms, $\mathcal{A} \setminus I$ represents false atoms
- Satisfaction
  1. $I \models a$ if $a \in I$
  2. $I \models \varphi \wedge \psi$ if $I \models \varphi$ and $I \models \psi$
  3. $I \models \varphi \vee \psi$ if $I \models \varphi$ or $I \models \psi$
  4. $I \models \varphi \rightarrow \psi$ if $I \not\models \varphi$ or $I \models \psi$
  5. $I \models \neg\varphi$ if $I \not\models \varphi$
- Model An interpretation I such that $I \models \varphi$

Logic à la George Boole

- Alphabet A set $\mathcal{A}$ of atoms
- Formula $\varphi ::= \bot \mid a \in \mathcal{A} \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi \rightarrow \varphi$

  Definition $\neg\varphi = \varphi \rightarrow \bot$
- Interpretation A set I of atoms with $I \subseteq \mathcal{A}$
- Intuition I represents true atoms, $\mathcal{A} \setminus I$ represents false atoms
- Satisfaction
  1. $I \models a$ if $a \in I$
  2. $I \models \varphi \wedge \psi$ if $I \models \varphi$ and $I \models \psi$
  3. $I \models \varphi \vee \psi$ if $I \models \varphi$ or $I \models \psi$
  4. $I \models \varphi \rightarrow \psi$ if $I \not\models \varphi$ or $I \models \psi$
  5. $I \models \neg\varphi$ if $I \not\models \varphi$
- Model An interpretation I such that $I \models \varphi$

Logic à la George Boole

- Alphabet A set $\mathcal{A}$ of atoms
- Formula $\varphi ::= \bot \mid a \in \mathcal{A} \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi \rightarrow \varphi$

  Definition $\neg\varphi = \varphi \rightarrow \bot$
- Assignment A set I of atoms with $I \subseteq \mathcal{A}$
- Intuition I represents true atoms, $\mathcal{A} \setminus I$ represents false atoms
- Satisfaction
  1. $I \models a$ if $a \in I$
  2. $I \models \varphi \wedge \psi$ if $I \models \varphi$ and $I \models \psi$
  3. $I \models \varphi \vee \psi$ if $I \models \varphi$ or $I \models \psi$
  4. $I \models \varphi \rightarrow \psi$ if $I \not\models \varphi$ or $I \models \psi$
  5. $I \models \neg\varphi$ if $I \not\models \varphi$
- Solution An assignment I such that $I \models \varphi$

Start here, go there

Logic à la Arend Heyting

and Kurt Gödel

- Alphabet A set $\mathcal{A}$ of atoms
- Formula $\varphi ::= \bot \mid a \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi \rightarrow \varphi$
- Interpretation A pair $\langle H, T \rangle$ of sets of atoms with $H \subseteq T \subseteq \mathcal{A}$
  - H is called "here" and
  - T is called "there"

  Note $\langle H, T \rangle$ is a simplified Kripke structure
- Intuition
  - H represents provably true atoms
  - T represents possibly true atoms
  - $\mathcal{A} \setminus T$ represents false atoms

Logic à la Arend Heyting

and Kurt Gödel

- Alphabet A set $\mathcal{A}$ of atoms
- Formula $\varphi ::= \bot \mid a \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi \rightarrow \varphi$
- Interpretation A pair $\langle H, T \rangle$ of sets of atoms with $H \subseteq T \subseteq \mathcal{A}$
  - H is called "here" and
  - T is called "there"

  Note $\langle H, T \rangle$ is a simplified Kripke structure
- Intuition
  - H represents provably true atoms
  - T represents possibly true atoms
  - $\mathcal{A} \setminus T$ represents false atoms

Logic à la Arend Heyting

and Kurt Gödel

- Alphabet A set $\mathcal{A}$ of atoms
- Formula $\varphi ::= \bot \mid a \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi \rightarrow \varphi$
- Interpretation A pair $\langle H, T \rangle$ of sets of atoms with $H \subseteq T \subseteq \mathcal{A}$
  - H is called "here" and
  - T is called "there"

  Note $\langle H, T \rangle$ is a simplified Kripke structure
- Intuition
  - H represents provably true atoms
  - T represents possibly true atoms
  - $\mathcal{A} \setminus T$ represents false atoms

Logic à la Arend Heyting

and Kurt Gödel

- Satisfaction
  1. $\langle H, T\rangle \models a$ if $a \in H$
  2. $\langle H, T\rangle \models \varphi \wedge \psi$ if $\langle H, T\rangle \models \varphi$ and $\langle H, T\rangle \models \psi$
  3. $\langle H, T\rangle \models \varphi \vee \psi$ if $\langle H, T\rangle \models \varphi$ or $\langle H, T\rangle \models \psi$
  4. $\langle H, T\rangle \models \varphi \rightarrow \psi$ if $\langle X, T\rangle \not\models \varphi$ or $\langle X, T\rangle \models \psi$ for both $X = H, T$
- Intuition
  - $\langle H, T\rangle \models \varphi \sim \varphi$ is provably true
  - $\langle T, T\rangle \models \varphi \sim \varphi$ is possibly true
- Model An interpretation $\langle H, T\rangle$ such that $\langle H, T\rangle \models \varphi$

Logic à la Arend Heyting

and Kurt Gödel

- Satisfaction
  1. $\langle H, T\rangle \models a$ if $a \in H$
  2. $\langle H, T\rangle \models \varphi \wedge \psi$ if $\langle H, T\rangle \models \varphi$ and $\langle H, T\rangle \models \psi$
  3. $\langle H, T\rangle \models \varphi \vee \psi$ if $\langle H, T\rangle \models \varphi$ or $\langle H, T\rangle \models \psi$
  4. $\langle H, T\rangle \models \varphi \rightarrow \psi$ if $\langle X, T\rangle \not\models \varphi$ or $\langle X, T\rangle \models \psi$ for both $X = H, T$
- Intuition
  - $\langle H, T\rangle \models \varphi \sim \varphi$ is provably true
  - $\langle T, T\rangle \models \varphi \sim \varphi$ is possibly true
- Model An interpretation $\langle H, T\rangle$ such that $\langle H, T\rangle \models \varphi$

Logic à la Arend Heyting

and Kurt Gödel

- Satisfaction
  1. $\langle H, T\rangle \models a$ if $a \in H$
  2. $\langle H, T\rangle \models \varphi \wedge \psi$ if $\langle H, T\rangle \models \varphi$ and $\langle H, T\rangle \models \psi$
  3. $\langle H, T\rangle \models \varphi \vee \psi$ if $\langle H, T\rangle \models \varphi$ or $\langle H, T\rangle \models \psi$
  4. $\langle H, T\rangle \models \varphi \rightarrow \psi$ if $\langle X, T\rangle \not\models \varphi$ or $\langle X, T\rangle \models \psi$ for both $X = H, T$
  5. $\langle H, T\rangle \models \neg\varphi$ if $\langle T, T\rangle \not\models \varphi$ $\quad (\neg\varphi = \varphi \rightarrow \bot)$
- Intuition
  - $\langle H, T\rangle \models \varphi \sim \varphi$ is provably true
  - $\langle T, T\rangle \models \varphi \sim \varphi$ is possibly true
- Model An interpretation $\langle H, T\rangle$ such that $\langle H, T\rangle \models \varphi$

Logic à la Arend Heyting

and Kurt Gödel

- Satisfaction
  1. $\langle H, T\rangle \models a$ if $a \in H$
  2. $\langle H, T\rangle \models \varphi \wedge \psi$ if $\langle H, T\rangle \models \varphi$ and $\langle H, T\rangle \models \psi$
  3. $\langle H, T\rangle \models \varphi \vee \psi$ if $\langle H, T\rangle \models \varphi$ or $\langle H, T\rangle \models \psi$
  4. $\langle H, T\rangle \models \varphi \rightarrow \psi$ if $\langle X, T\rangle \not\models \varphi$ or $\langle X, T\rangle \models \psi$ for both $X = H, T$
- Intuition
  - $\langle H, T\rangle \models \varphi \sim \varphi$ is provably true
  - $\langle T, T\rangle \models \varphi \sim \varphi$ is possibly true
- Model An interpretation $\langle H, T\rangle$ such that $\langle H, T\rangle \models \varphi$

Logic à la Arend Heyting

and Kurt Gödel

- Satisfaction
  1. $\langle H, T\rangle \models a$ if $a \in H$
  2. $\langle H, T\rangle \models \varphi \wedge \psi$ if $\langle H, T\rangle \models \varphi$ and $\langle H, T\rangle \models \psi$
  3. $\langle H, T\rangle \models \varphi \vee \psi$ if $\langle H, T\rangle \models \varphi$ or $\langle H, T\rangle \models \psi$
  4. $\langle H, T\rangle \models \varphi \rightarrow \psi$ if $\langle X, T\rangle \not\models \varphi$ or $\langle X, T\rangle \models \psi$ for both $X = H, T$
- Intuition
  - $\langle H, T\rangle \models \varphi \ \sim \ \varphi$ is provably true
  - $\langle T, T\rangle \models \varphi$ iff $T \models \varphi$
- Model An interpretation $\langle H, T\rangle$ such that $\langle H, T\rangle \models \varphi$

Logic à la Arend Heyting

and Kurt Gödel

- Satisfaction
  1. $\langle H, T\rangle \models a$ if $a \in H$
  2. $\langle H, T\rangle \models \varphi \wedge \psi$ if $\langle H, T\rangle \models \varphi$ and $\langle H, T\rangle \models \psi$
  3. $\langle H, T\rangle \models \varphi \vee \psi$ if $\langle H, T\rangle \models \varphi$ or $\langle H, T\rangle \models \psi$
  4. $\langle H, T\rangle \models \varphi \rightarrow \psi$ if $\langle X, T\rangle \not\models \varphi$ or $\langle X, T\rangle \models \psi$ for both $X = H, T$
- Intuition
  - $\langle H, T\rangle \models \varphi \ \sim \ \varphi$ is provably true
  - $\langle T, T\rangle \models \varphi$ iff $T \models \varphi$
- Model An interpretation $\langle H, T\rangle$ such that $\langle H, T\rangle \models \varphi$

Logic à la Arend Heyting

and Kurt Gödel

- Satisfaction
  1. $\langle H, T\rangle \models a$ if $a \in H$
  2. $\langle H, T\rangle \models \varphi \wedge \psi$ if $\langle H, T\rangle \models \varphi$ and $\langle H, T\rangle \models \psi$
  3. $\langle H, T\rangle \models \varphi \vee \psi$ if $\langle H, T\rangle \models \varphi$ or $\langle H, T\rangle \models \psi$
  4. $\langle H, T\rangle \models \varphi \rightarrow \psi$ if $\langle X, T\rangle \not\models \varphi$ or $\langle X, T\rangle \models \psi$ for both $X = H, T$
- Intuition
  - $\langle H, T\rangle \models \varphi \ \sim \ \varphi$ is provably true
  - $\langle T, T\rangle \models \varphi$ iff $T \models \varphi$
- Model An interpretation $\langle H, T\rangle$ such that $\langle H, T\rangle \models \varphi$

This system is called the Logic of Here-and-There (HT; Heyting, 1930), also known as G3 (Gödel, 1932)

Boolean vs Heytingian Logic

- Implication
  - Boolean $\varphi \rightarrow \phi$ is merely a logical shortcut for $\neg\varphi \vee \phi$
  - Heytingian Implication represents a directional derivation
- Foundedness Truth must be earned
  - The truth of a formula is only justified if there exists a derivation starting from the facts
  - This property separates Boolean from Heytingian-style systems
- Tautologies

	Boolean	Heytingian
$\varphi \vee \neg\varphi$	✓	×
$\neg\varphi \vee \neg\neg\varphi$	✓	✓
$\neg\neg\varphi \rightarrow \varphi$	✓	×
$\varphi \rightarrow \neg\neg\varphi$	✓	✓

Boolean vs Heytingian Logic

- Implication
  - Boolean $\varphi \rightarrow \phi$ is merely a logical shortcut for $\neg\varphi \vee \phi$
  - Heytingian Implication represents a directional derivation
- Foundedness Truth must be earned
  - The truth of a formula is only justified if there exists a derivation starting from the facts
  - This property separates Boolean from Heytingian-style systems
- Tautologies

	Boolean	Heytingian
$\varphi \vee \neg\varphi$	✓	×
$\neg\varphi \vee \neg\neg\varphi$	✓	✓
$\neg\neg\varphi \rightarrow \varphi$	✓	×
$\varphi \rightarrow \neg\neg\varphi$	✓	✓

Start here, go there, and find an equilibrium !

Non-monotonicity via Minimization

- Closed-world assumption Atoms are assumed false unless they are forced to be true
- John McCarthy's circumscription selects those Boolean models of a formula that minimize the set of true propositional variables
- David Pearce's equilibrium models are total Heytingian models of a formula that are minimally justified

Non-monotonicity via Minimization

- Closed-world assumption Atoms are assumed false unless they are forced to be true
- John McCarthy's circumscription selects those Boolean models of a formula that minimize the set of true propositional variables
- David Pearce's equilibrium models are total Heytingian models of a formula that are minimally justified

Non-monotonicity via Minimization

- Closed-world assumption Atoms are assumed false unless they are forced to be true
- John McCarthy's circumscription selects those Boolean models of a formula that minimize the set of true propositional variables
- David Pearce's equilibrium models are total Heytingian models of a formula that are minimally justified

David Pearce's equilibrium logic

- Equilibrium Model
  A total Heytingian model $\langle T, T\rangle$ is an equilibrium model of a formula φ if there is no proper subset $H \subset T$ such that $\langle H, T\rangle \models \varphi$
- Theorem The equilibrium models of a logic program correspond exactly to its stable models (answer sets)

  Stable models are (minimal) Boolean models
- Example

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

$$b \vee \neg b$$
$$b \wedge \neg c \rightarrow a$$

David Pearce's equilibrium logic

- Equilibrium Model
  A total Heytingian model $\langle T, T \rangle$ is an equilibrium model of a formula φ if there is no proper subset $H \subset T$ such that $\langle H, T \rangle \models \varphi$
- Theorem The equilibrium models of a logic program correspond exactly to its stable models (answer sets)

  Stable models are (minimal) Boolean models
- Example

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

$$b \vee \neg b$$
$$b \wedge \neg c \rightarrow a$$

David Pearce's equilibrium logic

- Equilibrium Model
  A total Heytingian model $\langle T, T \rangle$ is an equilibrium model of a formula φ if there is no proper subset $H \subset T$ such that $\langle H, T \rangle \models \varphi$
- Theorem The equilibrium models of a logic program correspond exactly to its stable models (answer sets)

  Stable models are (minimal) Boolean models
- Example

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

$$b \vee \neg b$$
$$b \wedge \neg c \rightarrow a$$

David Pearce's equilibrium logic

- Equilibrium Model
  A total Heytingian model $\langle T, T\rangle$ is an equilibrium model of a formula φ if there is no proper subset $H \subset T$ such that $\langle H, T\rangle \models \varphi$
- Theorem The equilibrium models of a logic program correspond exactly to its stable models (answer sets)

  Stable models are (minimal) Boolean models
- Example

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

$b \vee \neg b$

$b \wedge \neg c \rightarrow a$

David Pearce's equilibrium logic

- Equilibrium Model
  A total Heytingian model $\langle T, T\rangle$ is an equilibrium model of a formula φ if there is no proper subset $H \subset T$ such that $\langle H, T\rangle \models \varphi$
- Theorem The equilibrium models of a logic program correspond exactly to its stable models (answer sets)

  Stable models are (minimal) Boolean models
- Example

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

$b \vee \neg b$

$b \wedge \neg c \rightarrow a$

Stable models
{}, {a,b}

David Pearce's equilibrium logic

- Equilibrium Model
  A total Heytingian model $\langle T, T\rangle$ is an equilibrium model of a formula φ if there is no proper subset $H \subset T$ such that $\langle H, T\rangle \models \varphi$
- Theorem The equilibrium models of a logic program correspond exactly to its stable models (answer sets)

  Stable models are (minimal) Boolean models
- Example

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

$b \vee \neg b$

$b \wedge \neg c \rightarrow a$

Stable models
{}, {a,b}

David Pearce's equilibrium logic

- Equilibrium Model
  A total Heytingian model $\langle T, T\rangle$ is an equilibrium model of a formula φ if there is no proper subset $H \subset T$ such that $\langle H, T\rangle \models \varphi$
- Theorem The equilibrium models of a logic program correspond exactly to its stable models (answer sets)

  Stable models are (minimal) Boolean models
- Example

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

$b \vee \neg b$

$b \wedge \neg c \rightarrow a$

Stable models
{}, {a,b}

Equilibrium models
$\langle \emptyset, \emptyset\rangle$, $\langle \{a, b\}, \{a, b\}\rangle$

Outline

Foundedness in CASP systems

- Example

```
{ a }.                    a ∨ ¬a
"x > 7" :- a.             a → x > 7
```

- *clingcon* produces infinitely many solutions:
  - a is true and x takes all values greater than 7
  - a is false and x takes all possible integer values
- *flingo* yields infinitely many solutions:
  - a is true and x takes all values greater than 7
  - a is false and x is undefined
- *clingo*[DL] obtains only two solutions:
  - a is true and x is 8 and
  - a is false and x is undefined

Foundedness in CASP systems

- Example

```
{ a }.
&sum{ x } > 7 :- a.
```

$a \vee \neg a$

$a \rightarrow x > 7$

- *clingcon* produces infinitely many solutions:
  - a is true and x takes all values greater than 7
  - a is false and x takes all possible integer values
- *flingo* yields infinitely many solutions:
  - a is true and x takes all values greater than 7
  - a is false and x is undefined
- *clingo*[DL] obtains only two solutions:
  - a is true and x is 8 and
  - a is false and x is undefined

Foundedness in CASP systems

- Example

```
{ a }.                          a ∨ ¬a
&sum{ x } > 7 :- a.             a → x > 7
```

- *clingcon* produces infinitely many solutions:
  - a is true and x takes all values greater than 7
  - a is false and x takes all possible integer values
- *flingo* yields infinitely many solutions:
  - a is true and x takes all values greater than 7
  - a is false and x is undefined
- *clingo*[DL] obtains only two solutions:
  - a is true and x is 8 and
  - a is false and x is undefined

Foundedness in CASP systems

- Example

```
{ a }.
&sum{ x } > 7 :- a.
```

$a \vee \neg a$

$a \rightarrow x > 7$

- *clingcon* produces infinitely many solutions:
  - a is true and x takes all values greater than 7
  - a is false and x takes all possible integer values
- *flingo* yields infinitely many solutions:
  - a is true and x takes all values greater than 7
  - a is false and x is undefined
- *clingo*[DL] obtains only two solutions:
  - a is true and x is 8 and
  - a is false and x is undefined

Foundedness in CASP systems

- Example

```
{ a }.                          a ∨ ¬a
&sum{ x } > 7 :- a.             a → x > 7
```

- *clingcon* produces infinitely many solutions:
  - a is true and x takes all values greater than 7
  - a is false and x takes all possible integer values
- *flingo* yields infinitely many solutions:
  - a is true and x takes all values greater than 7
  - a is false and x is undefined
- *clingo*[DL] obtains only two solutions:
  - a is true and x is 8 and
  - a is false and x is undefined

Foundedness in CASP systems

- Example

`{ a }.`	$a \vee \neg a$	+ Axioms
`&sum{ x } > 7 :- a.`	$a \rightarrow x > 7$	

- *clingcon* produces infinitely many solutions:
  - a is true and x takes all values greater than 7
  - a is false and x takes all possible integer values
- *flingo* yields infinitely many solutions:
  - a is true and x takes all values greater than 7
  - a is false and x is undefined
- *clingo*[DL] obtains only two solutions:
  - a is true and x is 8 and
  - a is false and x is undefined

$\hookrightarrow$ Logic of Here-and-There with Constraints (HT_c)

Foundedness in CASP systems

- Example

```
{ a }.
&sum{ x } > 7 :- a.
```

$a \vee \neg a$ $\quad$ $a \rightarrow x > 7$ $\quad$ + Axioms

- *clingcon* produces infinitely many solutions:
  - a is true and x takes all values greater than 7
  - a is false and x takes all possible integer values
- *flingo* yields infinitely many solutions:
  - a is true and x takes all values greater than 7
  - a is false and x is undefined
- *clingo*[DL] obtains only two solutions:
  - a is true and x is 8 and
  - a is false and x is undefined

$\hookrightarrow$ Bound-founded Logic of Here-and-There (HT_b)

Foundedness in CASP systems

- Example

```
{ a }.
&sum{ x } > 7 :- a.
```

$a \vee \neg a$ $\quad$ $a \rightarrow x > 7$ $\quad$ + Axioms

- *clingcon* produces infinitely many solutions:
  - a is true and x takes all values greater than 7
  - a is false and x takes all possible integer values
- *flingo* yields infinitely many solutions:
  - a is true and x takes all values greater than 7
  - a is false and x is undefined
- *clingo*[DL] obtains only two solutions:
  - a is true and x is 8 and
  - a is false and x is undefined

$\hookrightarrow$ Bound-founded Logic of Here-and-There ("Peter Stuckey's HT_b")

Foundedness in CASP systems

- Example

`{ a }.`	$a \vee \neg a$	+ Axioms
`&sum{ x } > 7 :- a.`	$a \rightarrow x > 7$	

- *clingcon* produces infinitely many solutions:
  - a is true and x takes all values greater than 7
  - a is false and x takes all possible integer values
- *flingo* yields infinitely many solutions:
  - a is true and x takes all values greater than 7
  - a is false and x is undefined
- *clingo*[DL] obtains only two solutions:
  - a is true and x is 8 and
  - a is false and x is undefined

$\hookrightarrow$ Logic of Here-and-There with Constraints (HT_c)

HT_c Syntax

- Signature $\langle \mathcal{X}, \mathcal{D}, \mathcal{A} \rangle$
  - $\mathcal{X}$ variables
  - $\mathcal{D}$ domain
  - $\mathcal{A}$ atoms
- Note The syntax of atoms is left open
- Example Atom "$x - y \leq d$" with $x, y \in \mathcal{X}$ and $d \in \mathcal{D}$
- HT_c-formula φ over $\mathcal{A}$

$$\varphi ::= \bot \mid a \in \mathcal{A} \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi \rightarrow \varphi$$

- Notation For $a \in \mathcal{X}$ and $\boldsymbol{t} \in \mathcal{D}$, we abbreviate $(a = \boldsymbol{t}) \in \mathcal{A}$ by a.

HT_c Syntax

- Signature $\langle \mathcal{X}, \mathcal{D}, \mathcal{A} \rangle$
  - $\mathcal{X}$ variables
  - $\mathcal{D}$ domain
  - $\mathcal{A}$ atoms
- Note The syntax of atoms is left open
- Example Atom "$x - y \leq d$" with $x, y \in \mathcal{X}$ and $d \in \mathcal{D}$
- HT_c-formula φ over $\mathcal{A}$

$$\varphi \ ::= \ \bot \ \mid \ a \in \mathcal{A} \ \mid \ \varphi \wedge \varphi \ \mid \ \varphi \vee \varphi \ \mid \ \varphi \rightarrow \varphi$$

- Notation For $a \in \mathcal{X}$ and $\boldsymbol{t} \in \mathcal{D}$, we abbreviate $(a = \boldsymbol{t}) \in \mathcal{A}$ by a.

HT_c Syntax

- Signature $\langle \mathcal{X}, \mathcal{D}, \mathcal{A} \rangle$
  - $\mathcal{X}$ variables
  - $\mathcal{D}$ domain
  - $\mathcal{A}$ atoms
- Note The syntax of atoms is left open
- Example Atom "$x - y \leq d$" with $x, y \in \mathcal{X}$ and $d \in \mathcal{D}$
- HT_c-formula φ over $\mathcal{A}$

$$\varphi \ ::= \ \bot \ \mid \ a \in \mathcal{A} \ \mid \ \varphi \wedge \varphi \ \mid \ \varphi \vee \varphi \ \mid \ \varphi \rightarrow \varphi$$

- Notation For $a \in \mathcal{X}$ and $\boldsymbol{t} \in \mathcal{D}$, we abbreviate $(a = \boldsymbol{t}) \in \mathcal{A}$ by a.

HT_c Syntax

- Signature $\langle \mathcal{X}, \mathcal{D}, \mathcal{A} \rangle$
  - $\mathcal{X}$ variables
  - $\mathcal{D}$ domain
  - $\mathcal{A}$ atoms
- Note The syntax of atoms is left open
- Example Atom "$x - y \leq d$" with $x, y \in \mathcal{X}$ and $d \in \mathcal{D}$
- HT_c-formula φ over $\mathcal{A}$

$$\varphi \ ::= \ \bot \ \mid \ a \in \mathcal{A} \ \mid \ \varphi \wedge \varphi \ \mid \ \varphi \vee \varphi \ \mid \ \varphi \rightarrow \varphi$$

- Notation For $a \in \mathcal{X}$ and $\boldsymbol{t} \in \mathcal{D}$, we abbreviate $(a = \boldsymbol{t}) \in \mathcal{A}$ by a.

HT_c Semantics

- Valuation $v : \mathcal{X} \to \mathcal{D} \cup \{\boldsymbol{u}\}$
  - $\boldsymbol{u} \notin \mathcal{X} \cup \mathcal{D}$ stands for undefined

  Set-based representation $v \subseteq \mathcal{X} \times \mathcal{D}$
  - $(x, c) \in v$ and $(x, d) \in v$ implies $c = d$
  - $(x, d) \notin v$ if $v(x) = \boldsymbol{u}$

  $\mathcal{V}$ is the set of all valuations over $\mathcal{X}$ and $\mathcal{D}$
- Atom denotation $[\![\cdot]\!] : \mathcal{A} \to 2^{\mathcal{V}}$
- Example

$$[\![\ \text{“}x - y \leq d\text{”}\ ]\!] = \{v \in \mathcal{V} \mid v(x), v(y), d \in \mathbb{Z},\ v(x) - v(y) \leq d\}$$

HT_c Semantics

- Valuation $v : \mathcal{X} \to \mathcal{D} \cup \{\boldsymbol{u}\}$
  - $\boldsymbol{u} \notin \mathcal{X} \cup \mathcal{D}$ stands for undefined

  Set-based representation $v \subseteq \mathcal{X} \times \mathcal{D}$
  - $(x, c) \in v$ and $(x, d) \in v$ implies $c = d$
  - $(x, d) \notin v$ if $v(x) = \boldsymbol{u}$

  $\mathcal{V}$ is the set of all valuations over $\mathcal{X}$ and $\mathcal{D}$
- Atom denotation $[\![\cdot]\!] : \mathcal{A} \to 2^{\mathcal{V}}$
- Example

$$[\![\ \text{“}x - y \leq d\text{”}\ ]\!] = \{v \in \mathcal{V} \mid v(x), v(y), d \in \mathbb{Z},\ v(x) - v(y) \leq d\}$$

HT_c Semantics

- Valuation $v : \mathcal{X} \to \mathcal{D} \cup \{\boldsymbol{u}\}$
  - $\boldsymbol{u} \notin \mathcal{X} \cup \mathcal{D}$ stands for undefined

  Set-based representation $v \subseteq \mathcal{X} \times \mathcal{D}$
  - $(x, c) \in v$ and $(x, d) \in v$ implies $c = d$
  - $(x, d) \notin v$ if $v(x) = \boldsymbol{u}$

  $\mathcal{V}$ is the set of all valuations over $\mathcal{X}$ and $\mathcal{D}$
- Atom denotation $[\![\cdot]\!] : \mathcal{A} \to 2^{\mathcal{V}}$
- Example

$$[\![\text{“}x - y \leq d\text{”}]\!] = \{v \in \mathcal{V} \mid v(x), v(y), d \in \mathbb{Z},\ v(x) - v(y) \leq d\}$$

HT_c Semantics

- Valuation $v : \mathcal{X} \to \mathcal{D} \cup \{\boldsymbol{u}\}$
  - $\boldsymbol{u} \notin \mathcal{X} \cup \mathcal{D}$ stands for undefined

  Set-based representation $v \subseteq \mathcal{X} \times \mathcal{D}$
  - $(x, c) \in v$ and $(x, d) \in v$ implies $c = d$
  - $(x, d) \notin v$ if $v(x) = \boldsymbol{u}$

  $\mathcal{V}$ is the set of all valuations over $\mathcal{X}$ and $\mathcal{D}$
- Atom denotation $[\![\cdot]\!] : \mathcal{A} \to 2^{\mathcal{V}}$
- Example

$$[\![\text{“}x - y \leq d\text{”}]\!] = \{v \in \mathcal{V} \mid v(x), v(y), d \in \mathbb{Z},\ v(x) - v(y) \leq d\}$$

HT_c Satisfaction

- HT_c-interpretation over $\mathcal{X}$,$\mathcal{D}$ is a pair $\langle h, t\rangle$ of valuations over $\mathcal{X}$,$\mathcal{D}$ such that $h \subseteq t$
- An HT_c-interpretation $\langle h, t\rangle$ satisfies a formula φ, written $\langle h, t\rangle \models \varphi$, if the following conditions hold
  1. $\langle h, t\rangle \not\models \bot$
  2. $\langle h, t\rangle \models a$ if both $h \in [\![a]\!]$ and $t \in [\![a]\!]$ for $a \in \mathcal{A}$
  3. $\langle h, t\rangle \models \varphi \wedge \psi$ if $\langle h, t\rangle \models \varphi$ and $\langle h, t\rangle \models \psi$
  4. $\langle h, t\rangle \models \varphi \vee \psi$ if $\langle h, t\rangle \models \varphi$ or $\langle h, t\rangle \models \psi$
  5. $\langle h, t\rangle \models \varphi \rightarrow \psi$ if $\langle h', t\rangle \not\models \varphi$ or $\langle h', t\rangle \models \psi$ for both $h' = h$ and $h' = t$.

HT_c Satisfaction

- HT_c-interpretation over $\mathcal{X}$,$\mathcal{D}$ is a pair $\langle h, t\rangle$ of valuations over $\mathcal{X}$,$\mathcal{D}$ such that $h \subseteq t$
- An HT_c-interpretation $\langle h, t\rangle$ satisfies a formula φ, written $\langle h, t\rangle \models \varphi$, if the following conditions hold
  1. $\langle h, t\rangle \not\models \bot$
  2. $\langle h, t\rangle \models a$ if both $h \in [\![a]\!]$ and $t \in [\![a]\!]$ for $a \in \mathcal{A}$
  3. $\langle h, t\rangle \models \varphi \wedge \psi$ if $\langle h, t\rangle \models \varphi$ and $\langle h, t\rangle \models \psi$
  4. $\langle h, t\rangle \models \varphi \vee \psi$ if $\langle h, t\rangle \models \varphi$ or $\langle h, t\rangle \models \psi$
  5. $\langle h, t\rangle \models \varphi \rightarrow \psi$ if $\langle h', t\rangle \not\models \varphi$ or $\langle h', t\rangle \models \psi$ for both $h' = h$ and $h' = t$.

HT_c Equilibrium model

- A total interpretation $\langle t, t \rangle$ is an equilibrium model of a formula φ, if
  1. $\langle t, t \rangle \models \varphi$
  2. $\langle h, t \rangle \not\models \varphi$ for all $h \subset t$
- t is called an HT_c-stable model of φ

HT_c Equilibrium model

- A total interpretation $\langle t, t \rangle$ is an equilibrium model of a formula φ, if
  1. $\langle t, t \rangle \models \varphi$
  2. $\langle h, t \rangle \not\models \varphi$ for all $h \subset t$
- t is called an HT_c-stable model of φ

Outline

clingcon summation constraints

$\&\mathtt{sum}\{t_1; \ldots; t_n\} \prec s$ strict sum[1]

[1] t_i, s product terms, x an integer variable, $\prec$ a comparison

flingo summation constraints

$\&\mathtt{sum}\{t_1;\ldots;t_n\} \prec s$	non-strict sum[1]
$\&\mathtt{sus}\{t_1;\ldots;t_n\} \prec s$	strict sum[1]

[1] t_i, s product terms, x an integer variable, $\prec$ a comparison

summation constraints

and more

$\&\mathtt{sum}\{t_1;\ldots;t_n\} \prec s$	non-strict sum[1]
$\&\mathtt{sus}\{t_1;\ldots;t_n\} \prec s$	strict sum[1]
$\&\mathtt{df}(x)$	definedness
$\&\mathtt{int}(x)$	integrality

[1] t_i, s product terms, x an integer variable, $\prec$ a comparison

clingcon semantics

summation constraints

$$[\![\,\&\mathtt{sum}\{t_1;\ldots;t_n\} \prec s\,]\!] = \{v \in \mathcal{V} \mid v(t_i) \in \mathbb{Z}, \textstyle\sum_{1\leq i\leq n} v(t_i) \prec v(s)\}$$

$$[\![\,\&\mathtt{df}(x)\,]\!] = \{v \in \mathcal{V} \mid v(x) \neq \boldsymbol{u}\}$$
$$[\![\,\&\mathtt{int}(x)\,]\!] = \{v \in \mathcal{V} \mid v(x) \in \mathbb{Z}\}$$

$$v(n) = n \qquad v(n * x) = \begin{cases} n * v(x) & \text{if } v(x) \in \mathbb{Z} \\ \boldsymbol{u} & \text{otherwise} \end{cases}$$

flingo semantics

summation constraints

$$[\![\,\&\mathtt{sum}\{t_1;\ldots;t_n\} \prec s\,]\!] = \{v \in \mathcal{V} \mid \dot{v}(t_i) \in \mathbb{Z}, \textstyle\sum_{1\leq i\leq n} \dot{v}(t_i) \prec v(s)\}$$
$$[\![\,\&\mathtt{sus}\{t_1;\ldots;t_n\} \prec s\,]\!] = \{v \in \mathcal{V} \mid v(t_i) \in \mathbb{Z}, \textstyle\sum_{1\leq i\leq n} v(t_i) \prec v(s)\}$$

$$[\![\,\&\mathtt{df}(x)\,]\!] = \{v \in \mathcal{V} \mid v(x) \neq \boldsymbol{u}\}$$
$$[\![\,\&\mathtt{int}(x)\,]\!] = \{v \in \mathcal{V} \mid v(x) \in \mathbb{Z}\}$$

$$v(n) = n \quad v(n*x) = \begin{cases} n*v(x) & \text{if } v(x)\in\mathbb{Z} \\ \boldsymbol{u} & \text{otherwise} \end{cases} \quad \dot{v}(t) = \begin{cases} v(t) & \text{if } v(t)\in\mathbb{Z} \\ 0 & \text{otherwise} \end{cases}$$

flingo semantics

summation constraints

$$[\![\,\&\mathtt{sum}\{t_1;\ldots;t_n\} \prec s\,]\!] = \{v \in \mathcal{V} \mid \dot{v}(t_i) \in \mathbb{Z}, \textstyle\sum_{1\leq i\leq n} \dot{v}(t_i) \prec v(s)\}$$
$$[\![\,\&\mathtt{sus}\{t_1;\ldots;t_n\} \prec s\,]\!] = \{v \in \mathcal{V} \mid v(t_i) \in \mathbb{Z}, \textstyle\sum_{1\leq i\leq n} v(t_i) \prec v(s)\}$$

$$[\![\,\&\mathtt{df}(x)\,]\!] = \{v \in \mathcal{V} \mid v(x) \neq \boldsymbol{u}\}$$
$$[\![\,\&\mathtt{int}(x)\,]\!] = \{v \in \mathcal{V} \mid v(x) \in \mathbb{Z}\}$$

$$v(n) = n \quad v(n*x) = \begin{cases} n*v(x) & \text{if } v(x) \in \mathbb{Z} \\ \boldsymbol{u} & \text{otherwise} \end{cases} \quad \dot{v}(t) = \begin{cases} v(t) & \text{if } v(t) \in \mathbb{Z} \\ 0 & \text{otherwise} \end{cases}$$

Denotational semantics

summation constraints and more

$$[\![\,\&\mathtt{sum}\{t_1;\ldots;t_n\} \prec s\,]\!] = \{v \in \mathcal{V} \mid \dot{v}(t_i) \in \mathbb{Z}, \textstyle\sum_{1\leq i\leq n} \dot{v}(t_i) \prec v(s)\}$$
$$[\![\,\&\mathtt{sus}\{t_1;\ldots;t_n\} \prec s\,]\!] = \{v \in \mathcal{V} \mid v(t_i) \in \mathbb{Z}, \textstyle\sum_{1\leq i\leq n} v(t_i) \prec v(s)\}$$

$$[\![\,\&\mathtt{df}(x)\,]\!] = \{v \in \mathcal{V} \mid v(x) \neq \boldsymbol{u}\}$$
$$[\![\,\&\mathtt{int}(x)\,]\!] = \{v \in \mathcal{V} \mid v(x) \in \mathbb{Z}\}$$

$$v(n) = n \quad v(n*x) = \begin{cases} n*v(x) & \text{if } v(x)\in\mathbb{Z} \\ \boldsymbol{u} & \text{otherwise} \end{cases} \quad \dot{v}(t) = \begin{cases} v(t) & \text{if } v(t)\in\mathbb{Z} \\ 0 & \text{otherwise} \end{cases}$$

Example

- CASP program

```
{ a }.
&sum{ x } > 7 :- a.
```

- *clingcon* semantics

$$a \vee \neg a$$
$$a \rightarrow x > 7$$
$$\&\mathtt{df}(a) \rightarrow a$$
$$\&\mathtt{int}(x)$$

- *flingo* semantics

$$a \vee \neg a$$
$$a \rightarrow x > 7$$
$$\&\mathtt{df}(a) \rightarrow a$$
$$\&\mathtt{df}(x) \rightarrow \&\mathtt{int}(x)$$

Example

- CASP program

```
{ a }.
&sum{ x } > 7 :- a.
```

- *clingcon* semantics

$a \vee \neg a$
$a \rightarrow x > 7$
$\&\mathtt{df}(a) \rightarrow a$
$\&\mathtt{int}(x)$

- *flingo* semantics

$a \vee \neg a$
$a \rightarrow x > 7$
$\&\mathtt{df}(a) \rightarrow a$
$\&\mathtt{df}(x) \rightarrow \&\mathtt{int}(x)$

Example

- CASP program

```
{ a }.
&sum{ x } > 7 :- a.
```

- *clingcon* semantics

$(a = \boldsymbol{t}) \vee \neg(a = \boldsymbol{t})$
$(a = \boldsymbol{t}) \rightarrow x > 7$
$\&\mathtt{df}(a) \rightarrow (a = \boldsymbol{t})$
$\&\mathtt{int}(x)$

- *flingo* semantics

$a \vee \neg a$
$a \rightarrow x > 7$
$\&\mathtt{df}(a) \rightarrow a$
$\&\mathtt{df}(x) \rightarrow \&\mathtt{int}(x)$

Example

- CASP program

```
{ a }.
&sum{ x } > 7 :- a.
```

- *clingcon* semantics

$a \vee \neg a$
$a \rightarrow x > 7$
$\&\mathtt{df}(a) \rightarrow a$
$\&\mathtt{int}(x)$

- *flingo* semantics

$a \vee \neg a$
$a \rightarrow x > 7$
$\&\mathtt{df}(a) \rightarrow a$
$\&\mathtt{df}(x) \rightarrow \&\mathtt{int}(x)$

Example

Tariffs

```
&sum{ tariff(cars,ca) } = 25.
&sum{ tariff(P,eu) } = 15 :- sales(P,eu,_),
                             not &sus{ tariff(P,eu) } != 15.
&sum{ tariff(steel,eu) } = 0.
&sum{ tariff(aircraft,eu) } = 25.

&sum{ Y*tariff(P,C),P,C : sales(P,C,X), Y=X/100 } =: taxincome.
```

Example

Tariffs

```
&sum{ tariff(cars,ca) } = 25.
&sum{ tariff(P,eu) } = 15 :- sales(P,eu,_),
                             not &sus{ tariff(P,eu) } != 15.
&sum{ tariff(steel,eu) } = 0.
&sum{ tariff(aircraft,eu) } = 25.

&sum{ Y*tariff(P,C),P,C : sales(P,C,X), Y=X/100 } =: taxincome.
```

Example

Tariffs

```
&sum{ tariff(cars,ca) } = 25.
&sum{ tariff(P,eu) } = 15 :- sales(P,eu,_),
                             not &sus{ tariff(P,eu) } != 15.
&sum{ tariff(steel,eu) } = 0.
&sum{ tariff(aircraft,eu) } = 25.

&sum{ Y*tariff(P,C),P,C : sales(P,C,X), Y=X/100 } =: taxincome.
```

Example

Configuration

```
price(frame,15).
default_range(1,2).
select(frame).                          { select(bag) }.

&sus{ V } = price(P)     :- select(P), price(P,V).
&in{ L..U } =: price(P) :- select(P), default_range(L,U),
                  not &sus{ price(P) } < L, not &sus{ price(P) } > U.

&sus{ price(P) : select(P) } =: price(total).
```

Example

Configuration

```
price(frame,15).
default_range(1,2).
select(frame).                          { select(bag) }.

&sus{ V } = price(P)     :- select(P), price(P,V).
&in{ L..U } =: price(P) :- select(P), default_range(L,U),
                  not &sus{ price(P) } < L, not &sus{ price(P) } > U.

&sus{ price(P) : select(P) } =: price(total).
```

Example

Configuration

```
price(frame,15).
default_range(1,2).
select(frame).                          { select(bag) }.

&sus{ V } = price(P)     :- select(P), price(P,V).
&in{ L..U } =: price(P) :- select(P), default_range(L,U),
                  not &sus{ price(P) } < L, not &sus{ price(P) } > U.

&sus{ price(P) : select(P) } =: price(total).
```

Example

Configuration

```
price(frame,15).
default_range(1,2).
select(frame).                          { select(bag) }.

&sus{ V } = price(P)     :- select(P), price(P,V).
&in{ L..U } =: price(P) :- select(P), default_range(L,U),
                  not &sus{ price(P) } < L, not &sus{ price(P) } > U.

&sus{ price(P) : select(P) } =: price(total).
```

Outline

Summary: CASP = ASP+CP

- Logical Foundations
  - Logic of Here-and-There with Constraints (HT_c)
    - partial valuations
    - constructive implication
    - undefinedness and default values
  - Equilibrium Logic
    - non-monotonicity via minimization
    - basis for answer set style semantics
- CASP Systems
  - *clingcon* = ASP + monotonic CP
  - *flingo* = ASP + non-monotonic CP
- Visit us at `potassco.org` !

Summary: CASP = ASP+CP

- Logical Foundations
  - Logic of Here-and-There with Constraints (HT_c)
    - partial valuations
    - constructive implication
    - undefinedness and default values
  - Equilibrium Logic
    - non-monotonicity via minimization
    - basis for answer set style semantics
- CASP Systems
  - *clingcon* = ASP + monotonic CP
  - *flingo* = ASP + non- and monotonic CP
- Visit us at `potassco.org` !

Summary: CASP = ASP+CP

- Logical Foundations
  - Logic of Here-and-There with Constraints (HT_c)
    - partial valuations
    - constructive implication
    - undefinedness and default values
  - Equilibrium Logic
    - non-monotonicity via minimization
    - basis for answer set style semantics
- CASP Systems
  - *clingcon* = ASP + monotonic CP
  - *flingo* = ASP + non- and monotonic CP
- Visit us at `potassco.org` !