TECHNISCHE UNIVERSITÄT DRESDEN

RULE-BASED PARADIGMS IN KNOWLEDGE REPRESENTATION

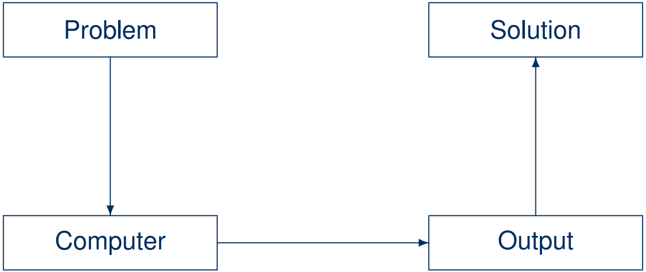
Seminar-Session 2: Declarative Problem Solving with Rules

Stefan Ellmauthaler

TU Dresden, 20th October 2021

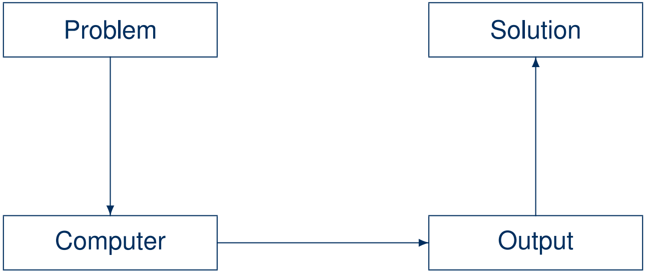
Outline

Computer Science



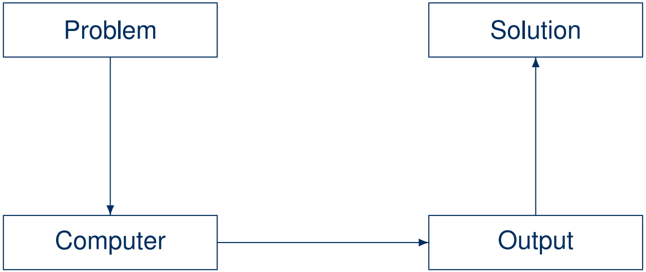
Computer Science

What is the problem? versus **What solves the problem?**



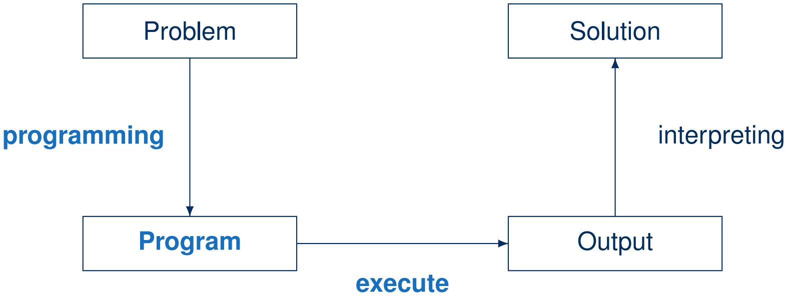
Traditional Programming

What is the problem? versus **What solves the problem?**



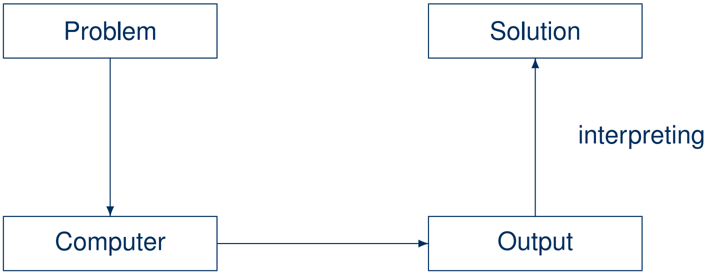
Traditional Programming

What is the problem? versus **What solves the problem?**



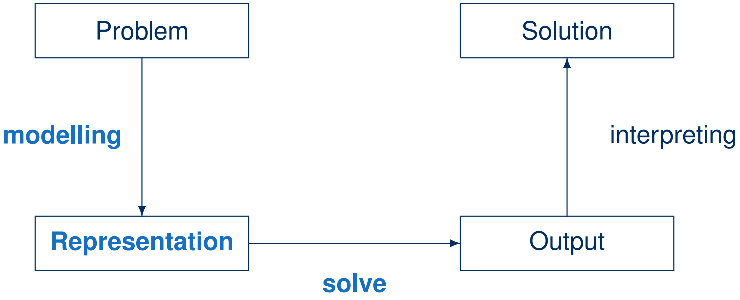
Declarative Problem Solving

What is the problem? versus **What solves the problem?**

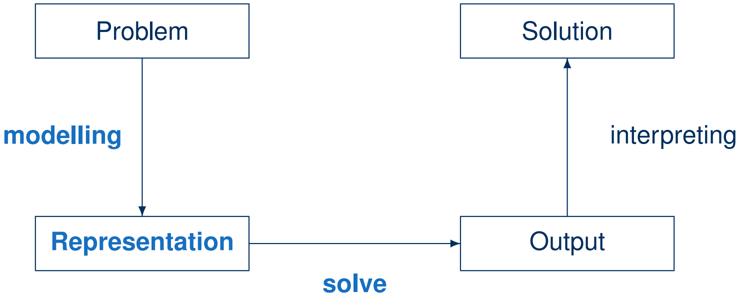


Declarative Problem Solving

What is the problem? versus **What solves the problem?**



Declarative Problem Solving



Why Declarative Rule-Based Problem Solving?

- Model problems . . .
  and use generalised and generic Algorithms to solve them
- Based on Logic and Computer Science
- Sub-field of Knowledge Representation
- Basic Idea (placatory):
  Human describes Problem, Computer solves Problem
- Rules are an intuitive way model problems

Definite Logic Programs: Syntax

Definition 2.1: Syntax of rules:

$$A \leftarrow B_1, \ldots, B_n$$

where A and the B_i are ground atoms.

- A is called head
- $B_1, \ldots, B_n$ is the body of the rule
- facts are rules where $n = 0$; "$\leftarrow$" may be omitted

Definite Logic Programs: Semantics

Definition 2.2: Rule derivation.
Let P a definite logic program and $R = (r_1, \ldots, r_n)$ be a sequence of rules in P such that

- each atom in the body of a rule r_i is a head of a rule r_j, where $j < i$.

$D_A = \{head(r) \mid r \in R\}$ is a derivation for an atom A on P iff $A \in D_A$.

Definite Logic Programs: Semantics

Definition 2.2: Rule derivation.
Let P a definite logic program and $R = (r_1, \ldots, r_n)$ be a sequence of rules in P such that

- each atom in the body of a rule r_i is a head of a rule r_j, where $j < i$.

$D_A = \{head(r) \mid r \in R\}$ is a derivation for an atom A on P iff $A \in D_A$.

Definition 2.3: Closed and Ground sets of Atoms.
Let S be a set of atoms, P a definite program.

- S is closed under P iff $A \in S$ whenever $A \leftarrow B_1, \ldots, B_n \in P$ and $\{B_1, \ldots, B_n\} \subseteq S$.
- S is grounded in P iff $A \in S$ implies there is a derivation for A from P.

Definite Logic Programs: Semantics

Definition 2.2: Rule derivation.
Let P a definite logic program and $R = (r_1, \ldots, r_n)$ be a sequence of rules in P such that

- each atom in the body of a rule r_i is a head of a rule r_j, where $j < i$.

$D_A = \{head(r) \mid r \in R\}$ is a derivation for an atom A on P iff $A \in D_A$.

Definition 2.3: Closed and Ground sets of Atoms.
Let S be a set of atoms, P a definite program.

- S is closed under P iff $A \in S$ whenever $A \leftarrow B_1, \ldots, B_n \in P$ and $\{B_1, \ldots, B_n\} \subseteq S$.
- S is grounded in P iff $A \in S$ implies there is a derivation for A from P.

We call S a consequence of P if it is closed and grounded in P, denoted by $Cn(P)$.

Definite logic Programs: Remarks

$Cn(P)$ is equivalent to

Definite logic Programs: Remarks

$Cn(P)$ is equivalent to

- the smallest set of atoms wich is closed under P and
- the minimal model of P, where
  $\leftarrow$ is read as implication and "," as the logical conjunction.
- the maximal set of atoms which are ground with respect to one (exhaustive) derivation from P

Definite logic Programs: Remarks

$Cn(P)$ is equivalent to

- the smallest set of atoms wich is closed under P and
- the minimal model of P, where
  $\leftarrow$ is read as implication and "," as the logical conjunction.
- the maximal set of atoms which are ground with respect to one (exhaustive) derivation from P

Further logical remarks:

- each rule is a definite clause
  - definite clauses are disjunctions with exactly one positive atom:

$$a_0 \vee \neg a_1, \vee \ldots \vee \neg a_m$$

  - a set of definite clauses has a unique smallest model

Definite logic Programs: Remarks

$Cn(P)$ is equivalent to

- the smallest set of atoms wich is closed under P and
- the minimal model of P, where
  $\leftarrow$ is read as implication and "," as the logical conjunction.
- the maximal set of atoms which are ground with respect to one (exhaustive) derivation from P

Further logical remarks:

- each rule is a definite clause
  - definite clauses are disjunctions with exactly one positive atom:

$$a_0 \vee \neg a_1, \vee \ldots \vee \neg a_m$$

  - a set of definite clauses has a unique smallest model
- horn clauses are clauses with at most one positive atom
  - every definite clause is a horn clause
  - a set of horn clauses has a unique smallest model or none

Definite Logic Programs

Example 2.4: A definite LP:

$a \leftarrow b.$

$b \leftarrow b.$

$c \leftarrow a.$

$c \leftarrow d.$

$d.$

$e \leftarrow a, b, c.$

$e \leftarrow c, d.$

Definite Logic Programs

Example 2.4: A definite LP:

a ←b.
b ←b.
c ←a.
c ←d.
d.
e ←a, b, c.
e ←c, d.

Example 2.5: Solution:

$$\{c, d, e\}$$

Considering Variables

So far, we only considered constant atoms, but how to handle variables?

Considering Variables

So far, we only considered constant atoms, but how to handle variables?

How is it handled in predicate logic?

Considering Variables

So far, we only considered constant atoms, but how to handle variables?

How is it handled in predicate logic? $\rightarrow$ reminder on predicate logic

Interlude: Syntax of Terms in Predicate Logic

Definition 2.6: Let $\mathcal{F}$ be a set of function symbols (with arity), $\mathcal{V}$ a set of variable symbols. The set $\mathcal{T}_{\mathcal{F},\mathcal{V}}$ of **terms over $\mathcal{F}$ and $\mathcal{V}$** is the $\subset$-minimal set, such that:

- $\mathcal{V} \subseteq \mathcal{T}_{\mathcal{F},\mathcal{V}}$
- $t_1, \ldots, t_n \in \mathcal{T}_{\mathcal{F},\mathcal{V}}$ and $f/n \in \mathcal{F}$ implies $f(t_1, \ldots, t_n) \in \mathcal{T}_{\mathcal{F},\mathcal{V}}$.

Interlude: Syntax of Terms in Predicate Logic

Definition 2.6: Let $\mathcal{F}$ be a set of function symbols (with arity), $\mathcal{V}$ a set of variable symbols. The set $\mathcal{T}_{\mathcal{F},\mathcal{V}}$ of **terms over $\mathcal{F}$ and $\mathcal{V}$** is the $\subset$-minimal set, such that:

- $\mathcal{V} \subseteq \mathcal{T}_{\mathcal{F},\mathcal{V}}$
- $t_1, \ldots, t_n \in \mathcal{T}_{\mathcal{F},\mathcal{V}}$ and $f/n \in \mathcal{F}$ implies $f(t_1, \ldots, t_n) \in \mathcal{T}_{\mathcal{F},\mathcal{V}}$.

Example 2.7: $\mathcal{F} = \{1, 0, e, \pi, +, -\}$, $\mathcal{V} = \{X, Y, Z\}$

- We write $+/2 \in \mathcal{F}$ and $-/2 \in \mathcal{F}$
- Terms: $+(1, 0) = 1 + 0$, $-(\pi, +(e, 1)) = \pi - (e + 1)$, $+(X, -(\pi, Y)) = X + (\pi - Y)$
- no Terms: $\cdot(\pi, X) = \pi \cdot X$, $+(2, 2) = 2 + 2$, $-(\pi, 0, e)$

Interlude: Syntax of Atoms in Predicate Logic

Definition 2.8: Let $\mathcal{F}$ be a set of function symbols. The set $\mathcal{T}$ of **variable-free terms of** $\mathcal{F}$ is the $\subset$-minimal set, such that:
$t_1, \ldots, t_n \in \mathcal{T}$ and $f/n \in \mathcal{F}$ implies $f(t_1, \ldots, t_n) \in \mathcal{T}$.

Interlude: Syntax of Atoms in Predicate Logic

Definition 2.8: Let $\mathcal{F}$ be a set of function symbols. The set $\mathcal{T}$ of **variable-free terms of** $\mathcal{F}$ is the $\subset$-minimal set, such that:
$t_1, \ldots, t_n \in \mathcal{T}$ and $f/n \in \mathcal{F}$ implies $f(t_1, \ldots, t_n) \in \mathcal{T}$.

Example 2.9: $\mathcal{F} = \{ a/0, f/1 \} \rightsquigarrow \mathcal{T} = \{ a, f(a), f(f(a)), \ldots \}$; $\mathcal{F} = \{ g/1 \} \rightsquigarrow \mathcal{T} = \emptyset$.

Interlude: Syntax of Atoms in Predicate Logic

Definition 2.8: Let $\mathcal{F}$ be a set of function symbols. The set $\mathcal{T}$ of **variable-free terms of** $\mathcal{F}$ is the $\subset$-minimal set, such that:
$t_1, \ldots, t_n \in \mathcal{T}$ and $f/n \in \mathcal{F}$ implies $f(t_1, \ldots, t_n) \in \mathcal{T}$.

Example 2.9: $\mathcal{F} = \{ a/0, f/1 \} \rightsquigarrow \mathcal{T} = \{ a, f(a), f(f(a)), \ldots \}$; $\mathcal{F} = \{ g/1 \} \rightsquigarrow \mathcal{T} = \emptyset$.

Definition 2.10: Let $\mathcal{P}$ a set of predicate symbols, $\mathcal{F}$ a set of function symbols, $\mathcal{V}$ a set of variable symbols. The set $\mathcal{A}$ of **atoms over $\mathcal{P}$, $\mathcal{F}$, and $\mathcal{V}$** is the $\subseteq$-minimal set, such that:
$t_1, \ldots, t_n \in \mathcal{T}_{\mathcal{F},\mathcal{V}}$ and $p/n \in \mathcal{P}$ implies $p(t_1, \ldots, t_n) \in \mathcal{A}$.

Interlude: Syntax of Atoms in Predicate Logic

Definition 2.8: Let $\mathcal{F}$ be a set of function symbols. The set $\mathcal{T}$ of **variable-free terms of** $\mathcal{F}$ is the $\subset$-minimal set, such that:
$t_1, \ldots, t_n \in \mathcal{T}$ and $f/n \in \mathcal{F}$ implies $f(t_1, \ldots, t_n) \in \mathcal{T}$.

Example 2.9: $\mathcal{F} = \{a/0, f/1\} \rightsquigarrow \mathcal{T} = \{a, f(a), f(f(a)), \ldots\}$; $\mathcal{F} = \{g/1\} \rightsquigarrow \mathcal{T} = \emptyset$.

Definition 2.10: Let $\mathcal{P}$ a set of predicate symbols, $\mathcal{F}$ a set of function symbols, $\mathcal{V}$ a set of variable symbols. The set $\mathcal{A}$ of **atoms over** $\mathcal{P}$**,** $\mathcal{F}$**, and** $\mathcal{V}$ is the $\subseteq$-minimal set, such that:
$t_1, \ldots, t_n \in \mathcal{T}_{\mathcal{F},\mathcal{V}}$ and $p/n \in \mathcal{P}$ implies $p(t_1, \ldots, t_n) \in \mathcal{A}$.

Example 2.11: $\mathcal{P} = \{\text{even}/1, \leq/2, p/3\}$, $\mathcal{F} = \{1, 0, e, \pi, +/2, -/2\}$, $\mathcal{V} = \{X, Y, Z\}$.

- Atoms: $\text{even}(0)$, $\text{even}(1)$, $\leq(X, \pi)$, $p(1, +(1, Y), Y)$
- no Atoms: $\text{odd}(1)$, $\leq(1, 1, 1)$, $\text{even}(2)$

Considering Variables

So far, we only considered constant atoms, but how to handle variables?

How is it handled in predicate logic?
Atoms over predicates, function-symbols, and variables.

Considering Variables

So far, we only considered constant atoms, but how to handle variables?

How is it handled in predicate logic?
Atoms over predicates, function-symbols, and variables.

Issues:

- n-ary function symbols allow for infinitely long terms

Considering Variables

So far, we only considered constant atoms, but how to handle variables?

How is it handled in predicate logic?
Atoms over predicates, function-symbols, and variables.

Issues:

- n-ary function symbols allow for infinitely long terms

Solution:

- use only 0 arity function symbols (i.e. constant symbols)

Considering Variables

So far, we only considered constant atoms, but how to handle variables?

How is it handled in predicate logic?
Atoms over predicates, function-symbols, and variables.

Issues:

- n-ary function symbols allow for infinitely long terms

Solution:

- use only 0 arity function symbols (i.e. constant symbols)

How to check rules with variables against definite logic programs?

Variables - How to Handle Them?

Substitute the variables by terms.

A substitution to variable-free terms is called a ground-instantiation.

Two options:

- Construct an exhaustive derivation and find one matching substitution for each rule
- Create a ground instantiation of all variables in all rules . . .
  then solve the variable free set of rules as before

Derivation + Local Match

The basic concept of **Datalog**

- find a homomorphism to map variables in rule to be an applicable rule with ground atoms
- apply rules as long as possible
  semi-naive evaluation
- distinction between extensional and intentional database

Ground + Solve

The basic concept of **Answer Set Programming**[1]

[1] Example taken from Torsten Schaubs teaching slides on “Answer Set Solving in Practice”

Ground + Solve

The basic concept of **Answer Set Programming**[1]

Example 2.12: $P = \{\, r(a,b) \leftarrow, r(b,c) \leftarrow, s(X,Y) \leftarrow r(X,Y) \,\}$

$T = \{\, a, b, c \,\}$

$$A = \left\{ \begin{array}{l} r(a,a), r(a,b), r(a,c), r(b,a), r(b,b), r(b,c), r(c,a), r(c,b), r(c,c) \\ s(a,a), s(a,b), s(a,c), s(b,a), s(b,b), s(b,c), s(c,a), s(c,b), s(c,c) \end{array} \right\}$$

$g(P) =$

$\{r(a,b) \leftarrow, r(b,c) \leftarrow$

$s(a,a) \leftarrow r(a,a), s(a,b) \leftarrow r(a,b), s(a,c) \leftarrow r(a,c)$

$s(b,a) \leftarrow r(b,a), s(b,b) \leftarrow r(b,b), s(b,c) \leftarrow r(b,c)$

$s(c,a) \leftarrow r(c,a), s(c,b) \leftarrow r(c,b), s(c,c) \leftarrow r(c,c)\}$

[1] Example taken from Torsten Schaubs teaching slides on “Answer Set Solving in Practice”

Ground + Solve

The basic concept of **Answer Set Programming**[1]

Example 2.12: $P = \{ r(a,b) \leftarrow, r(b,c) \leftarrow, s(X,Y) \leftarrow r(X,Y) \}$

$T = \{ a, b, c \}$

$$A = \left\{ \begin{array}{l} r(a,a), r(a,b), r(a,c), r(b,a), r(b,b), r(b,c), r(c,a), r(c,b), r(c,c) \\ s(a,a), s(a,b), s(a,c), s(b,a), s(b,b), s(b,c), s(c,a), s(c,b), s(c,c) \end{array} \right\}$$

$g(P) =$

$\{r(a,b) \leftarrow, r(b,c) \leftarrow$

$s(a,a) \leftarrow r(a,a), s(a,b) \leftarrow r(a,b), s(a,c) \leftarrow r(a,c)$

$s(b,a) \leftarrow r(b,a), s(b,b) \leftarrow r(b,b), s(b,c) \leftarrow r(b,c)$

$s(c,a) \leftarrow r(c,a), s(c,b) \leftarrow r(c,b), s(c,c) \leftarrow r(c,c)\}$

+ default negation

[1] Example taken from Torsten Schaubs teaching slides on “Answer Set Solving in Practice”

Normal Logic Programs: Syntax

Definition 2.13: Syntax of rules:

$$A \leftarrow B_1, \ldots, B_n, \text{not } C_1, \ldots, \text{not } C_m$$

where A, the B_i and the C_j are ground atoms.

Normal Logic Programs: Answer Set Semantics

Definition 2.14: Let S be a set of atoms, P a normal program.

- S is closed under P iff $A \in S$ whenever
  $A \leftarrow B_1, \ldots, B_n, \text{not } C_1, \ldots, \text{not } C_m \in P$ and
  $B_1, \ldots, B_n \in S$, $C_1, \ldots, C_m \notin S$.
- S is grounded in P iff $A \in S$ implies there is a valid derivation for A from P.

An Answer Set (AS) of P is called a stable model if it is closed and grounded in P. We call $SM(P)$ the set of stable models of P.

Normal Logic Programs: Answer Set Semantics

Definition 2.14: Let S be a set of atoms, P a normal program.

- S is closed under P iff $A \in S$ whenever $A \leftarrow B_1, \ldots, B_n, \text{not } C_1, \ldots, \text{not } C_m \in P$ and $B_1, \ldots, B_n \in S$, $C_1, \ldots, C_m \notin S$.
- S is grounded in P iff $A \in S$ implies there is a valid derivation for A from P.

An Answer Set (AS) of P is called a stable model if it is closed and grounded in P. We call $SM(P)$ the set of stable models of P.

Definition 2.15: Valid derivation

- S defeats $Q \leftarrow B_1, \ldots, B_n, \text{not } C_1, \ldots, \text{not } C_m$ iff $C_j \in S, j \in \{1, \ldots, m\}$
- derivation is valid in S iff it is only based on rules undefeated by S.

Stable Model Semantics: Gelfond-Lifschitz Reduct

Definition 2.16: Let P be a (ground) normal program, S a set of atoms. P^S is the program obtained from P by

- eliminating all rules containing not C where $C \in S$,
- eliminating all negated literals from the remaining rules.

S is an answer set under stable model semantics for P iff $S = Cn(P^S)$.

Stable Model Semantics: Gelfond-Lifschitz Reduct

Definition 2.16: Let P be a (ground) normal program, S a set of atoms. P^S is the program obtained from P by

- eliminating all rules containing not C where $C \in S$,
- eliminating all negated literals from the remaining rules.

S is an answer set under stable model semantics for P iff $S = Cn(P^S)$.

Remarks:

- P^S contains no default-negation
- therefore P^S is a definite logic program
- $Cn(P^S)$ has one unique result
- P can have many (or no) stable models
- a sub(super)set of a stable model cannot be a stable model too

Some Examples on ASP-Programs

$$P_0 = \{a \leftarrow b, c \quad b \leftarrow a, \quad c \leftarrow\}$$

$$P_1 = \{a \leftarrow \text{not } b, \quad b \leftarrow \text{not } a\}$$

$$P_2 = \{b \leftarrow \text{not } c, \quad c \leftarrow \text{not } b\}$$

$$P_3 = \{a \leftarrow \text{not } b, \text{not } c, \quad b \leftarrow \text{not } a, \text{not } c, \quad c \leftarrow \text{not } a, \text{not } b\}$$

$$P_4 = \{a \leftarrow \text{not } b, \quad a \leftarrow \text{not } c, \quad b \leftarrow \text{not } a, \quad b \leftarrow \text{not } c, \quad c \leftarrow \text{not } a, \quad c \leftarrow \text{not } b\}$$

$$Q = \{z \leftarrow \text{not } z\}$$

$$R = \{z \leftarrow a, \text{not } z\}$$

$$S = \{z \leftarrow \text{not } a, \text{not } z\}$$

Further Extensions for Datalog and ASP

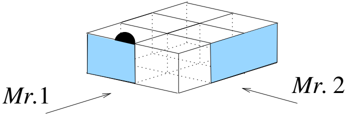
Datalog

- Tuple Generating Dependencies (existentially quantified variables - TGDs)
- Negation
- Constraints
- Various chase-variants with TGDs(e.g. skolem, restricted, core, ...)
- Various evaluations
- ...

Answer Set Programming

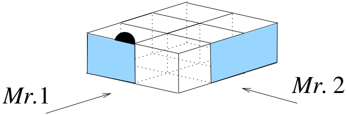
- Disjunctive heads
- Optimisation
- On-Demand Grounding
- Meta-ASP
- ...

Contextual Reasoning
Ghidini & Giunchiglias magic box



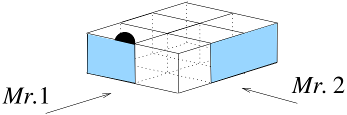
Contextual Reasoning

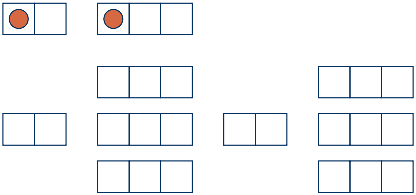
Ghidini & Giunchiglias magic box



Contextual Reasoning

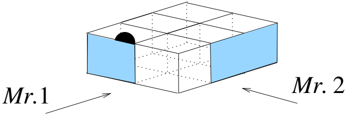
Ghidini & Giunchiglias magic box

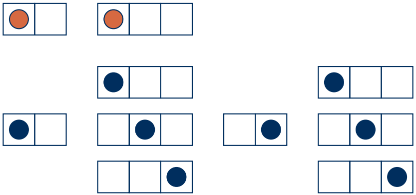




Contextual Reasoning

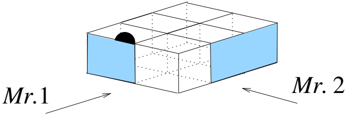
Ghidini & Giunchiglias magic box

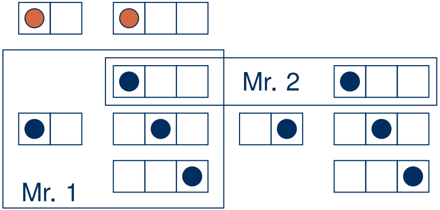




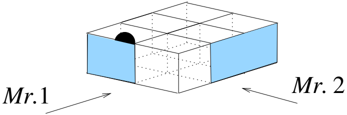
Contextual Reasoning

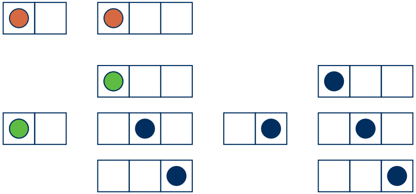
Ghidini & Giunchiglias magic box





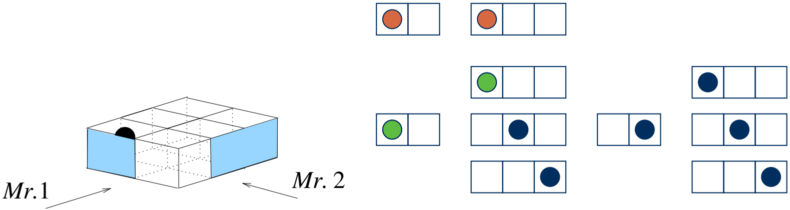
Contextual Reasoning
Ghidini & Giunchiglias magic box





Contextual Reasoning

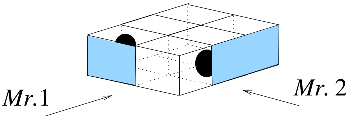
Ghidini & Giunchiglias magic box



- model information
- integrate knowledge bases and context-based information
- synchronise knowledge, reasoning, and conclusions

Contextual Reasoning

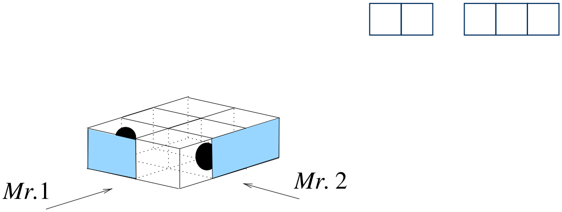
Ghidini & Giunchiglias magic box



- model information
- integrate knowledge bases and context-based information
- synchronise knowledge, reasoning, and conclusions

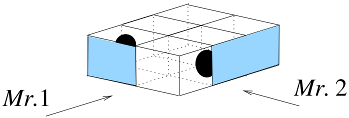
Contextual Reasoning

Ghidini & Giunchiglias magic box



- model information
- integrate knowledge bases and context-based information
- synchronise knowledge, reasoning, and conclusions

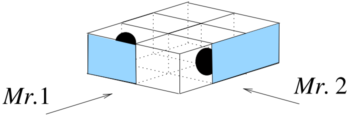
Ghidini & Giunchiglias magic box

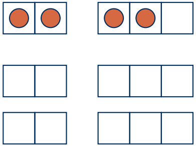


- model information
- integrate knowledge bases and context-based information
- synchronise knowledge, reasoning, and conclusions

Contextual Reasoning

Ghidini & Giunchiglias magic box

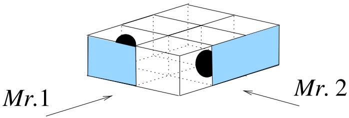


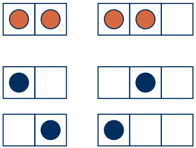


- model information
- integrate knowledge bases and context-based information
- synchronise knowledge, reasoning, and conclusions

Contextual Reasoning

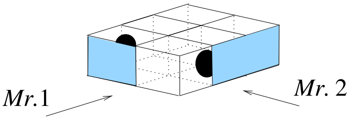
Ghidini & Giunchiglias magic box

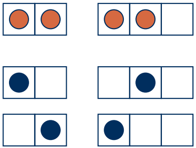




- model information
- integrate knowledge bases and context-based information
- synchronise knowledge, reasoning, and conclusions

Contextual Reasoning
Ghidini & Giunchiglias magic box





- model information
- integrate knowledge bases and context-based information
- synchronise knowledge, reasoning, and conclusions
- handle non-determinism and non-mononotonic behaviour

Possible Topics

- Answer Set Programming
- Datalog
- Distributed reasoning

Next Week . . .

Till next week . . .

have a look at the list and choose your favourite papers

Next Week . . .

- we will fix your topic to one paper
- we will discuss the format of the summary paper