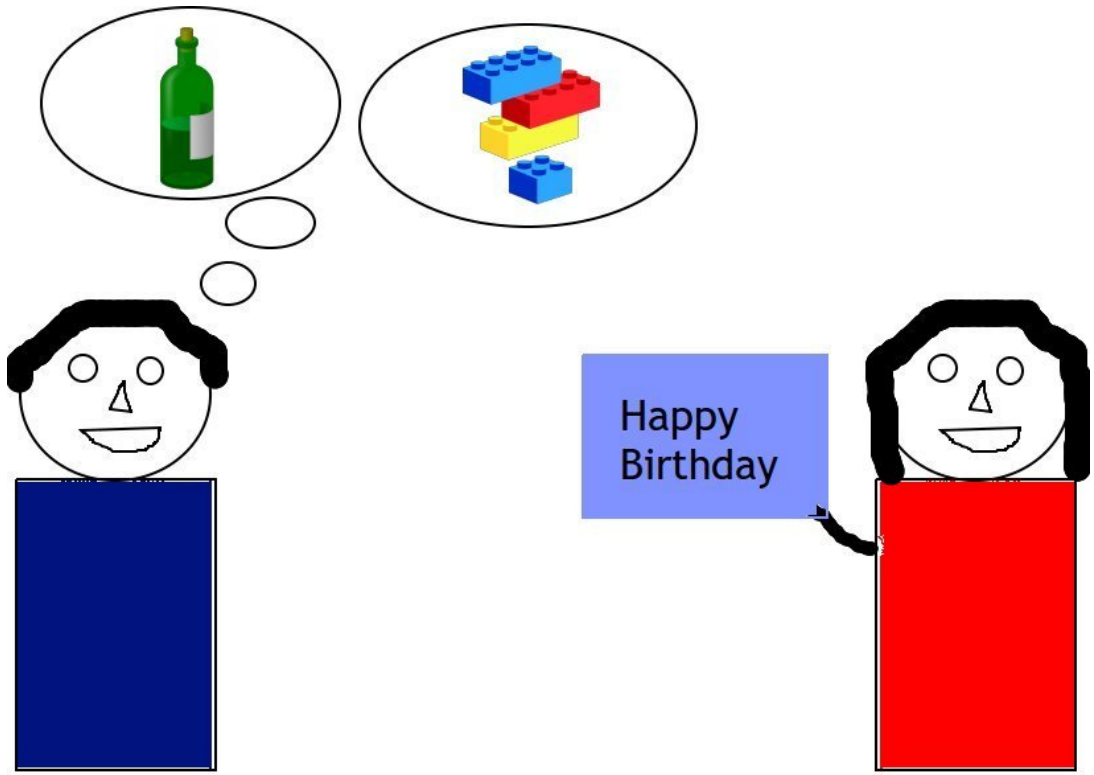
More or Less Tractable Reasoning with Limited Belief

Christoph Schwering

UNSW Sydney

8 May 2018

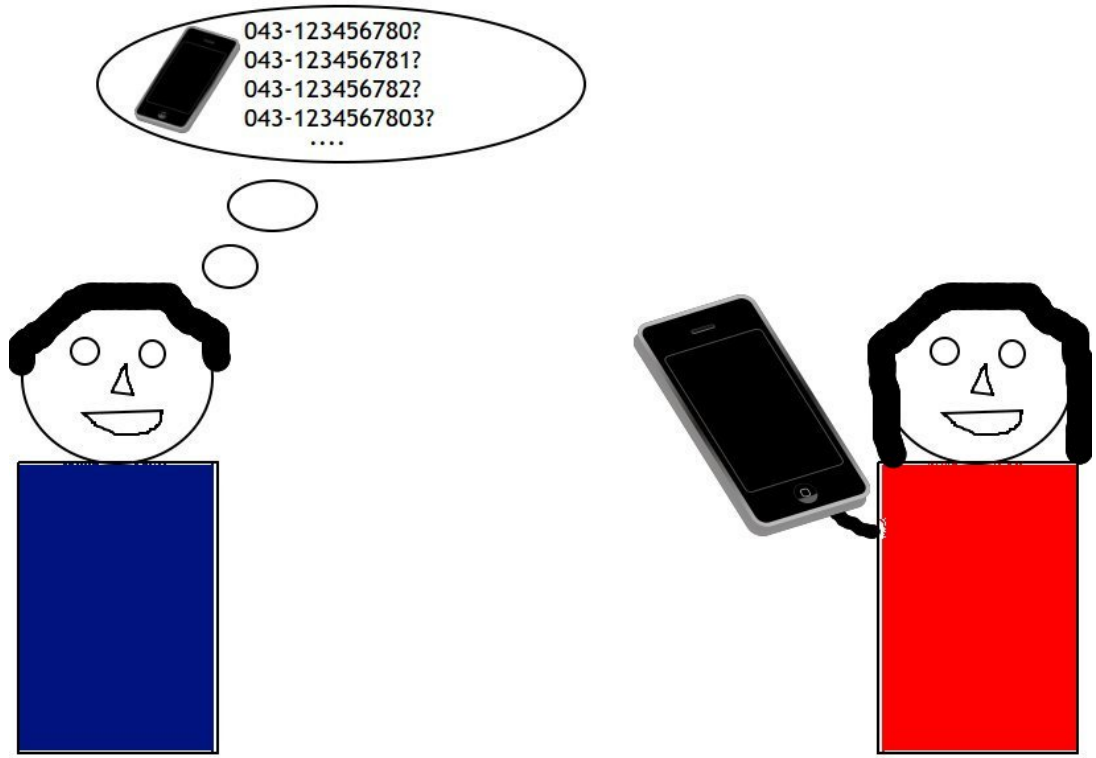
Why reason about knowledge?



You don’t know what’s in the gift box.

You’ll treat it with great care.

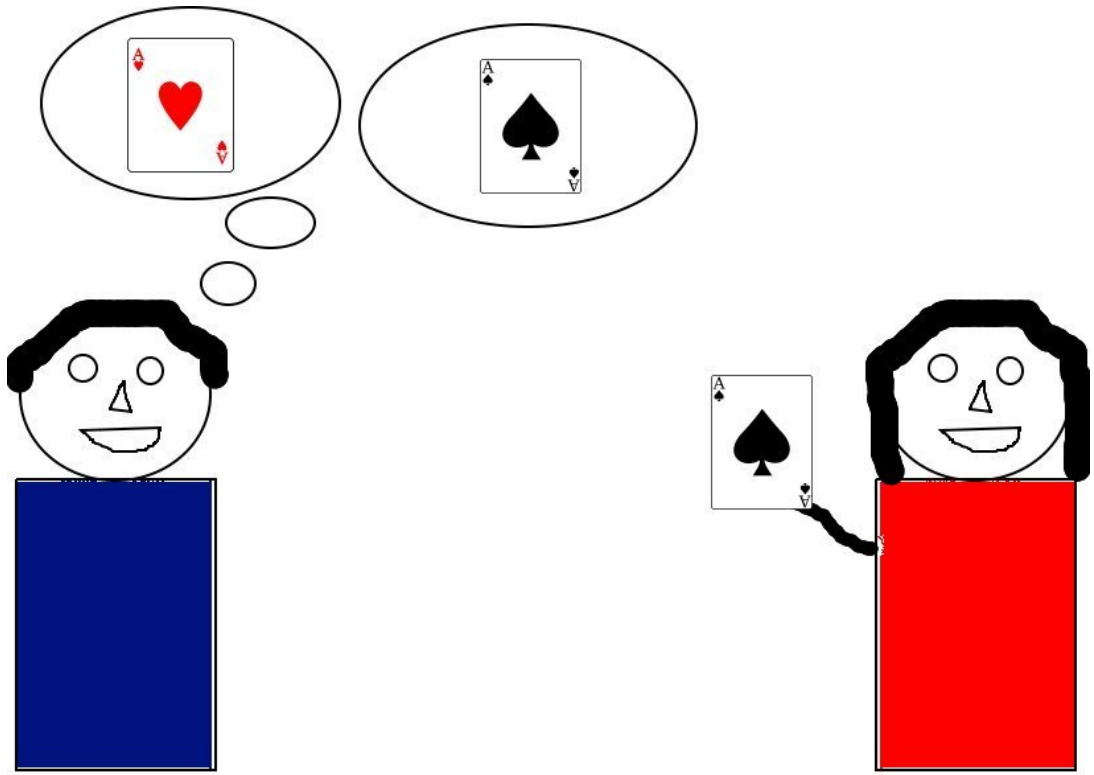
Why reason about knowledge?



You know Jane has a phone, but you don't know her number.

You'll look it up.

Why reason about knowledge?



You know Jane is holding ace of spades *or* ace of hearts, but not which.

You need a strategy that wins in either case.

Why reason about knowledge?

We deal with

- incomplete knowledge and beliefs

all the time.

Why reason about knowledge?

We deal with

- incomplete knowledge and beliefs
- quantification about objects

all the time.

Why reason about knowledge?

We deal with

- incomplete knowledge and beliefs
- quantification about objects
- introspection (know what you know)

all the time.

Why reason about knowledge?

We deal with

- incomplete knowledge and beliefs
- quantification about objects
- introspection (know what you know)
- multiple agents

all the time.

Why reason about knowledge?

We deal with

- incomplete knowledge and beliefs
- quantification about objects
- introspection (know what you know)
- multiple agents
- action and perception

all the time.

Why reason about knowledge?

We deal with

- incomplete knowledge and beliefs
- quantification about objects
- introspection (know what you know)
- multiple agents
- action and perception
- belief revision

all the time.

How can computers do reasoning?

Need a formal model for:

Question: If we know A, do we know B then?

How can computers do reasoning?

Need a formal model for:

Question: If we know A, do we know B then?

In logic: Does every interpretation satisfy “A implies B”?

How can computers do reasoning?

Need a formal model for:

Question: If we know A, do we know B then?

In logic: Does every interpretation satisfy “A implies B”?

Is “A implies B” valid?

How can computers do reasoning?

Need a formal model for:

Question: If we know A, do we know B then?

In logic: Does every interpretation satisfy “A implies B”?

Is “A implies B” valid?

Tasks:

1. Find an expressive language
2. Manage computational complexity

A basic propositional language

- Propositions
- not α
- α or β
- α and β

A basic propositional language

- Propositions
- not α
- α or β

A basic propositional language

- Propositions: $P_1, P_2, P_3, \ldots$
- not α: $\neg\alpha$
- α or β: $(\alpha \vee \beta)$

A basic propositional language

A world w maps propositions to $\{0, 1\}$:

- $w \models P$ iff $w(P) = 1$
- $w \models \neg\alpha$ iff $w \not\models \alpha$
- $w \models (\alpha \vee \beta)$ iff $w \models \alpha$ or $w \models \beta$

A basic propositional language

A world w maps propositions to $\{0, 1\}$:

- $w \models P$ iff $w(P) = 1$
- $w \models \neg\alpha$ iff $w \not\models \alpha$
- $w \models (\alpha \vee \beta)$ iff $w \models \alpha$ or $w \models \beta$

- α is satisfiable iff $w \models \alpha$ for some w
- α is valid iff $w \models \alpha$ for all w

A basic propositional language

A world w maps propositions to $\{0, 1\}$:

- $w \models P$ iff $w(P) = 1$
- $w \models \neg\alpha$ iff $w \not\models \alpha$
- $w \models (\alpha \vee \beta)$ iff $w \models \alpha$ or $w \models \beta$

- α is satisfiable iff $w \models \alpha$ for some w — NP-hard
- α is valid iff $w \models \alpha$ for all w — co-NP-hard

Reasoning is really hard 

A basic functional language

- Object names: $^{\#}1, ^{\#}2, ^{\#}3, \ldots$
- Functions: $f(n_1, \ldots, n_j)$
- not α: $\neg\alpha$
- α or β: $(\alpha \vee \beta)$

A basic functional language

A world w maps functions to names:

- $w \models n_1 = n_2$ iff $n_1 = n_2$
- $w \models f(\vec{n}) = n$ iff $w(f(\vec{n})) = n$
- $w \models \neg\alpha$ iff $w \not\models \alpha$
- $w \models (\alpha \vee \beta)$ iff $w \models \alpha$ or $w \models \beta$

A basic functional language

A world w maps functions to names:

- $w \models n_1 = n_2$ iff $n_1 = n_2$
- $w \models f(\vec{n}) = n$ iff $w(f(\vec{n})) = n$
- $w \models \neg\alpha$ iff $w \not\models \alpha$
- $w \models (\alpha \vee \beta)$ iff $w \models \alpha$ or $w \models \beta$

- α is satisfiable iff $w \models \alpha$ for some w NP-hard
- α is valid iff $w \models \alpha$ for all w co-NP-hard

Reasoning is really hard 

A basic first-order language

- Object names: $^{\#}1, ^{\#}2, ^{\#}3, \ldots$
- Functions: $f(n_1, \ldots, n_j)$
- not α: $\neg\alpha$
- α or β: $(\alpha \vee \beta)$
- for some x, α: $\exists x\, \alpha$

A basic first-order language

A world w maps functions to names:

- $w \models n_1 = n_2$ iff $n_1 = n_2$
- $w \models f(\vec{n}) = n$ iff $w(f(\vec{n})) = n$
- $w \models \neg\alpha$ iff $w \not\models \alpha$
- $w \models (\alpha \vee \beta)$ iff $w \models \alpha$ or $w \models \beta$
- $w \models \exists x\, \alpha$ iff $w \models \alpha^x_n$ for some name n

A basic first-order language

A world w maps functions to names:

- $w \models n_1 = n_2$ iff $n_1 = n_2$
- $w \models f(\vec{n}) = n$ iff $w(f(\vec{n})) = n$
- $w \models \neg\alpha$ iff $w \not\models \alpha$
- $w \models (\alpha \vee \beta)$ iff $w \models \alpha$ or $w \models \beta$
- $w \models \exists x\, \alpha$ iff $w \models \alpha^x_n$ for some name n

- α is satisfiable iff $w \models \alpha$ for some w — undecidable
- α is valid iff $w \models \alpha$ for all w — semidecidable

Reasoning is really, really hard 

Is reasoning really that hard?

Pigeonhole principle: $k+1$ pigeons cannot be put in k holes

$$k = 1: \quad (h_1 = {}^{\#}p_1) \wedge (h_1 = {}^{\#}p_2)$$

Is reasoning really that hard?

Pigeonhole principle: $k+1$ pigeons cannot be put in k holes

$$
\begin{aligned}
k=2: \quad & (h_1 = {}^{\#}p_1 \vee h_2 = {}^{\#}p_1) \wedge \\
& (h_1 = {}^{\#}p_2 \vee h_2 = {}^{\#}p_2) \wedge \\
& (h_1 = {}^{\#}p_3 \vee h_2 = {}^{\#}p_3)
\end{aligned}
$$

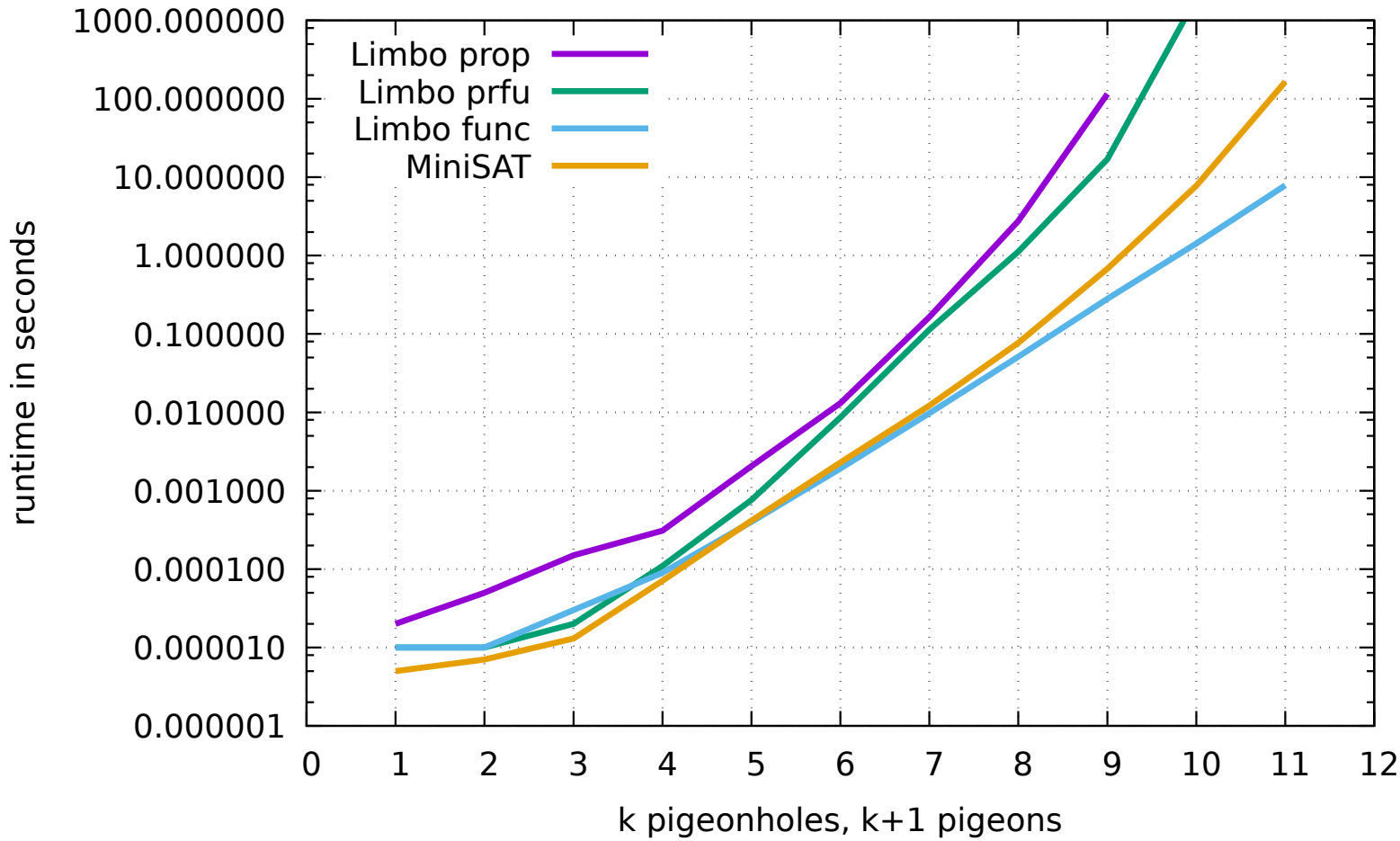
Is reasoning really that hard?

Pigeonhole principle: $k+1$ pigeons cannot be put in k holes

$$
\begin{aligned}
k=3: \quad & (h_1 = {}^{\#}p_1 \vee h_2 = {}^{\#}p_1 \vee h_3 = {}^{\#}p_1) \wedge \\
& (h_1 = {}^{\#}p_2 \vee h_2 = {}^{\#}p_2 \vee h_3 = {}^{\#}p_2) \wedge \\
& (h_1 = {}^{\#}p_3 \vee h_2 = {}^{\#}p_3 \vee h_3 = {}^{\#}p_3) \wedge \\
& (h_1 = {}^{\#}p_4 \vee h_2 = {}^{\#}p_4 \vee h_3 = {}^{\#}p_4)
\end{aligned}
$$

Is reasoning really that hard?

Pigeonhole principle: $k + 1$ pigeons cannot be put in k holes



How to deal with the complexity? – Syntactic approach

- Quantifier-prefix classes
- Description logics
- Levesque's proper knowledge bases

How to deal with the complexity? – Naive approach

1. Use an off-the-shelf solver as blackbox
   - SAT solver
   - Theorem prover
2. Introduce a computational budget
   - Time
   - Memory
3. Terminate once budget is spent

How to deal with the complexity? – Naive approach

1. Use an off-the-shelf solver as blackbox
   - SAT solver
   - Theorem prover
2. Introduce a computational budget
   - Time
   - Memory
3. Terminate once budget is spent

Problem:

- Behaviour not explainable because of unknown semantics
- Possibly nondeterministic

Need a more principled approach

Why is reasoning hard? Why can humans still do it?

Using classical logic to model knowledge has some implications:

- All tautologies are known
  If α is valid, then α is known

Why is reasoning hard? Why can humans still do it?

Using classical logic to model knowledge has some implications:

- All tautologies are known
  If α is valid, then α is known
- Knowledge is closed under logical consequence
  If α and $(\alpha \rightarrow \beta)$ are known, then β is known

Why is reasoning hard? Why can humans still do it?

Using classical logic to model knowledge has some implications:

- All tautologies are known
  If α is valid, then α is known
- Knowledge is closed under logical consequence
  If α and $(\alpha \rightarrow \beta)$ are known, then β is known
- Knowledge is closed under equivalence
  If α is known and β is equivalent to α, then β is known

Why is reasoning hard? Why can humans still do it?

Using classical logic to model knowledge has some implications:

- All tautologies are known
  If α is valid, then α is known
- Knowledge is closed under logical consequence
  If α and $(\alpha \rightarrow \beta)$ are known, then β is known
- Knowledge is closed under equivalence
  If α is known and β is equivalent to α, then β is known
- Inconsistent knowledge implies everything

Why is reasoning hard? Why can humans still do it?

Using classical logic to model knowledge has some implications:

- All tautologies are known
  If α is valid, then α is known
- Knowledge is closed under logical consequence
  If α and $(\alpha \to \beta)$ are known, then β is known
- Knowledge is closed under equivalence
  If α is known and β is equivalent to α, then β is known
- Inconsistent knowledge implies everything

This is called **logical omniscience**.

Why is reasoning hard? Why can humans still do it?

Using classical logic to model knowledge has some implications:

- All tautologies are known
  If α is valid, then α is known
- Knowledge is closed under logical consequence
  If α and $(\alpha \to \beta)$ are known, then β is known
- Knowledge is closed under equivalence
  If α is known and β is equivalent to α, then β is known
- Inconsistent knowledge implies everything

This is called **logical omniscience**.

Humans are not omniscient! But we can think harder or not.

How to deal with the complexity? – Semantic approach

The Logic of Limited Belief

- Very expressive language
  - First-order logic
  - Modal operators for knowledge
  - Modal operators for action

How to deal with the complexity? – Semantic approach

The Logic of Limited Belief

- Very expressive language
  - First-order logic
  - Modal operators for knowledge
  - Modal operators for action
- Computational budget
  - How hard should the computer think?
  - Explicit beliefs are for free
  - Implicit beliefs require reasoning
  - Hypothesis: Cheap inferences are often good enough

How to deal with the complexity? – Semantic approach

The Logic of Limited Belief

- Very expressive language
  - First-order logic
  - Modal operators for knowledge
  - Modal operators for action
- Computational budget
  - How hard should the computer think?
  - Explicit beliefs are for free
  - Implicit beliefs require reasoning
  - Hypothesis: Cheap inferences are often good enough
- Desired properties:
  - Propositional: tractable / first-order: decidable
  - Relationship to classical logic: sound but incomplete
  - Results should be explainable

Example: Minesweeper

`http://www.cse.unsw.edu.au/~cschwering/limbo/minesweeper-js.html`

size medium, seed 5, level 0 and 1

An alternative model of knowledge

- Classical model of knowledge:
  - Set of possible worlds $\{w_1, w_2, \ldots\}$ (α is known iff all $w_i \models \alpha$)
  - World w_i is a conjunction $\ell_1 \wedge \ell_2 \wedge \ldots$
  - World w_i is one possibility
  - Hence: Disjunction of conjunctions (DNF)

An alternative model of knowledge

- Classical model of knowledge:
  - Set of possible worlds $\{w_1, w_2, \ldots\}$ (α is known iff all $w_i \models \alpha$)
  - World w_i is a conjunction $\ell_1 \wedge \ell_2 \wedge \ldots$
  - World w_i is one possibility
  - Hence: Disjunction of conjunctions (DNF)

- Alternative model: CNF instead of DNF!
  - Conjunction of disjunctions (CNF)
  - Clause c_j is a disjunction $\ell_1 \vee \ell_2 \vee \ldots$
  - Setup s is a set of clauses $\{c_1, c_2, \ldots\}$

Semantics of limited belief

A setup s is a set of ground clauses
A clause c is a disjunction of literals

- $s \not\approx c$ iff $c \in s$

Semantics of limited belief

A setup s is a set of ground clauses
A clause c is a disjunction of literals

- $s \not\approx c$ iff $c \in s$
- $s \not\approx (\alpha \vee \beta)$ iff $s \not\approx \alpha$ or $s \not\approx \beta$ if $(\alpha \vee \beta)$ is not a clause
- $s \not\approx \neg(\alpha \vee \beta)$ iff $s \not\approx \neg\alpha$ and $s \not\approx \neg\beta$
- $s \not\approx \neg\neg\alpha$ iff $s \not\approx \alpha$

Semantics of limited belief

A setup s is a set of ground clauses
A clause c is a disjunction of literals

- $s \models\!\!\approx c$ iff $c \in s$
- $s \models\!\!\approx (\alpha \vee \beta)$ iff $s \models\!\!\approx \alpha$ or $s \models\!\!\approx \beta$ if $(\alpha \vee \beta)$ is not a clause
- $s \models\!\!\approx \neg(\alpha \vee \beta)$ iff $s \models\!\!\approx \neg\alpha$ and $s \models\!\!\approx \neg\beta$
- $s \models\!\!\approx \neg\neg\alpha$ iff $s \models\!\!\approx \alpha$
- $s \models\!\!\approx \exists x\, \alpha$ iff $s \models\!\!\approx \alpha^x_n$ for some name n
- $s \models\!\!\approx \neg\exists x\, \alpha$ iff $s \models\!\!\approx \neg\alpha^x_n$ for all names n

Semantics of limited belief

A setup s is a set of ground clauses
A clause c is a disjunction of literals

- $s \models\!\!\approx c$ iff $c \in s$
- $s \models\!\!\approx (\alpha \vee \beta)$ iff $s \models\!\!\approx \alpha$ or $s \models\!\!\approx \beta$ if $(\alpha \vee \beta)$ is not a clause
- $s \models\!\!\approx \neg(\alpha \vee \beta)$ iff $s \models\!\!\approx \neg\alpha$ and $s \models\!\!\approx \neg\beta$
- $s \models\!\!\approx \neg\neg\alpha$ iff $s \models\!\!\approx \alpha$
- $s \models\!\!\approx \exists x\, \alpha$ iff $s \models\!\!\approx \alpha^x_n$ for some name n
- $s \models\!\!\approx \neg\exists x\, \alpha$ iff $s \models\!\!\approx \neg\alpha^x_n$ for all names n
- $s \models\!\!\approx \mathbf{K}_0\, \alpha$ iff $s \models\!\!\approx \alpha$
- $s \models\!\!\approx \mathbf{K}_{k+1}\, \alpha$ iff for some ℓ, $s \cup \{\ell\} \models\!\!\approx \mathbf{K}_k\, \alpha$ and $s \cup \{\bar{\ell}\} \models\!\!\approx \mathbf{K}_k\, \alpha$

Semantics of limited belief

A setup s is a set of ground clauses
A clause c is a disjunction of literals

- $s \mid\!\approx c$ iff $c \in s$
- $s \mid\!\approx (\alpha \vee \beta)$ iff $s \mid\!\approx \alpha$ or $s \mid\!\approx \beta$ if $(\alpha \vee \beta)$ is not a clause
- $s \mid\!\approx \neg(\alpha \vee \beta)$ iff $s \mid\!\approx \neg\alpha$ and $s \mid\!\approx \neg\beta$
- $s \mid\!\approx \neg\neg\alpha$ iff $s \mid\!\approx \alpha$
- $s \mid\!\approx \exists x\, \alpha$ iff $s \mid\!\approx \alpha^x_n$ for some name n
- $s \mid\!\approx \neg\exists x\, \alpha$ iff $s \mid\!\approx \neg\alpha^x_n$ for all names n
- $s \mid\!\approx \mathbf{K}_0\, \alpha$ iff $s \mid\!\approx \alpha$
- $s \mid\!\approx \mathbf{K}_{k+1}\, \alpha$ iff for some ℓ, $s \cup \{\ell\} \mid\!\approx \mathbf{K}_k\, \alpha$ and $s \cup \{\bar{\ell}\} \mid\!\approx \mathbf{K}_k\, \alpha$

Example: $\{\} \not\mid\!\approx \mathbf{K}_0\, (P \vee \neg P)$ $\{\} \mid\!\approx \mathbf{K}_1\, (P \vee \neg P)$

Subsumption and unit propagation (1/4)

Propositional case:

- c_1 subsumes c_2 iff $c_1 \subseteq c_2$
  - Example: $(P \vee Q)$ subsumes $(P \vee Q \vee R)$

Subsumption and unit propagation (1/4)

Propositional case:

- c_1 subsumes c_2 iff $c_1 \subseteq c_2$
  - Example: $(P \vee Q)$ subsumes $(P \vee Q \vee R)$
- From c and ℓ infer $c \setminus \{\overline{\ell}\}$ by unit propagation
  - Example: $(P \vee Q \vee R)$ and $\neg P$ yields $(Q \vee R)$

Subsumption and unit propagation (1/4)

Propositional case:

- c_1 subsumes c_2 iff $c_1 \subseteq c_2$
  - Example: $(P \vee Q)$ subsumes $(P \vee Q \vee R)$
- From c and ℓ infer $c \setminus \{\bar{\ell}\}$ by unit propagation
  - Example: $(P \vee Q \vee R)$ and $\neg P$ yields $(Q \vee R)$

Close setups under unit propagation and subsumption

Example: $\{(\neg P \vee Q), (P \vee R)\} \mid\!\approx \mathbf{K}_1 (Q \vee R)$

because $\{(\neg P \vee Q), (P \vee R), \quad P, \quad Q\} \mid\!\approx (Q \vee R)$

and $\{(\neg P \vee Q), (P \vee R), \neg P, \quad R\} \mid\!\approx (Q \vee R)$

Subsumption and unit propagation (2/4)

Functional case: $f(\vec{n}) = n$ and $f(\vec{n}) \neq n$ and $n = n'$ and $n \neq n'$

- ✓ $\text{Sally} \neq \text{Frank}$
- ✓ $\text{fatherOf}(\text{Sally}) = \text{Frank}$
- ✗ $\text{motherOf}(\text{fatherOf}(\text{Sally})) = \text{Frank}$
- ✗ $\text{fatherOf}(\text{Sally}) \neq \text{fatherOf}(\text{Sandy})$

} can be flattened

Subsumption and unit propagation (2/4)

Functional case: $f(\vec{n}) = n$ and $f(\vec{n}) \neq n$ and $n = n'$ and $n \neq n'$

- ✓ $\text{Sally} \neq \text{Frank}$
- ✓ $\text{fatherOf}(\text{Sally}) = \text{Frank}$
- ✗ $\text{motherOf}(\text{fatherOf}(\text{Sally})) = \text{Frank}$
- ✗ $\text{fatherOf}(\text{Sally}) \neq \text{fatherOf}(\text{Sandy})$

} can be flattened

Subsumption and unit propagation (2/4)

Functional case: $f(\vec{n}) = n$ and $f(\vec{n}) \neq n$ and $n = n'$ and $n \neq n'$

- What is subsumed by $f(\vec{n}) = n$? $\quad f(\vec{n}) = n \quad f(\vec{n}) \neq n'$
- What is subsumed by $f(\vec{n}) \neq n$? $\quad f(\vec{n}) \neq n$

Subsumption and unit propagation (2/4)

Functional case: $f(\vec{n}) = n$ and $f(\vec{n}) \neq n$ and $n = n'$ and $n \neq n'$

- What is subsumed by $f(\vec{n}) = n$? $\quad f(\vec{n}) = n \quad f(\vec{n}) \neq n'$
- What is subsumed by $f(\vec{n}) \neq n$? $\quad f(\vec{n}) \neq n$

- **Lemma:** $\ell_1 \rightarrow \ell_2$ is valid iff ℓ_1 subsumes ℓ_2
- **Lemma:** $\ell_1 \wedge \ell_2$ is unsatisfiable iff ℓ_1 subsumes $\overline{\ell}_2$

Subsumption and unit propagation (3/4)

Functional case: $f(\vec{n}) = n$ and $f(\vec{n}) \neq n$

- c_1 subsumes c_2 iff every $\ell_1 \in c_1$ subsumes some $\ell_2 \in c_2$
  - $\text{fatherOf}(\text{Sally}) = \text{Frank} \vee \text{fatherOf}(\text{Sally}) = \text{Fred}$
    subsumes
    $\text{fatherOf}(\text{Sally}) \neq \text{Tom}$

Subsumption and unit propagation (3/4)

Functional case: $f(\vec{n}) = n$ and $f(\vec{n}) \neq n$

- c_1 subsumes c_2 iff every $\ell_1 \in c_1$ subsumes some $\ell_2 \in c_2$
  - $\mathrm{fatherOf}(\mathrm{Sally}) = \mathrm{Frank} \vee \mathrm{fatherOf}(\mathrm{Sally}) = \mathrm{Fred}$
    subsumes
    $\mathrm{fatherOf}(\mathrm{Sally}) \neq \mathrm{Tom}$
- From c and ℓ infer $c \setminus \{\ell' \mid \ell \text{ subsumes } \overline{\ell'}\}$ by unit propagation
  - $\mathrm{fatherOf}(\mathrm{Sally}) \neq \mathrm{Frank} \vee \mathrm{rich}(\mathrm{Frank}) = \top$
    and
    $\mathrm{fatherOf}(\mathrm{Sally}) = \mathrm{Frank}$
    yield
    $\mathrm{rich}(\mathrm{Frank}) = \top$

Subsumption and unit propagation (3/4)

Functional case: $f(\vec{n}) = n$ and $f(\vec{n}) \neq n$

- c_1 subsumes c_2 iff every $\ell_1 \in c_1$ subsumes some $\ell_2 \in c_2$
  - $\mathrm{fatherOf}(\mathrm{Sally}) = \mathrm{Frank} \vee \mathrm{fatherOf}(\mathrm{Sally}) = \mathrm{Fred}$
    subsumes
    $\mathrm{fatherOf}(\mathrm{Sally}) \neq \mathrm{Tom}$
- From c and ℓ infer $c \setminus \{\ell' \mid \ell \text{ subsumes } \overline{\ell'}\}$ by unit propagation
  - $\mathrm{fatherOf}(\mathrm{Sally}) \neq \mathrm{Frank} \vee \mathrm{rich}(\mathrm{Frank}) = \top$
    and
    $\mathrm{fatherOf}(\mathrm{Sally}) = \mathrm{Frank}$
    yield
    $\mathrm{rich}(\mathrm{Frank}) = \top$

- **Lemma:** $c_1 \rightarrow c_2$ is valid iff c_1 subsumes c_2
- **Lemma:** $c \wedge \ell \rightarrow c'$ is valid iff c, ℓ yields c' by unit propagation

Subsumption and unit propagation (4/4)

- Propositional subsumption is transitive, anti-symmetric, reflexive

Subsumption and unit propagation (4/4)

- Propositional subsumption is transitive, anti-symmetric, reflexive
- Functional subsumption is not anti-symmetric:
  - $f(\vec{n}) \neq n \vee f(\vec{n}) = n'$
  - $f(\vec{n}) \neq n$

Subsumption and unit propagation (4/4)

- Propositional subsumption is transitive, anti-symmetric, reflexive
- Functional subsumption is not anti-symmetric:
  - $f(\vec{n}) \neq n \vee f(\vec{n}) = n'$
  - $f(\vec{n}) \neq n$
- Normalization $N(c)$ removes subsuming literals from c

Subsumption and unit propagation (4/4)

- Propositional subsumption is transitive, anti-symmetric, reflexive
- Functional subsumption is not anti-symmetric:
  - $f(\vec{n}) \neq n \vee f(\vec{n}) = n' \quad =: \; c$
  - $f(\vec{n}) \neq n \qquad\qquad\quad = \; N(c)$
- Normalization $N(c)$ removes subsuming literals from c

Subsumption and unit propagation (4/4)

- Propositional subsumption is transitive, anti-symmetric, reflexive
- Functional subsumption is not anti-symmetric:
  - $f(\vec{n}) \neq n \vee f(\vec{n}) = n' \;=:\; c$
  - $f(\vec{n}) \neq n \;=\; N(c)$
- Normalization $N(c)$ removes subsuming literals from c

- **Lemma:** c and $N(c)$ are equivalent
- **Lemma:** if $N(c_1), N(c_2)$ sub. one other, then $N(c_1) = N(c_2)$
- **Lemma:** no infinite chains of subsuming clauses:
  every "$N(c_1)$ sub. by $N(c_2)$ sub. by $N(c_3)$ ..." is finite

Semantics of limited belief revisited

A setup is a set of ground clauses closed under

- unit propagation
- subsumption
- normalization

Semantics of limited belief revisited

A setup is a set of ground clauses closed under

- unit propagation
- subsumption
- normalization

- $s \not\approx c$ iff $c \in s$
- $s \not\approx (\alpha \vee \beta)$ iff $s \not\approx \alpha$ or $s \not\approx \beta$ if $(\alpha \vee \beta)$ is not a clause
- $s \not\approx \neg(\alpha \vee \beta)$ iff $s \not\approx \neg\alpha$ and $s \not\approx \neg\beta$
- $s \not\approx \neg\neg\alpha$ iff $s \not\approx \alpha$
- $s \not\approx \exists x\, \alpha$ iff $s \not\approx \alpha^x_n$ for some name n
- $s \not\approx \neg\exists x\, \alpha$ iff $s \not\approx \neg\alpha^x_n$ for all names n
- $s \not\approx \mathbf{K}_0\, \alpha$ iff $s \not\approx \alpha$
- $s \not\approx \mathbf{K}_{k+1}\, \alpha$ iff for some t, for all n, $s \cup \{t = n\} \not\approx \mathbf{K}_k\, \alpha$

Reasoning with limited belief

Question: If we know KB, do we then know α at level k?

Reasoning with limited belief

Question: If we know KB, do we then know α at level k?

- KB conjunction of clauses
  - skolemize existentials: $\forall \vec{x} \exists y \, \mathrm{KB} \;\; \rightarrow \;\; \forall \vec{x} \, \mathrm{KB}^{y}_{f(\vec{x})}$
  - flatten right-hand side functions: $t = f(\vec{n}) \;\; \rightarrow \;\; \forall x \, (f(\vec{n}) \neq x \vee t = x)$
  - flatten nested functions: $f(g(\vec{n})) = t \;\; \rightarrow \;\; \forall x \, (g(\vec{n}) \neq x \vee f(x) = t)$

Reasoning with limited belief

Question: If we know KB, do we then know α at level k?

- KB conjunction of clauses
  - skolemize existentials: $\forall \vec{x} \exists y \, \mathrm{KB} \;\rightarrow\; \forall \vec{x} \, \mathrm{KB}^{y}_{f(\vec{x})}$
  - flatten right-hand side functions: $t = f(\vec{n}) \;\rightarrow\; \forall x \, (f(\vec{n}) \neq x \vee t = x)$
  - flatten nested functions: $f(g(\vec{n})) = t \;\rightarrow\; \forall x \, (g(\vec{n}) \neq x \vee f(x) = t)$
- s = grounding of KB with all names

Reasoning with limited belief

Question: If we know KB, do we then know α at level k?

- KB conjunction of clauses
  - skolemize existentials: $\forall \vec{x} \exists y \, \mathrm{KB} \;\rightarrow\; \forall \vec{x} \, \mathrm{KB}^{y}_{f(\vec{x})}$
  - flatten right-hand side functions: $t = f(\vec{n}) \;\rightarrow\; \forall x \, (f(\vec{n}) \neq x \vee t = x)$
  - flatten nested functions: $f(g(\vec{n})) = t \;\rightarrow\; \forall x \, (g(\vec{n}) \neq x \vee f(x) = t)$
- s = grounding of KB with all names
- $s \mid\!\approx \mathbf{K}_k \, \alpha$

Reasoning with limited belief

Question: If we know KB, do we then know α at level k?

- KB conjunction of clauses
  - skolemize existentials: $\forall \vec{x} \exists y \, \mathrm{KB} \;\rightarrow\; \forall \vec{x} \, \mathrm{KB}^{y}_{f(\vec{x})}$
  - flatten right-hand side functions: $t = f(\vec{n}) \;\rightarrow\; \forall x \, (f(\vec{n}) \neq x \vee t = x)$
  - flatten nested functions: $f(g(\vec{n})) = t \;\rightarrow\; \forall x \, (g(\vec{n}) \neq x \vee f(x) = t)$
- s = grounding of KB with all names
  - Semantics: infinitely many
  - Lemma: Names in KB, α + one per variable suffice
- $s \mid\!\approx \mathbf{K}_k \alpha$
  - Lemma: Names in KB, α + 1 suffice for quantification

Theorem:

- $s \mid\!\approx \mathbf{K}_k \alpha$ is decidable

Reasoning with limited belief

Question: If we know KB, do we then know α at level k?

- KB conjunction of clauses
  - skolemize existentials: $\forall \vec{x} \exists y \, \mathrm{KB} \;\rightarrow\; \forall \vec{x} \, \mathrm{KB}^{y}_{f(\vec{x})}$
  - flatten right-hand side functions: $t = f(\vec{n}) \;\rightarrow\; \forall x \, (f(\vec{n}) \neq x \vee t = x)$
  - flatten nested functions: $f(g(\vec{n})) = t \;\rightarrow\; \forall x \, (g(\vec{n}) \neq x \vee f(x) = t)$
- s = grounding of KB with all names
  - Semantics: infinitely many
  - Lemma: Names in KB, α + one per variable suffice
- $s \mid\!\approx \mathbf{K}_k \alpha$
  - Lemma: Names in KB, α + 1 suffice for quantification

Theorems:

- $s \mid\!\approx \mathbf{K}_k \alpha$ is decidable quantifier-free, constant k: tractable

Reasoning with limited belief

Question: If we know KB, do we then know α at level k?

- KB conjunction of clauses
  - skolemize existentials: $\forall \vec{x} \exists y \, \mathrm{KB} \;\rightarrow\; \forall \vec{x} \, \mathrm{KB}^{y}_{f(\vec{x})}$
  - flatten right-hand side functions: $t = f(\vec{n}) \;\rightarrow\; \forall x \, (f(\vec{n}) \neq x \vee t = x)$
  - flatten nested functions: $f(g(\vec{n})) = t \;\rightarrow\; \forall x \, (g(\vec{n}) \neq x \vee f(x) = t)$
- s = grounding of KB with all names
  - Semantics: infinitely many
  - Lemma: Names in KB, α + one per variable suffice
- $s \mid\!\approx \mathbf{K}_k \alpha$
  - Lemma: Names in KB, α + 1 suffice for quantification

Theorems:

- $s \mid\!\approx \mathbf{K}_k \alpha$ is decidable quantifier-free, constant k: tractable
- If $s \mid\!\approx \mathbf{K}_k \alpha$, then $s \models \alpha$

Reasoning with limited belief

Question: If we know KB, do we then know α at level k?

- KB conjunction of clauses
  - skolemize existentials: $\forall \vec{x} \exists y \, \mathrm{KB} \;\rightarrow\; \forall \vec{x} \, \mathrm{KB}^{y}_{f(\vec{x})}$
  - flatten right-hand side functions: $t = f(\vec{n}) \;\rightarrow\; \forall x \, (f(\vec{n}) \neq x \vee t = x)$
  - flatten nested functions: $f(g(\vec{n})) = t \;\rightarrow\; \forall x \, (g(\vec{n}) \neq x \vee f(x) = t)$
- s = grounding of KB with all names
  - Semantics: infinitely many
  - Lemma: Names in KB, α + one per variable suffice
- $s \mid\!\approx \mathbf{K}_k \alpha$
  - Lemma: Names in KB, α + 1 suffice for quantification

Theorems:

- $s \mid\!\approx \mathbf{K}_k \alpha$ is decidable — quantifier-free, constant k: tractable
- If $s \mid\!\approx \mathbf{K}_k \alpha$, then $s \models \alpha$ — quantifier-free: converse for large k

Example: Sally's father

Knowledge base:

- Frank or Fred is Sally's father
- Sally's father is rich

Example: Sally's father

Knowledge base:

- $\text{fatherOf}(\text{Sally}) = \text{Frank} \vee \text{fatherOf}(\text{Sally}) = \text{Fred}$
- $\text{Rich}(\text{fatherOf}(\text{Sally}))$

Example: Sally's father

Knowledge base:

- $\text{fatherOf}(\text{Sally}) = \text{Frank} \vee \text{fatherOf}(\text{Sally}) = \text{Fred}$
- $\text{rich}(\text{fatherOf}(\text{Sally})) = \top$

Example: Sally's father

Knowledge base:

- $\text{fatherOf}(\text{Sally}) = \text{Frank} \vee \text{fatherOf}(\text{Sally}) = \text{Fred}$
- $\forall x\,(\text{fatherOf}(\text{Sally}) \neq x \vee \text{rich}(x) = \top)$

Example: Sally's father

Knowledge base:

- $\mathrm{fatherOf}(\mathrm{Sally}) = \mathrm{Frank} \vee \mathrm{fatherOf}(\mathrm{Sally}) = \mathrm{Fred}$
- $\forall x\,(\mathrm{fatherOf}(\mathrm{Sally}) \neq x \vee \mathrm{rich}(x) = \top)$

Are Frank or Fred rich?

Example: Sally's father

Setup s contains:

- $\mathrm{fatherOf}(\mathrm{Sally}) = \mathrm{Frank} \vee \mathrm{fatherOf}(\mathrm{Sally}) = \mathrm{Fred}$
- $\mathrm{fatherOf}(\mathrm{Sally}) \neq n \vee \mathrm{rich}(n) = \top$ for every n

$s \mid\!\approx \mathbf{K}_1 (\mathrm{rich}(\mathrm{Frank}) = \top \vee \mathrm{rich}(\mathrm{Fred}) = \top)$?

Example: Sally's father

Setup s contains:

- $\mathrm{fatherOf}(\mathrm{Sally}) = \mathrm{Frank} \vee \mathrm{fatherOf}(\mathrm{Sally}) = \mathrm{Fred}$
- $\mathrm{fatherOf}(\mathrm{Sally}) \neq n \vee \mathrm{rich}(n) = \top$ for every n

$s \mid\!\approx \mathbf{K}_1 (\mathrm{rich}(\mathrm{Frank}) = \top \vee \mathrm{rich}(\mathrm{Fred}) = \top)$?

- $s \cup \{\mathrm{fatherOf}(\mathrm{Sally}) = \mathrm{Frank}\}$ contains $\mathrm{rich}(\mathrm{Frank}) = \top$

Example: Sally's father

Setup s contains:

- $\mathrm{fatherOf}(\mathrm{Sally}) = \mathrm{Frank} \vee \mathrm{fatherOf}(\mathrm{Sally}) = \mathrm{Fred}$
- $\mathrm{fatherOf}(\mathrm{Sally}) \neq n \vee \mathrm{rich}(n) = \top$ for every n

$s \mid\!\approx \mathbf{K}_1 (\mathrm{rich}(\mathrm{Frank}) = \top \vee \mathrm{rich}(\mathrm{Fred}) = \top)$?

- $s \cup \{\mathrm{fatherOf}(\mathrm{Sally}) = \mathrm{Frank}\}$ contains $\mathrm{rich}(\mathrm{Frank}) = \top$
- $s \cup \{\mathrm{fatherOf}(\mathrm{Sally}) = \mathrm{Fred}\}$ contains $\mathrm{rich}(\mathrm{Fred}) = \top$

Example: Sally's father

Setup s contains:

- $\mathrm{fatherOf}(\mathrm{Sally}) = \mathrm{Frank} \vee \mathrm{fatherOf}(\mathrm{Sally}) = \mathrm{Fred}$
- $\mathrm{fatherOf}(\mathrm{Sally}) \neq n \vee \mathrm{rich}(n) = \top$ for every n

$s \mid\!\approx \mathbf{K}_1 (\mathrm{rich}(\mathrm{Frank}) = \top \vee \mathrm{rich}(\mathrm{Fred}) = \top)$?

- $s \cup \{\mathrm{fatherOf}(\mathrm{Sally}) = \mathrm{Frank}\}$ contains $\mathrm{rich}(\mathrm{Frank}) = \top$
- $s \cup \{\mathrm{fatherOf}(\mathrm{Sally}) = \mathrm{Fred}\}$ contains $\mathrm{rich}(\mathrm{Fred}) = \top$
- $s \cup \{\mathrm{fatherOf}(\mathrm{Sally}) = n\}$ for any other n contains empty clause

Example: Sally's father

Setup s contains:

- $\mathrm{fatherOf}(\mathrm{Sally}) = \mathrm{Frank} \vee \mathrm{fatherOf}(\mathrm{Sally}) = \mathrm{Fred}$
- $\mathrm{fatherOf}(\mathrm{Sally}) \neq n \vee \mathrm{rich}(n) = \top$ for every n

$s \mid\!\approx \mathbf{K}_1 (\mathrm{rich}(\mathrm{Frank}) = \top \vee \mathrm{rich}(\mathrm{Fred}) = \top)$? ✓

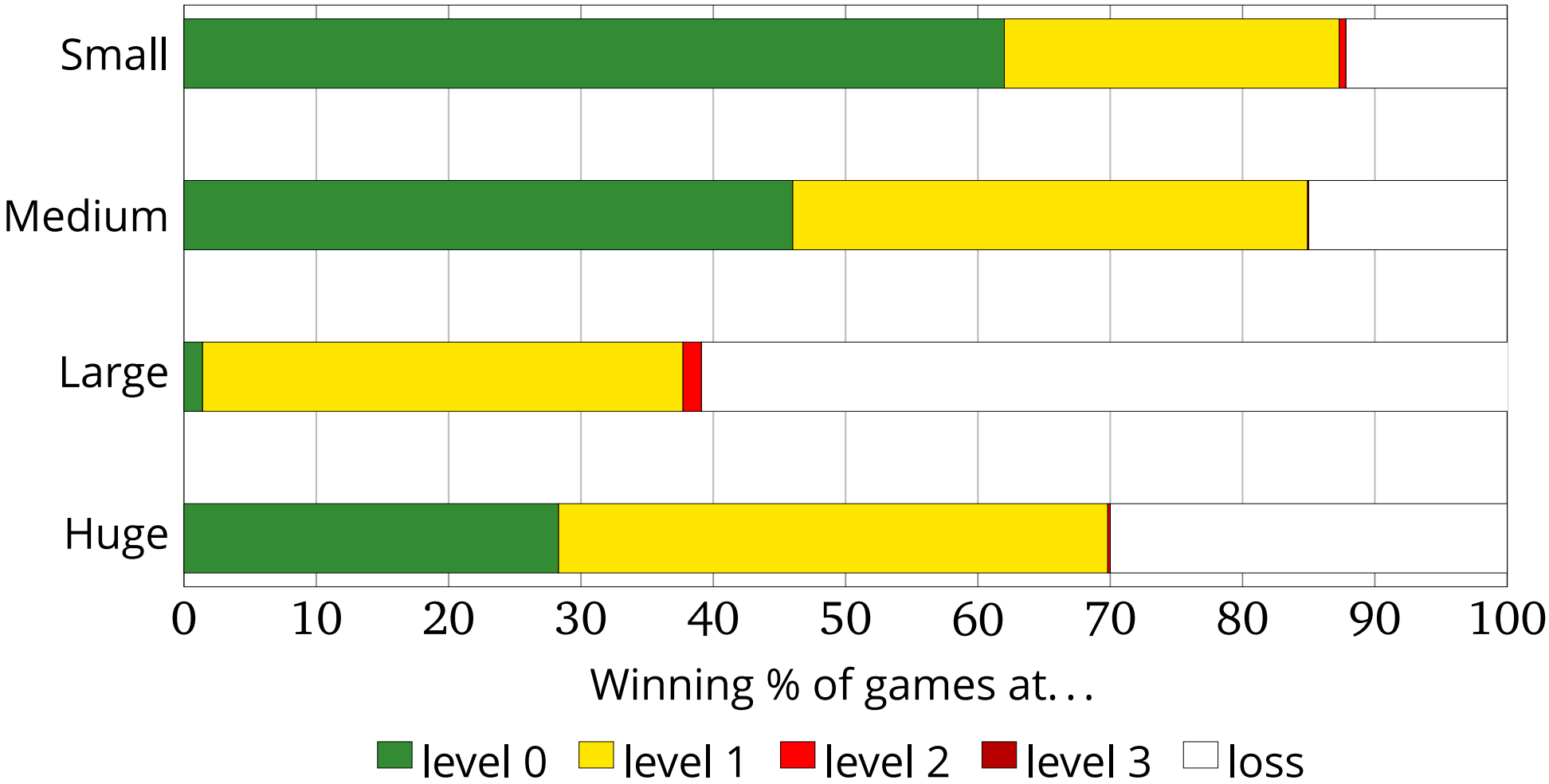
- $s \cup \{\mathrm{fatherOf}(\mathrm{Sally}) = \mathrm{Frank}\}$ contains $\mathrm{rich}(\mathrm{Frank}) = \top$
- $s \cup \{\mathrm{fatherOf}(\mathrm{Sally}) = \mathrm{Fred}\}$ contains $\mathrm{rich}(\mathrm{Fred}) = \top$
- $s \cup \{\mathrm{fatherOf}(\mathrm{Sally}) = n\}$ for any other n contains empty clause

Example: Minesweeper

`http://www.cse.unsw.edu.au/~cschwering/limbo/minesweeper-js.html`

size medium, seed 5, level 0 and 1

Experiment: Minesweeper

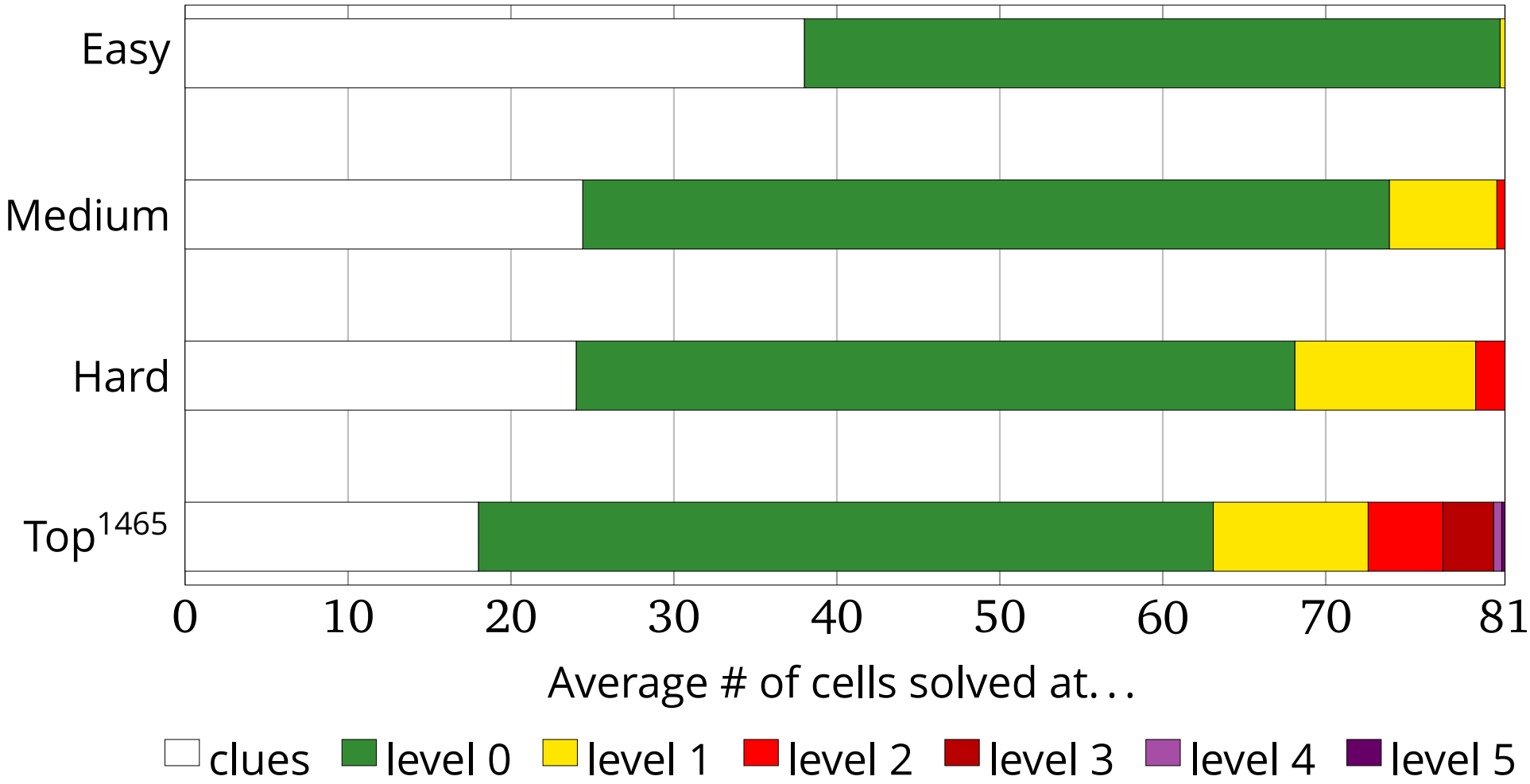


Example: Sudoku

`http://www.cse.unsw.edu.au/~cschwering/limbo/minesweeper/sudoku-js.html`

Jan 29, level 2

Experiment: Sudoku



Extensions

- Dual operator to $\mathbf{K}_k$
  - $\mathbf{M}_k \alpha$ means α
- Introspection
  - $\mathbf{K}_1 \exists x \, (\mathrm{fatherOf}(\mathrm{Sally}) = x \wedge \mathrm{rich}(x) = \top \wedge \mathbf{M}_1 \, \mathrm{fatherOf}(\mathrm{Sally}) \neq x)$
- Conditional beliefs
  - $\mathbf{B}_{k_1,k_2} \mathrm{Aussie}(x) \Rightarrow \mathrm{Eats}(x, \mathrm{roo})$
- Actions and sensing

Actions

- New modal operators
  - $\Box\alpha$ means α holds always only for KB
  - $[A]\alpha$ means α holds after action A

Actions

- New modal operators
  - $\Box\alpha$ means α holds always only for KB
  - $[A]\alpha$ means α holds after action A
- Reiter's situation calculus
  - Regression: reduce query about future to present ✓
  - Progression: update KB to future ✗

Actions

- New modal operators
  - $\Box\alpha$ means α holds always only for KB
  - $[A]\alpha$ means α holds after action A
- Reiter's situation calculus
  - Regression: reduce query about future to present ✓
  - Progression: update KB to future ✗
- Special standard names
  - So far: standard names are atomic like Frank, Sally
  - Now: compound standard names like $\mathrm{kill}(\mathrm{Frank})$
  - Watch out: unlimited nesting makes unit propagation undecidable
    i.e., no $\mathrm{undo}(\mathrm{undo}(\ldots(\mathrm{undo}(A))))$

Example: Unknown Spouse/Father

`http://www.cse.unsw.edu.au/~cschwering/limbo/tui/tui-js.html`

Unknown Spouse/Father

Complexity

Input: setup s, formula α, belief level $k \geq 0$

Problem: $s \mid\!\approx \mathbf{K}_k \alpha$?

Complexity

Input: setup s, formula α, belief level $k \geq 0$

Problem: $s \mid\!\approx \mathbf{K}_k \alpha$?

- With quantifiers: decidable 
- Without quantifiers, constant k: PTIME 

Complexity

Input: setup s, formula α, belief level $k \geq 0$

Problem: $s \mid\!\approx \mathbf{K}_k \alpha$?

- With quantifiers: decidable 
- Without quantifiers, constant k: PTIME 

What if k is not constant?

Complexity

Input: setup s, formula α, belief level $k \geq 0$

Problem: $s \mid\!\approx \mathbf{K}_k \alpha$?

- With quantifiers: decidable 
- Without quantifiers, constant k: PTIME 

What if k is not constant?

Here: quantifier-free case only

Classical complexity (1/2)

Quantified Boolean Formula:

- $\exists X \phi$ iff ϕ^X_{TRUE} or ϕ^X_{FALSE}
- $\forall X \phi$ iff ϕ^X_{TRUE} and ϕ^X_{FALSE}

Classical complexity (1/2)

Quantified Boolean Formula:

- $\exists X\, \phi$ iff ϕ^X_{TRUE} or ϕ^X_{FALSE}
- $\forall X\, \phi$ iff ϕ^X_{TRUE} and ϕ^X_{FALSE}

NP	$\exists \vec{X}\, \phi$
co-NP	$\forall \vec{X}\, \phi$
Σ^{P}_k	$\exists \vec{X}_1\, \forall \vec{X}_2\, \exists \vec{X}_1\, \ldots\, \mathsf{Q}\, \vec{X}_k\, \phi$
Π^{P}_k	$\forall \vec{X}_1\, \exists \vec{X}_2\, \forall \vec{X}_1\, \ldots\, \mathsf{Q}\, \vec{X}_k\, \phi$
PSPACE	$\forall \vec{X}_1\, \exists \vec{X}_2\, \forall \vec{X}_3\, \exists \vec{X}_4\, \ldots\, \phi$

Classical complexity (2/2)

- $s \mid\!\approx \mathbf{K}_0 \alpha$ iff $s \mid\!\approx \alpha$
- $s \mid\!\approx \mathbf{K}_{k+1} \alpha$ iff for some t, for all n, $s \cup \{t = n\} \mid\!\approx \mathbf{K}_k \alpha$

Classical complexity (2/2)

- $s \mid\!\approx \mathbf{K}_0\,\alpha$ iff $s \mid\!\approx \alpha$
- $s \mid\!\approx \mathbf{K}_{k+1}\,\alpha$ iff for some t, for all n, $s \cup \{t = n\} \mid\!\approx \mathbf{K}_k\,\alpha$

Order of splits matters:

$$f = n \vee g_1 = n \vee c \qquad f \neq n \vee g_2 = n \vee c$$
$$f = n \vee g_1 \neq n \vee c \qquad f \neq n \vee g_2 \neq n \vee c$$

Classical complexity (2/2)

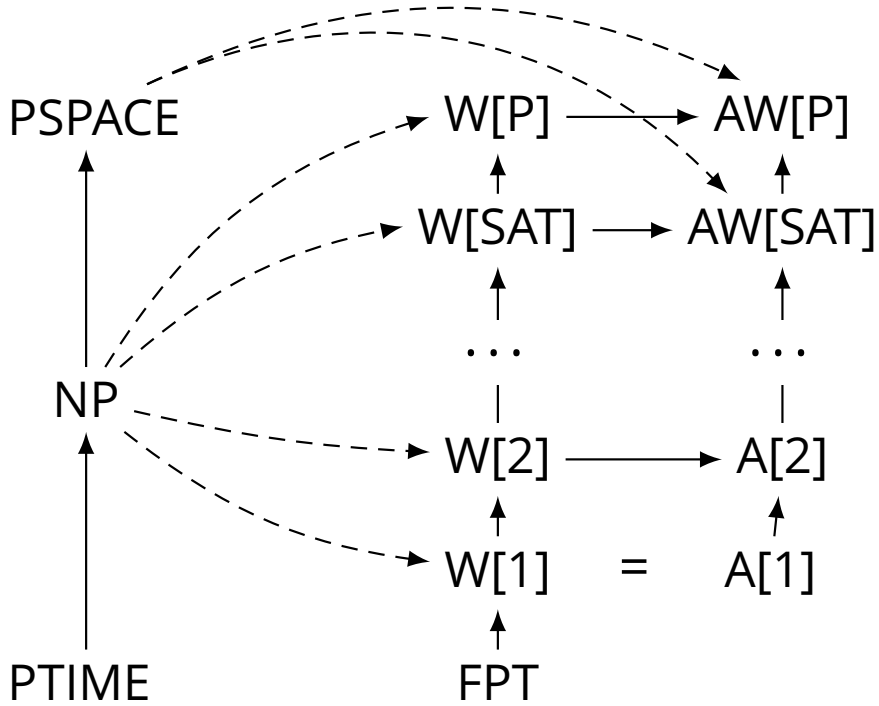
- $s \mid\!\approx \mathbf{K}_0\,\alpha$ iff $s \mid\!\approx \alpha$
- $s \mid\!\approx \mathbf{K}_{k+1}\,\alpha$ iff for some t, for all n, $s \cup \{t = n\} \mid\!\approx \mathbf{K}_k\,\alpha$

Order of splits matters:

$$f = n \vee g_1 = n \vee c \qquad f \neq n \vee g_2 = n \vee c$$
$$f = n \vee g_1 \neq n \vee c \qquad f \neq n \vee g_2 \neq n \vee c$$

Theorem: $s \models \mathbf{K}_k\,\alpha$ is PSPACE-complete 

Parameterized complexity



- FPT: k-vertex cover
  - $f(k) \cdot p(n)$, f computable, p polynomial
- W[1], A[1]: weighted 3CNF satisfiability
  - $f(k) \cdot p(n)$ steps with EXISTS steps at the end
- (A)W[SAT]: (quantified) weighted satisfiability
- (A)W[P]: (quantified) weighted circuit satisfiability
  - $f(k) \cdot p(n)$ steps + $g(k)$ EXISTS (+ FORALL) steps

Complexity overview

<table>
<tr><th>#Functions</th><th>k</th><th>#Names</th><th></th></tr>
<tr><td rowspan="3">Input</td><td>Input</td><td>—</td><td></td></tr>
<tr><td rowspan="2">Param</td><td>Input</td><td></td></tr>
<tr><td>Param
Const</td><td></td></tr>
<tr><td rowspan="2">Param</td><td>Input
Param</td><td>Input</td><td></td></tr>
<tr><td>—</td><td>Param
Const</td><td></td></tr>
<tr><td>—
Const</td><td>Const
—</td><td>—
—</td><td>PTIME</td></tr>
</table>

Complexity overview

<table>
<tr><th>#Functions</th><th>k</th><th>#Names</th><th></th></tr>
<tr><td rowspan="3">Input</td><td>Input</td><td>—</td><td>PSPACE-complete</td></tr>
<tr><td rowspan="2">Param</td><td>Input</td><td></td></tr>
<tr><td>Param
Const</td><td></td></tr>
<tr><td rowspan="2">Param</td><td>Input
Param</td><td>Input</td><td></td></tr>
<tr><td>—</td><td>Param
Const</td><td></td></tr>
<tr><td>—
Const</td><td>Const
—</td><td>—
—</td><td>PTIME</td></tr>
</table>

Complexity overview

<table>
<tr><th>#Functions</th><th>k</th><th>#Names</th><th></th></tr>
<tr><td rowspan="3">Input</td><td>Input</td><td>—</td><td>PSPACE-complete</td></tr>
<tr><td rowspan="2">Param</td><td>Input</td><td></td></tr>
<tr><td>Param
Const</td><td></td></tr>
<tr><td rowspan="2">Param</td><td>Input
Param</td><td>Input</td><td></td></tr>
<tr><td>—</td><td>Param
Const</td><td>FPT</td></tr>
<tr><td>—
Const</td><td>Const
—</td><td>—
—</td><td>PTIME</td></tr>
</table>

Complexity overview

#Functions	k	#Names	
Input	Input	—	PSPACE-complete
	Param	Input	AW[P]-complete
		Param Const	W[P]-complete
Param	Input Param	Input	co-W[P]-complete
	—	Param Const	FPT
— Const	Const —	— —	PTIME

Summary

- Highly expressive language
  - First-order quantification
  - Incomplete knowledge
  - Introspection
  - Actions
- Attractive (more or less) computational properties
  - Quantifier-free: tractable
  - First-order: decidable

Open problems

- Better performance
  - Sudoku: 1 weekend vs $<$ 1 second
- Multiple agents
- Actions and progression
- Limited belief revision
- Probabilities
- Reduce complexity
- Robots

Bibliography

- Lakemeyer, Leveque: Evaluation-based reasoning with disjunctive information in first-order knowledge bases. KR 2002
- Liu, Lakemeyer, Levesque: A logic of limited belief for reasoning with disjunctive information. KR 2004
- Claßen, Lakemeyer: Tractable first-order Golog with disjunctive knowledge bases. Commonsense 2009
- Lakemeyer, Levesque: Decidable reasoning in a logic of limited belief with introspection and unknown individuals. IJCAI 2013
- Lakemeyer, Leveque: Decidable reasoning in a fragment of the epistemic situation calculus. KR 2014
- Lakemeyer, Leveque: Decidable reasoning in a logic of limited belief with function symbols. KR 2016
- Klassen, McIlraith, Levesque: Towards tractable inference for resource-bounded agents. Commonsense 2015
- Schwering, Lakemeyer: Decidable reasoning in a first-order logic of limited conditional belief. ECAI 2016
- Schwering: A Reasoning System for a First-Order Logic of Limited Belief. IJCAI 2017
- Schwering: Reasoning in the Situation Calculus with Limited Belief. Commonsense 2017
- Chen, Saffidine, Schwering: The Complexity of Limited Belief Reasoning. IJCAI 2018