Fundamentals of Artificial Intelligence
Chapter 08: **First-Order Logic**

Roberto Sebastiani

DISI, Università di Trento, Italy – roberto.sebastiani@unitn.it
http://disi.unitn.it/rseba/DIDATTICA/fai_2020/

Teaching assistant: Mauro Dragoni – dragoni@fbk.eu
http://www.maurodragoni.com/teaching/fai/

M.S. Course “Artificial Intelligence Systems”, academic year 2020-2021

Last update: Tuesday 8th December, 2020, 13:08

Outline

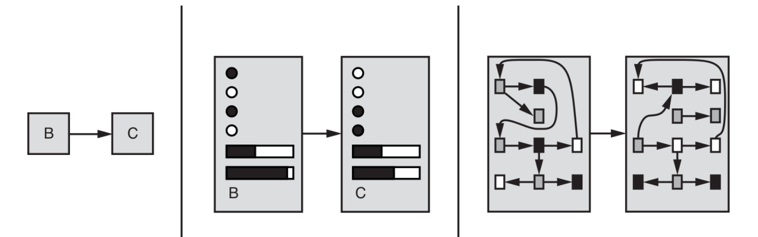
1. Generalities
2. Syntax and Semantics of FOL
3. Using FOL
4. Knowledge Engineering in FOL

Outline

Recall: State Representations [Ch. 02]

Representations of states and transitions

- Three ways to represent states and transitions between them:
  - atomic: a state is a black box with no internal structure
  - factored: a state consists of a vector of attribute values
  - structured: a state includes objects, each of which may have attributes of its own as well as relationships to other objects
- increasing expressive power and computational complexity
- reality represented at different levels of abstraction



(a) Atomic (b) Factored (b) Structured

(© S. Russell & P. Norwig, AIMA)

Pros and Cons of Propositional Logic

+ PL language is formal
  - non-ambiguous semantics
  - unlike natural language, which is intrinsically ambiguous (ex "key")
+ PL is declarative
  - knowledge and inference are separate
  - inference is entirely domain independent
+ PL allows for partial/disjunctive/negated information
  - unlike, e.g., data bases
+ PL is compositional
  - the meaning of $(A \wedge B) \rightarrow C$ derives from the meaning of A,B,C
+ The meaning of PL sentence is context independent
  - unlike with natural language, where meaning depends on context
- PL has has very limited expressive power
  - unlike natural language
  - cannot concisely describe an environment with many objects
  - e.g., cannot say "pits cause breezes in adjacent squares" (need writing one sentence for each square)

Pros and Cons of Propositional Logic

+ PL language is formal
  - non-ambiguous semantics
  - unlike natural language, which is intrinsically ambiguous (ex “key”)
+ PL is declarative
  - knowledge and inference are separate
  - inference is entirely domain independent
+ PL allows for partial/disjunctive/negated information
  - unlike, e.g., data bases
+ PL is compositional
  - the meaning of $(A \wedge B) \rightarrow C$ derives from the meaning of A,B,C
+ The meaning of PL sentence is context independent
  - unlike with natural language, where meaning depends on context
- PL has has very limited expressive power
  - unlike natural language
  - cannot concisely describe an environment with many objects
  - e.g., cannot say “pits cause breezes in adjacent squares” (need writing one sentence for each square)

Pros and Cons of Propositional Logic

+ PL language is formal
  - non-ambiguous semantics
  - unlike natural language, which is intrinsically ambiguous (ex “key”)
+ PL is declarative
  - knowledge and inference are separate
  - inference is entirely domain independent
+ PL allows for partial/disjunctive/negated information
  - unlike, e.g., data bases
+ PL is compositional
  - the meaning of $(A \wedge B) \rightarrow C$ derives from the meaning of A,B,C
+ The meaning of PL sentence is context independent
  - unlike with natural language, where meaning depends on context
- PL has has very limited expressive power
  - unlike natural language
  - cannot concisely describe an environment with many objects
  - e.g., cannot say “pits cause breezes in adjacent squares” (need writing one sentence for each square)

Pros and Cons of Propositional Logic

+ PL language is formal
  - non-ambiguous semantics
  - unlike natural language, which is intrinsically ambiguous (ex “key”)
+ PL is declarative
  - knowledge and inference are separate
  - inference is entirely domain independent
+ PL allows for partial/disjunctive/negated information
  - unlike, e.g., data bases
+ PL is compositional
  - the meaning of $(A \land B) \rightarrow C$ derives from the meaning of A,B,C
+ The meaning of PL sentence is context independent
  - unlike with natural language, where meaning depends on context
- PL has has very limited expressive power
  - unlike natural language
  - cannot concisely describe an environment with many objects
  - e.g., cannot say “pits cause breezes in adjacent squares” (need writing one sentence for each square)

Pros and Cons of Propositional Logic

+ PL language is formal
  - non-ambiguous semantics
  - unlike natural language, which is intrinsically ambiguous (ex “key”)
+ PL is declarative
  - knowledge and inference are separate
  - inference is entirely domain independent
+ PL allows for partial/disjunctive/negated information
  - unlike, e.g., data bases
+ PL is compositional
  - the meaning of $(A \land B) \rightarrow C$ derives from the meaning of A,B,C
+ The meaning of PL sentence is context independent
  - unlike with natural language, where meaning depends on context
- PL has has very limited expressive power
  - unlike natural language
  - cannot concisely describe an environment with many objects
  - e.g., cannot say “pits cause breezes in adjacent squares” (need writing one sentence for each square)

Pros and Cons of Propositional Logic

+ PL language is formal
  - non-ambiguous semantics
  - unlike natural language, which is intrinsically ambiguous (ex “key”)
+ PL is declarative
  - knowledge and inference are separate
  - inference is entirely domain independent
+ PL allows for partial/disjunctive/negated information
  - unlike, e.g., data bases
+ PL is compositional
  - the meaning of $(A \wedge B) \rightarrow C$ derives from the meaning of A,B,C
+ The meaning of PL sentence is context independent
  - unlike with natural language, where meaning depends on context

- PL has has very limited expressive power
  - unlike natural language
  - cannot concisely describe an environment with many objects
  - e.g., cannot say “pits cause breezes in adjacent squares” (need writing one sentence for each square)

Pros and Cons of Propositional Logic

+ PL language is formal
  - non-ambiguous semantics
  - unlike natural language, which is intrinsically ambiguous (ex “key”)
+ PL is declarative
  - knowledge and inference are separate
  - inference is entirely domain independent
+ PL allows for partial/disjunctive/negated information
  - unlike, e.g., data bases
+ PL is compositional
  - the meaning of $(A \land B) \rightarrow C$ derives from the meaning of A,B,C
+ The meaning of PL sentence is context independent
  - unlike with natural language, where meaning depends on context

\- PL has has very limited expressive power
  - unlike natural language
  - cannot concisely describe an environment with many objects
  - e.g., cannot say “pits cause breezes in adjacent squares” (need writing one sentence for each square)

First-Order Logic (FOL)

- PL assumes world contains facts
  - atomic events
- First-order logic (FOL) assumes the world contains:
  - Objects: e.g., people, houses, numbers, theories, Jim Morrison, colors, basketball games, wars, centuries, ...
  - Relations: e.g., red, round, bogus, prime, tall ..., brother of, bigger than, inside, part of, has color, occurred after, owns, comes between, ...
  - Functions: e.g., father of, best friend, one more than, end of, ...

First-Order Logic (FOL)

- PL assumes world contains facts
  - atomic events
- First-order logic (FOL) assumes the world contains:
  - Objects: e.g., people, houses, numbers, theories, Jim Morrison, colors, basketball games, wars, centuries, ...
  - Relations: e.g., red, round, bogus, prime, tall ..., brother of, bigger than, inside, part of, has color, occurred after, owns, comes between, ...
  - Functions: e.g., father of, best friend, one more than, end of, ...

First-Order Logic (FOL)

- PL assumes world contains facts
  - atomic events
- First-order logic (FOL) assumes the world contains:
  - Objects: e.g., people, houses, numbers, theories, Jim Morrison, colors, basketball games, wars, centuries, ...
  - Relations: e.g., red, round, bogus, prime, tall ..., brother of, bigger than, inside, part of, has color, occurred after, owns, comes between, ...
  - Functions: e.g., father of, best friend, one more than, end of, ...

First-Order Logic (FOL)

- PL assumes world contains facts
  - atomic events
- First-order logic (FOL) assumes the world contains:
  - Objects: e.g., people, houses, numbers, theories, Jim Morrison, colors, basketball games, wars, centuries, ...
  - Relations: e.g., red, round, bogus, prime, tall ...,
    brother of, bigger than, inside, part of, has color, occurred after, owns, comes between, ...
  - Functions: e.g., father of, best friend, one more than, end of, ...

First-Order Logic (FOL)

- PL assumes world contains facts
  - atomic events
- First-order logic (FOL) assumes the world contains:
  - Objects: e.g., people, houses, numbers, theories, Jim Morrison, colors, basketball games, wars, centuries, ...
  - Relations: e.g., red, round, bogus, prime, tall ..., brother of, bigger than, inside, part of, has color, occurred after, owns, comes between, ...
  - Functions: e.g., father of, best friend, one more than, end of, ...

First-Order Logic (FOL)

- PL assumes world contains facts
  - atomic events
- First-order logic (FOL) assumes the world contains:
  - Objects: e.g., people, houses, numbers, theories, Jim Morrison, colors, basketball games, wars, centuries, ...
  - Relations: e.g., red, round, bogus, prime, tall ..., brother of, bigger than, inside, part of, has color, occurred after, owns, comes between, ...
  - Functions: e.g., father of, best friend, one more than, end of, ...

Logics in General

- Ontological Commitment: What exists in the world
- Epistemological Commitment: What an agent believes about facts

Language	Ontological Commitment	Epistemological Commitment
Propositional logic	facts	true/false/unknown
First-order logic	facts, objects, relations	true/false/unknown
Temporal logic	facts, objects, relations, times	true/false/unknown
Probability theory	facts	degree of belief
Fuzzy logic	facts + degree of truth	known interval value

Logics in General

- **Ontological Commitment**: What exists in the world
- Epistemological Commitment: What an agent believes about facts

Language	Ontological Commitment	Epistemological Commitment
Propositional logic	facts	true/false/unknown
First-order logic	facts, objects, relations	true/false/unknown
Temporal logic	facts, objects, relations, times	true/false/unknown
Probability theory	facts	degree of belief
Fuzzy logic	facts + degree of truth	known interval value

Logics in General

- Ontological Commitment: What exists in the world
- Epistemological Commitment: What an agent believes about facts

Language	Ontological Commitment	Epistemological Commitment
Propositional logic	facts	true/false/unknown
First-order logic	facts, objects, relations	true/false/unknown
Temporal logic	facts, objects, relations, times	true/false/unknown
Probability theory	facts	degree of belief
Fuzzy logic	facts + degree of truth	known interval value

Outline

Syntax of FOL: Basic Elements

- Constant symbols: KingJohn, 2, UniversityofTrento,...
- Predicate symbols: Man(.), Brother(.,.), $(. > .)$, AllDifferent(...),...
  - may have different arities (1,2,3,...)
  - may be prefix (e.g. Brother(.,.)) or infix (e.g. $(. > .)$)
- Function symbols: Sqrt, Leftleg, MotherOf
  - may have different arities (1,2,3,...)
  - may be prefix (e.g. Sqrt(.)) or infix (e.g. $(. + .)$)
- Variable symbols: x, y, a, b, ...
- Propositional Connectives: $\neg, \wedge, \vee, \rightarrow, \leftrightarrow, \oplus$
- Equality: "$=$" (also "$\neq$" s.t. "$a \neq b$" shortcut for "$\neg(a = b)$")
- Quantifiers: "$\forall$" ("forall"), "$\exists$" ("exists", aka "for some")
- Punctuation Symbols: ",", "(", ")"

- Constants symbols are 0-ary function symbols
- Propositions are 0-ary predicates $\Longrightarrow$ PL subcase of FOL
- Signature: the set of predicate, function & constant symbols

Syntax of FOL: Basic Elements

- Constant symbols: KingJohn, 2, UniversityofTrento,...
- Predicate symbols: Man(.), Brother(.,.), $(. > .)$, AllDifferent(...),...
  - may have different arities (1,2,3,...)
  - may be prefix (e.g. Brother(.,.)) or infix (e.g. $(. > .)$)
- Function symbols: Sqrt, Leftleg, MotherOf
  - may have different arities (1,2,3,...)
  - may be prefix (e.g. Sqrt(.)) or infix (e.g. $(. + .)$)
- Variable symbols: x, y, a, b, ...
- Propositional Connectives: $\neg, \wedge, \vee, \rightarrow, \leftrightarrow, \oplus$
- Equality: "$=$" (also "$\neq$" s.t. "$a \neq b$" shortcut for "$\neg(a = b)$")
- Quantifiers: "$\forall$" ("forall"), "$\exists$" ("exists", aka "for some")
- Punctuation Symbols: ",", "(", ")"

- Constants symbols are 0-ary function symbols
- Propositions are 0-ary predicates $\Longrightarrow$ PL subcase of FOL
- Signature: the set of predicate, function & constant symbols

Syntax of FOL: Basic Elements

- Constant symbols: KingJohn, 2, UniversityofTrento,...
- Predicate symbols: Man(.), Brother(.,.), $(. > .)$, AllDifferent(...),...
  - may have different arities (1,2,3,...)
  - may be prefix (e.g. Brother(.,.)) or infix (e.g. $(. > .)$)
- Function symbols: Sqrt, Leftleg, MotherOf
  - may have different arities (1,2,3,...)
  - may be prefix (e.g. Sqrt(.)) or infix (e.g. $(. + .)$)
- Variable symbols: x, y, a, b, ...
- Propositional Connectives: $\neg, \wedge, \vee, \rightarrow, \leftrightarrow, \oplus$
- Equality: "$=$" (also "$\neq$" s.t. "$a \neq b$" shortcut for "$\neg(a = b)$")
- Quantifiers: "$\forall$" ("forall"), "$\exists$" ("exists", aka "for some")
- Punctuation Symbols: ",", "(", ")"

- Constants symbols are 0-ary function symbols
- Propositions are 0-ary predicates $\Longrightarrow$ PL subcase of FOL
- Signature: the set of predicate, function & constant symbols

Syntax of FOL: Basic Elements

- Constant symbols: KingJohn, 2, UniversityofTrento,...
- Predicate symbols: Man(.), Brother(.,.), $(. > .)$, AllDifferent(...),...
  - may have different arities (1,2,3,...)
  - may be prefix (e.g. Brother(.,.)) or infix (e.g. $(. > .)$)
- Function symbols: Sqrt, Leftleg, MotherOf
  - may have different arities (1,2,3,...)
  - may be prefix (e.g. Sqrt(.)) or infix (e.g. $(. + .)$)
- Variable symbols: x, y, a, b, ...
- Propositional Connectives: $\neg, \wedge, \vee, \rightarrow, \leftrightarrow, \oplus$
- Equality: "$=$" (also "$\neq$" s.t. "$a \neq b$" shortcut for "$\neg(a = b)$")
- Quantifiers: "$\forall$" ("forall"), "$\exists$" ("exists", aka "for some")
- Punctuation Symbols: ",", "(", ")"

- Constants symbols are 0-ary function symbols
- Propositions are 0-ary predicates $\Longrightarrow$ PL subcase of FOL
- Signature: the set of predicate, function & constant symbols

Syntax of FOL: Basic Elements

- Constant symbols: KingJohn, 2, UniversityofTrento,...
- Predicate symbols: Man(.), Brother(.,.), $(. > .)$, AllDifferent(...),...
  - may have different arities (1,2,3,...)
  - may be prefix (e.g. Brother(.,.)) or infix (e.g. $(. > .)$)
- Function symbols: Sqrt, Leftleg, MotherOf
  - may have different arities (1,2,3,...)
  - may be prefix (e.g. Sqrt(.)) or infix (e.g. $(. + .)$)
- Variable symbols: x, y, a, b, ...
- Propositional Connectives: $\neg, \wedge, \vee, \rightarrow, \leftrightarrow, \oplus$
- Equality: "$=$" (also "$\neq$" s.t. "$a \neq b$" shortcut for "$\neg(a = b)$")
- Quantifiers: "$\forall$" ("forall"), "$\exists$" ("exists", aka "for some")
- Punctuation Symbols: ",", "(", ")"

- Constants symbols are 0-ary function symbols
- Propositions are 0-ary predicates $\Longrightarrow$ PL subcase of FOL
- Signature: the set of predicate, function & constant symbols

Syntax of FOL: Basic Elements

- Constant symbols: KingJohn, 2, UniversityofTrento,...
- Predicate symbols: Man(.), Brother(.,.), $(. > .)$, AllDifferent(...),...
  - may have different arities (1,2,3,...)
  - may be prefix (e.g. Brother(.,.)) or infix (e.g. $(. > .)$)
- Function symbols: Sqrt, Leftleg, MotherOf
  - may have different arities (1,2,3,...)
  - may be prefix (e.g. Sqrt(.)) or infix (e.g. $(. + .)$)
- Variable symbols: x, y, a, b, ...
- Propositional Connectives: $\neg, \wedge, \vee, \rightarrow, \leftrightarrow, \oplus$
- Equality: "$=$" (also "$\neq$" s.t. "$a \neq b$" shortcut for "$\neg(a = b)$")
- Quantifiers: "$\forall$" ("forall"), "$\exists$" ("exists", aka "for some")
- Punctuation Symbols: ",", "(", ")"

- Constants symbols are 0-ary function symbols
- Propositions are 0-ary predicates $\Longrightarrow$ PL subcase of FOL
- Signature: the set of predicate, function & constant symbols

Syntax of FOL: Basic Elements

- Constant symbols: KingJohn, 2, UniversityofTrento,...
- Predicate symbols: Man(.), Brother(.,.), $(. > .)$, AllDifferent(...),...
  - may have different arities (1,2,3,...)
  - may be prefix (e.g. Brother(.,.)) or infix (e.g. $(. > .)$)
- Function symbols: Sqrt, Leftleg, MotherOf
  - may have different arities (1,2,3,...)
  - may be prefix (e.g. Sqrt(.)) or infix (e.g. $(. + .)$)
- Variable symbols: x, y, a, b, ...
- Propositional Connectives: $\neg, \wedge, \vee, \rightarrow, \leftrightarrow, \oplus$
- Equality: "$=$" (also "$\neq$" s.t. "$a \neq b$" shortcut for "$\neg(a = b)$")
- Quantifiers: "$\forall$" ("forall"), "$\exists$" ("exists", aka "for some")
- Punctuation Symbols: ",", "(", ")"

- Constants symbols are 0-ary function symbols
- Propositions are 0-ary predicates $\Longrightarrow$ PL subcase of FOL
- Signature: the set of predicate, function & constant symbols

Syntax of FOL: Basic Elements

- Constant symbols: KingJohn, 2, UniversityofTrento,...
- Predicate symbols: Man(.), Brother(.,.), $(. > .)$, AllDifferent(...),...
  - may have different arities (1,2,3,...)
  - may be prefix (e.g. Brother(.,.)) or infix (e.g. $(. > .)$)
- Function symbols: Sqrt, Leftleg, MotherOf
  - may have different arities (1,2,3,...)
  - may be prefix (e.g. Sqrt(.)) or infix (e.g. $(. + .)$)
- Variable symbols: x, y, a, b, ...
- Propositional Connectives: $\neg, \land, \lor, \rightarrow, \leftrightarrow, \oplus$
- Equality: "$=$" (also "$\neq$" s.t. "$a \neq b$" shortcut for "$\neg(a = b)$")
- Quantifiers: "$\forall$" ("forall"), "$\exists$" ("exists", aka "for some")
- Punctuation Symbols: ",", "(", ")"

- Constants symbols are 0-ary function symbols
- Propositions are 0-ary predicates $\Longrightarrow$ PL subcase of FOL
- Signature: the set of predicate, function & constant symbols

Syntax of FOL: Basic Elements

- Constant symbols: KingJohn, 2, UniversityofTrento,...
- Predicate symbols: Man(.), Brother(.,.), $(. > .)$, AllDifferent(...),...
  - may have different arities (1,2,3,...)
  - may be prefix (e.g. Brother(.,.)) or infix (e.g. $(. > .)$)
- Function symbols: Sqrt, Leftleg, MotherOf
  - may have different arities (1,2,3,...)
  - may be prefix (e.g. Sqrt(.)) or infix (e.g. $(. + .)$)
- Variable symbols: x, y, a, b, ...
- Propositional Connectives: $\neg, \wedge, \vee, \rightarrow, \leftrightarrow, \oplus$
- Equality: "$=$" (also "$\neq$" s.t. "$a \neq b$" shortcut for "$\neg(a = b)$")
- Quantifiers: "$\forall$" ("forall"), "$\exists$" ("exists", aka "for some")
- Punctuation Symbols: ",", "(", ")"

- Constants symbols are 0-ary function symbols
- Propositions are 0-ary predicates $\Longrightarrow$ PL subcase of FOL
- Signature: the set of predicate, function & constant symbols

FOL: Syntax

- Terms:
  - constant or variable or $function(term_1, ..., term_n)$
  - ex: KingJohn, x, Leftleg(Richard), (z*log(2))
  - denote objects
- Atomic sentences (aka atomic formulas):
  - $proposition$ or $predicate(term_1, ..., term_n)$ or $term_1 = term_2$
  - $(Length(Leftleg(Richard)) > Length(Leftleg(KingJohn)))$
  - denote facts
- Non-atomic sentences/formulas:
  - $\neg\alpha$, $\alpha \wedge \beta$, $\alpha \vee \beta$, $\alpha \rightarrow \beta$, $\alpha \leftrightarrow \beta$, $\alpha \oplus \beta$,
    $\forall x.\alpha$, $\exists x.\alpha$ s.t. x occurs in α
  - Ex: $\forall y.(Italian(y) \rightarrow President(Mattarella, y))$
    $\exists x \forall y.President(x, y) \rightarrow \forall y \exists x.President(x, y)$
    $\forall x.(P(x) \wedge Q(x)) \leftrightarrow ((\forall x.P(x)) \wedge (\forall x.Q(x)))$
    $\forall x.(((x \geq 0) \wedge (x \leq \pi)) \rightarrow (sin(x) \geq 0))$
  - denote (complex) facts
- A term/formula is ground iff no variable occurs in it

FOL: Syntax

- Terms:
  - constant or variable or $function(term_1, ..., term_n)$
  - ex: KingJohn, x, Leftleg(Richard), (z*log(2))
  - denote objects
- Atomic sentences (aka atomic formulas):
  - $proposition$ or $predicate(term_1, ..., term_n)$ or $term_1 = term_2$
  - $(Length(Leftleg(Richard)) > Length(Leftleg(KingJohn)))$
  - denote facts
- Non-atomic sentences/formulas:
  - $\neg\alpha$, $\alpha \land \beta$, $\alpha \lor \beta$, $\alpha \rightarrow \beta$, $\alpha \leftrightarrow \beta$, $\alpha \oplus \beta$, $\forall x.\alpha$, $\exists x.\alpha$ s.t. x occurs in α
  - Ex: $\forall y.(Italian(y) \rightarrow President(Mattarella, y))$
    $\exists x \forall y.President(x, y) \rightarrow \forall y \exists x.President(x, y)$
    $\forall x.(P(x) \land Q(x)) \leftrightarrow ((\forall x.P(x)) \land (\forall x.Q(x)))$
    $\forall x.(((x \geq 0) \land (x \leq \pi)) \rightarrow (sin(x) \geq 0))$
  - denote (complex) facts
- A term/formula is ground iff no variable occurs in it

FOL: Syntax

- Terms:
  - constant or variable or $function(term_1, ..., term_n)$
  - ex: KingJohn, x, Leftleg(Richard), (z*log(2))
  - denote objects
- Atomic sentences (aka atomic formulas):
  - $proposition$ or $predicate(term_1, ..., term_n)$ or $term_1 = term_2$
  - $(Length(Leftleg(Richard)) > Length(Leftleg(KingJohn)))$
  - denote facts
- Non-atomic sentences/formulas:
  - $\neg\alpha$, $\alpha \wedge \beta$, $\alpha \vee \beta$, $\alpha \rightarrow \beta$, $\alpha \leftrightarrow \beta$, $\alpha \oplus \beta$, $\forall x.\alpha$, $\exists x.\alpha$ s.t. x occurs in α
  - Ex: $\forall y.(Italian(y) \rightarrow President(Mattarella, y))$
    $\exists x \forall y.President(x, y) \rightarrow \forall y \exists x.President(x, y)$
    $\forall x.(P(x) \wedge Q(x)) \leftrightarrow ((\forall x.P(x)) \wedge (\forall x.Q(x)))$
    $\forall x.(((x \geq 0) \wedge (x \leq \pi)) \rightarrow (sin(x) \geq 0))$
  - denote (complex) facts
- A term/formula is ground iff no variable occurs in it

FOL: Syntax

- Terms:
  - constant or variable or $function(term_1, ..., term_n)$
  - ex: KingJohn, x, Leftleg(Richard), (z*log(2))
  - denote objects
- Atomic sentences (aka atomic formulas):
  - $proposition$ or $predicate(term_1, ..., term_n)$ or $term_1 = term_2$
  - $(Length(Leftleg(Richard)) > Length(Leftleg(KingJohn)))$
  - denote facts
- Non-atomic sentences/formulas:
  - $\neg\alpha$, $\alpha \wedge \beta$, $\alpha \vee \beta$, $\alpha \rightarrow \beta$, $\alpha \leftrightarrow \beta$, $\alpha \oplus \beta$,
    $\forall x.\alpha$, $\exists x.\alpha$ s.t. x occurs in α
  - Ex: $\forall y.(Italian(y) \rightarrow President(Mattarella, y))$
    $\exists x \forall y.President(x, y) \rightarrow \forall y \exists x.President(x, y)$
    $\forall x.(P(x) \wedge Q(x)) \leftrightarrow ((\forall x.P(x)) \wedge (\forall x.Q(x)))$
    $\forall x.(((x \geq 0) \wedge (x \leq \pi)) \rightarrow (sin(x) \geq 0))$
  - denote (complex) facts
- A term/formula is ground iff no variable occurs in it

FOL: Syntax

- Terms:
  - constant or variable or $function(term_1, ..., term_n)$
  - ex: KingJohn, x, Leftleg(Richard), (z*log(2))
  - denote objects
- Atomic sentences (aka atomic formulas):
  - $proposition$ or $predicate(term_1, ..., term_n)$ or $term_1 = term_2$
  - $(Length(Leftleg(Richard)) > Length(Leftleg(KingJohn)))$
  - denote facts
- Non-atomic sentences/formulas:
  - $\neg\alpha$, $\alpha \wedge \beta$, $\alpha \vee \beta$, $\alpha \rightarrow \beta$, $\alpha \leftrightarrow \beta$, $\alpha \oplus \beta$,
    $\forall x.\alpha$, $\exists x.\alpha$ s.t. x occurs in α
  - Ex: $\forall y.(Italian(y) \rightarrow President(Mattarella, y))$
    $\exists x \forall y.President(x, y) \rightarrow \forall y \exists x.President(x, y)$
    $\forall x.(P(x) \wedge Q(x)) \leftrightarrow ((\forall x.P(x)) \wedge (\forall x.Q(x)))$
    $\forall x.(((x \geq 0) \wedge (x \leq \pi)) \rightarrow (sin(x) \geq 0))$
  - denote (complex) facts
- A term/formula is ground iff no variable occurs in it

FOL: Syntax (BNF)

$$
\begin{array}{rcl}
\textit{Sentence} & \rightarrow & \textit{AtomicSentence} \mid \textit{ComplexSentence} \\
\textit{AtomicSentence} & \rightarrow & \textit{Predicate} \mid \textit{Predicate}(\textit{Term}, \ldots) \mid \textit{Term} = \textit{Term} \\
\textit{ComplexSentence} & \rightarrow & (\ \textit{Sentence}\ ) \mid [\ \textit{Sentence}\ ] \\
& \mid & \neg\ \textit{Sentence} \\
& \mid & \textit{Sentence} \land \textit{Sentence} \\
& \mid & \textit{Sentence} \lor \textit{Sentence} \\
& \mid & \textit{Sentence} \ \Rightarrow\ \textit{Sentence} \\
& \mid & \textit{Sentence} \ \Leftrightarrow\ \textit{Sentence} \\
& \mid & \textit{Quantifier}\ \textit{Variable}, \ldots\ \textit{Sentence} \\
\\
\textit{Term} & \rightarrow & \textit{Function}(\textit{Term}, \ldots) \\
& \mid & \textit{Constant} \\
& \mid & \textit{Variable} \\
\\
\textit{Quantifier} & \rightarrow & \forall \mid \exists \\
\textit{Constant} & \rightarrow & A \mid X_1 \mid \textit{John} \mid \cdots \\
\textit{Variable} & \rightarrow & a \mid x \mid s \mid \cdots \\
\textit{Predicate} & \rightarrow & \textit{True} \mid \textit{False} \mid \textit{After} \mid \textit{Loves} \mid \textit{Raining} \mid \cdots \\
\textit{Function} & \rightarrow & \textit{Mother} \mid \textit{LeftLeg} \mid \cdots \\
\textsc{Operator Precedence} & : & \neg, =, \land, \lor, \Rightarrow, \Leftrightarrow
\end{array}
$$

FOL: Semantics

Possible worlds (aka models)

- A model $\mathcal{M}$ is a pair $\langle \mathcal{D}, \mathcal{I} \rangle$ ($\langle$*domain*, *interpretation*$\rangle$)
- Domain $\mathcal{D}$: a non-empty set of objects (aka domain elements)
- Interpretation $\mathcal{I}$: a map on elements of the signature
  - constant symbols $\longmapsto$ domain elements:
    a constant symbol C is mapped into a particular object $[C]^{\mathcal{I}}$ in $\mathcal{D}$
  - predicate symbols $\longmapsto$ domain relations:
    a $k$ary predicate $P(...)$ is mapped into a subset $[P]^{\mathcal{I}}$ of $\mathcal{D}^k$
    (i.e., the set of object tuples satisfying the predicate in this world)
  - functions symbols $\longmapsto$ domain functions:
    a $k$ary function f is mapped into a domain function $[f]^{\mathcal{I}} : \mathcal{D}^k \longmapsto \mathcal{D}$
    ($[f]^{\mathcal{I}}$ must be total)

(we denote by $[.]^{\mathcal{I}}$ the result of the interpretation $\mathcal{I}$)

Remark

Two distinct constants can be mapped into the same object.

FOL: Semantics

Possible worlds (aka models)

- A model $\mathcal{M}$ is a pair $\langle \mathcal{D}, \mathcal{I} \rangle$ ($\langle$*domain*, *interpretation*$\rangle$)
- Domain $\mathcal{D}$: a non-empty set of objects (aka domain elements)
- Interpretation $\mathcal{I}$: a map on elements of the signature
  - constant symbols $\longmapsto$ domain elements:
    a constant symbol C is mapped into a particular object $[C]^{\mathcal{I}}$ in $\mathcal{D}$
  - predicate symbols $\longmapsto$ domain relations:
    a $k$ary predicate $P(...)$ is mapped into a subset $[P]^{\mathcal{I}}$ of $\mathcal{D}^k$
    (i.e., the set of object tuples satisfying the predicate in this world)
  - functions symbols $\longmapsto$ domain functions:
    a $k$ary function f is mapped into a domain function $[f]^{\mathcal{I}} : \mathcal{D}^k \longmapsto \mathcal{D}$
    ($[f]^{\mathcal{I}}$ must be total)

  (we denote by $[.]^{\mathcal{I}}$ the result of the interpretation $\mathcal{I}$)

Remark

Two distinct constants can be mapped into the same object.

FOL: Semantics

Possible worlds (aka models)

- A model $\mathcal{M}$ is a pair $\langle \mathcal{D}, \mathcal{I} \rangle$ ($\langle$*domain*, *interpretation*$\rangle$)
- Domain $\mathcal{D}$: a non-empty set of objects (aka domain elements)
- Interpretation $\mathcal{I}$: a map on elements of the signature
  - constant symbols $\longmapsto$ domain elements:
    a constant symbol C is mapped into a particular object $[C]^{\mathcal{I}}$ in $\mathcal{D}$
  - predicate symbols $\longmapsto$ domain relations:
    a $k$ary predicate $P(...)$ is mapped into a subset $[P]^{\mathcal{I}}$ of $\mathcal{D}^k$
    (i.e., the set of object tuples satisfying the predicate in this world)
  - functions symbols $\longmapsto$ domain functions:
    a $k$ary function f is mapped into a domain function $[f]^{\mathcal{I}} : \mathcal{D}^k \longmapsto \mathcal{D}$
    ($[f]^{\mathcal{I}}$ must be total)

(we denote by $[.]^{\mathcal{I}}$ the result of the interpretation $\mathcal{I}$)

Remark

Two distinct constants can be mapped into the same object.

FOL: Semantics

Possible worlds (aka models)

- A model $\mathcal{M}$ is a pair $\langle \mathcal{D}, \mathcal{I} \rangle$ ($\langle$*domain*, *interpretation*$\rangle$)
- Domain $\mathcal{D}$: a non-empty set of objects (aka domain elements)
- Interpretation $\mathcal{I}$: a map on elements of the signature
  - constant symbols $\longmapsto$ domain elements:
    a constant symbol C is mapped into a particular object $[C]^{\mathcal{I}}$ in $\mathcal{D}$
  - predicate symbols $\longmapsto$ domain relations:
    a $k$ary predicate $P(...)$ is mapped into a subset $[P]^{\mathcal{I}}$ of $\mathcal{D}^k$
    (i.e., the set of object tuples satisfying the predicate in this world)
  - functions symbols $\longmapsto$ domain functions:
    a $k$ary function f is mapped into a domain function $[f]^{\mathcal{I}} : \mathcal{D}^k \longmapsto \mathcal{D}$
    ($[f]^{\mathcal{I}}$ must be total)

(we denote by $[.]^{\mathcal{I}}$ the result of the interpretation $\mathcal{I}$)

Remark

Two distinct constants can be mapped into the same object.

FOL: Semantics [cont.]

Interpretation of terms

- $\mathcal{I}$ maps (ground) terms into domain elements
- A term $f(t_1, ..., t_k)$ is mapped by $\mathcal{I}$ into the value $[f(t_1, ..., t_k)]^{\mathcal{I}}$ returned by applying the domain function $[f]^{\mathcal{I}}$, into which f is mapped, to the values $[t_1]^{\mathcal{I}}, ..., [t_k]^{\mathcal{I}}$ obtained by applying recursively $\mathcal{I}$ to the terms $t_1, ..., t_k$:
  - $[f(t_1, ..., t_k)]^{\mathcal{I}} = [f]^{\mathcal{I}}([t_1]^{\mathcal{I}}, ..., [t_k]^{\mathcal{I}})$
  - Ex: if "Me, Mother, Father" are interpreted as usual, then "Mother(Father(Me))" is interpreted as my (paternal) grandmother
  - Ex: if "$+, -, \cdot, 0, 1, 2, 3, 4$" are interpreted as usual, then "$(3 - 1) \cdot (0 + 2)$" is interpreted as 4

FOL: Semantics [cont.]

Interpretation of terms

- $\mathcal{I}$ maps (ground) terms into domain elements
- A term $f(t_1, ..., t_k)$ is mapped by $\mathcal{I}$ into the value $[f(t_1, ..., t_k)]^{\mathcal{I}}$ returned by applying the domain function $[f]^{\mathcal{I}}$, into which f is mapped, to the values $[t_1]^{\mathcal{I}}, ..., [t_k]^{\mathcal{I}}$ obtained by applying recursively $\mathcal{I}$ to the terms $t_1, ..., t_k$:
  - $[f(t_1, ..., t_k)]^{\mathcal{I}} = [f]^{\mathcal{I}}([t_1]^{\mathcal{I}}, ..., [t_k]^{\mathcal{I}})$
  - Ex: if "Me, Mother, Father" are interpreted as usual, then "Mother(Father(Me))" is interpreted as my (paternal) grandmother
  - Ex: if "$+, -, \cdot, 0, 1, 2, 3, 4$" are interpreted as usual, then "$(3 - 1) \cdot (0 + 2)$" is interpreted as 4

FOL: Semantics [cont.]

Interpretation of terms

- $\mathcal{I}$ maps (ground) terms into domain elements
- A term $f(t_1, ..., t_k)$ is mapped by $\mathcal{I}$ into the value $[f(t_1, ..., t_k)]^{\mathcal{I}}$ returned by applying the domain function $[f]^{\mathcal{I}}$, into which f is mapped, to the values $[t_1]^{\mathcal{I}}, ..., [t_k]^{\mathcal{I}}$ obtained by applying recursively $\mathcal{I}$ to the terms $t_1, ..., t_k$:
  - $[f(t_1, ..., t_k)]^{\mathcal{I}} = [f]^{\mathcal{I}}([t_1]^{\mathcal{I}}, ..., [t_k]^{\mathcal{I}})$
  - Ex: if "Me, Mother, Father" are interpreted as usual, then "Mother(Father(Me))" is interpreted as my (paternal) grandmother
  - Ex: if "$+, -, \cdot, 0, 1, 2, 3, 4$" are interpreted as usual, then "$(3 - 1) \cdot (0 + 2)$" is interpreted as 4

FOL: Semantics [cont.]

Interpretation of formulas

- $\mathcal{I}$ maps (ground) formulas into truth values
- An atomic formula $P(t_1, ..., t_k)$ is true in $\mathcal{I}$ iff the objects into which the terms t_1,...t_k are mapped by $\mathcal{I}$ comply to the relation into which P is mapped
  - $[P(t_1, ..., t_k)]^{\mathcal{I}}$ is true iff $\langle [t_1]^{\mathcal{I}}, ..., [t_k]^{\mathcal{I}} \rangle \in [P]^{\mathcal{I}}$
  - Ex: if "Me, Mother, Father, married" are interpreted as usual, then "Married(Mother(Me),Father(Me))" is interpreted as true
  - Ex: if "$+, -, >, 0, 1, 2, 3, 4$" are interpreted as usual, then "$(4 - 0) > (1 + 2)$" is interpreted as true
- An atomic formula $t_1 = t_2$ is true in $\mathcal{I}$ iff the terms t_1, t_2 are mapped by $\mathcal{I}$ into the same domain element
  - $[t_1 = t_2]^{\mathcal{I}}$ is true iff $[t_1]^{\mathcal{I}}$ same as $[t_2]^{\mathcal{I}}$
  - Ex: if "Mother" is interpreted as usual, Richard, John are brothers, then "Mother(Richard)=Mother(John))" is interpreted as true
  - Ex: if "$+, -, 0, 1, 2, 3, 4$" are interpreted as usual, then "$(4 - 1) = (1 + 2)$" is interpreted as true
- $\neg, \wedge, \vee, \rightarrow, \leftrightarrow, \oplus$ interpreted by $\mathcal{I}$ as in PL

FOL: Semantics [cont.]

Interpretation of formulas

- $\mathcal{I}$ maps (ground) formulas into truth values
- An atomic formula $P(t_1, ..., t_k)$ is true in $\mathcal{I}$ iff the objects into which the terms t_1,...t_k are mapped by $\mathcal{I}$ comply to the relation into which P is mapped
  - $[P(t_1, ..., t_k)]^{\mathcal{I}}$ is true iff $\langle [t_1]^{\mathcal{I}}, ..., [t_k]^{\mathcal{I}} \rangle \in [P]^{\mathcal{I}}$
  - Ex: if "Me, Mother, Father, married" are interpreted as usual, then "Married(Mother(Me),Father(Me))" is interpreted as true
  - Ex: if "$+, -, >, 0, 1, 2, 3, 4$" are interpreted as usual, then "$(4 - 0) > (1 + 2)$" is interpreted as true
- An atomic formula $t_1 = t_2$ is true in $\mathcal{I}$ iff the terms t_1, t_2 are mapped by $\mathcal{I}$ into the same domain element
  - $[t_1 = t_2]^{\mathcal{I}}$ is true iff $[t_1]^{\mathcal{I}}$ same as $[t_2]^{\mathcal{I}}$
  - Ex: if "Mother" is interpreted as usual, Richard, John are brothers, then "Mother(Richard)=Mother(John))" is interpreted as true
  - Ex: if "$+, -, 0, 1, 2, 3, 4$" are interpreted as usual, then "$(4 - 1) = (1 + 2)$" is interpreted as true
- $\neg, \wedge, \vee, \rightarrow, \leftrightarrow, \oplus$ interpreted by $\mathcal{I}$ as in PL

FOL: Semantics [cont.]

Interpretation of formulas

- $\mathcal{I}$ maps (ground) formulas into truth values
- An atomic formula $P(t_1, ..., t_k)$ is true in $\mathcal{I}$ iff the objects into which the terms t_1,...t_k are mapped by $\mathcal{I}$ comply to the relation into which P is mapped
  - $[P(t_1, ..., t_k)]^{\mathcal{I}}$ is true iff $\langle [t_1]^{\mathcal{I}}, ..., [t_k]^{\mathcal{I}} \rangle \in [P]^{\mathcal{I}}$
  - Ex: if "Me, Mother, Father, married" are interpreted as usual, then "Married(Mother(Me),Father(Me))" is interpreted as true
  - Ex: if "$+, -, >, 0, 1, 2, 3, 4$" are interpreted as usual, then "$(4 - 0) > (1 + 2)$" is interpreted as true
- An atomic formula $t_1 = t_2$ is true in $\mathcal{I}$ iff the terms t_1, t_2 are mapped by $\mathcal{I}$ into the same domain element
  - $[t_1 = t_2]^{\mathcal{I}}$ is true iff $[t_1]^{\mathcal{I}}$ same as $[t_2]^{\mathcal{I}}$
  - Ex: if "Mother" is interpreted as usual, Richard, John are brothers, then "Mother(Richard)=Mother(John))" is interpreted as true
  - Ex: if "$+, -, 0, 1, 2, 3, 4$" are interpreted as usual, then "$(4 - 1) = (1 + 2)$" is interpreted as true
- $\neg, \wedge, \vee, \rightarrow, \leftrightarrow, \oplus$ interpreted by $\mathcal{I}$ as in PL

FOL: Semantics [cont.]

Interpretation of formulas

- $\mathcal{I}$ maps (ground) formulas into truth values
- An atomic formula $P(t_1, ..., t_k)$ is true in $\mathcal{I}$ iff the objects into which the terms t_1,...t_k are mapped by $\mathcal{I}$ comply to the relation into which P is mapped
  - $[P(t_1, ..., t_k)]^{\mathcal{I}}$ is true iff $\langle [t_1]^{\mathcal{I}}, ..., [t_k]^{\mathcal{I}} \rangle \in [P]^{\mathcal{I}}$
  - Ex: if "Me, Mother, Father, married" are interpreted as usual, then "Married(Mother(Me),Father(Me))" is interpreted as true
  - Ex: if "$+, -, >, 0, 1, 2, 3, 4$" are interpreted as usual, then "$(4 - 0) > (1 + 2)$" is interpreted as true
- An atomic formula $t_1 = t_2$ is true in $\mathcal{I}$ iff the terms t_1, t_2 are mapped by $\mathcal{I}$ into the same domain element
  - $[t_1 = t_2]^{\mathcal{I}}$ is true iff $[t_1]^{\mathcal{I}}$ same as $[t_2]^{\mathcal{I}}$
  - Ex: if "Mother" is interpreted as usual, Richard, John are brothers, then "Mother(Richard)=Mother(John))" is interpreted as true
  - Ex: if "$+, -, 0, 1, 2, 3, 4$" are interpreted as usual, then "$(4 - 1) = (1 + 2)$" is interpreted as true
- $\neg, \wedge, \vee, \rightarrow, \leftrightarrow, \oplus$ interpreted by $\mathcal{I}$ as in PL

FOL: Semantics [cont.]

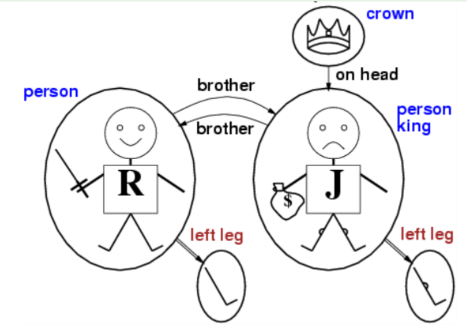
Interpretation of formulas

- $\mathcal{I}$ maps (ground) formulas into truth values
- An atomic formula $P(t_1, ..., t_k)$ is true in $\mathcal{I}$ iff the objects into which the terms t_1,...t_k are mapped by $\mathcal{I}$ comply to the relation into which P is mapped
  - $[P(t_1, ..., t_k)]^{\mathcal{I}}$ is true iff $\langle [t_1]^{\mathcal{I}}, ..., [t_k]^{\mathcal{I}} \rangle \in [P]^{\mathcal{I}}$
  - Ex: if "Me, Mother, Father, married" are interpreted as usual, then "Married(Mother(Me),Father(Me))" is interpreted as true
  - Ex: if "$+, -, >, 0, 1, 2, 3, 4$" are interpreted as usual, then "$(4-0) > (1+2)$" is interpreted as true
- An atomic formula $t_1 = t_2$ is true in $\mathcal{I}$ iff the terms t_1, t_2 are mapped by $\mathcal{I}$ into the same domain element
  - $[t_1 = t_2]^{\mathcal{I}}$ is true iff $[t_1]^{\mathcal{I}}$ same as $[t_2]^{\mathcal{I}}$
  - Ex: if "Mother" is interpreted as usual, Richard, John are brothers, then "Mother(Richard)=Mother(John))" is interpreted as true
  - Ex: if "$+, -, 0, 1, 2, 3, 4$" are interpreted as usual, then "$(4-1) = (1+2)$" is interpreted as true
- $\neg, \wedge, \vee, \rightarrow, \leftrightarrow, \oplus$ interpreted by $\mathcal{I}$ as in PL

Models for FOL: Example

Richard Lionhearth and John Lackland

- $\mathcal{D}$: domain at right
- $\mathcal{I}$: s.t.
  - $[Richard]^{\mathcal{I}}$: Richard the Lionhearth
  - $[John]^{\mathcal{I}}$: evil King John
  - $[Brother]^{\mathcal{I}}$: brotherhood
- $[Brother(Richard, John)]^{\mathcal{I}}$ is true
- $[Leftleg]^{\mathcal{I}}$ maps any individual to his left leg
- ...



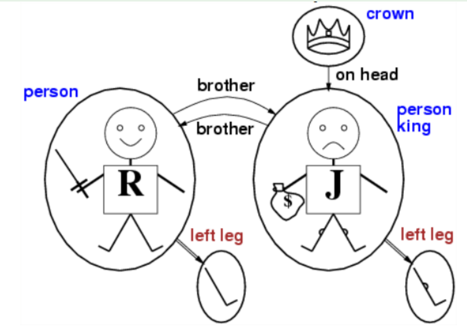
(© S. Russell & P. Norwig, AIMA)

- $[f]^{\mathcal{I}}$ total: must provide an output for every input
- e.g.: $[Leftleg(crown)]^{\mathcal{I}}$?
- possible solution: assume "null" object ($[Leftleg(crown) = null]^{\mathcal{I}}$

Models for FOL: Example

Richard Lionhearth and John Lackland

- $\mathcal{D}$: domain at right
- $\mathcal{I}$: s.t.
  - $[Richard]^{\mathcal{I}}$: Richard the Lionhearth
  - $[John]^{\mathcal{I}}$: evil King John
  - $[Brother]^{\mathcal{I}}$: brotherhood
- $[Brother(Richard, John)]^{\mathcal{I}}$ is true
- $[Leftleg]^{\mathcal{I}}$ maps any individual to his left leg
- ...



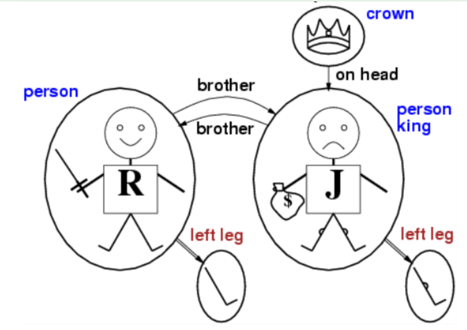
(© S. Russell & P. Norwig, AIMA)

- $[f]^{\mathcal{I}}$ total: must provide an output for every input
- e.g.: $[Leftleg(crown)]^{\mathcal{I}}$?
- possible solution: assume "null" object ($[Leftleg(crown) = null]^{\mathcal{I}}$)

Models for FOL: Example

Richard Lionhearth and John Lackland

- $\mathcal{D}$: domain at right
- $\mathcal{I}$: s.t.
  - $[Richard]^{\mathcal{I}}$: Richard the Lionhearth
  - $[John]^{\mathcal{I}}$: evil King John
  - $[Brother]^{\mathcal{I}}$: brotherhood
- $[Brother(Richard, John)]^{\mathcal{I}}$ is true
- $[Leftleg]^{\mathcal{I}}$ maps any individual to his left leg
- ...



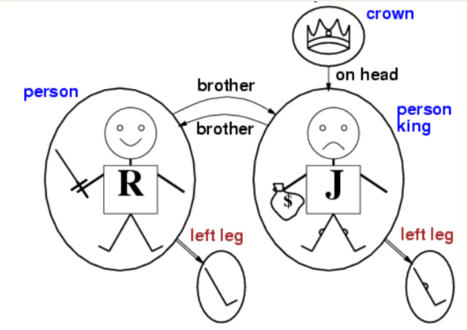
(© S. Russell & P. Norwig, AIMA)

- $[f]^{\mathcal{I}}$ total: must provide an output for every input
- e.g.: $[Leftleg(crown)]^{\mathcal{I}}$?
- possible solution: assume “null” object ($[Leftleg(crown) = null]^{\mathcal{I}}$)

Models for FOL: Example

Richard Lionhearth and John Lackland

- $\mathcal{D}$: domain at right
- $\mathcal{I}$: s.t.
  - $[Richard]^{\mathcal{I}}$: Richard the Lionhearth
  - $[John]^{\mathcal{I}}$: evil King John
  - $[Brother]^{\mathcal{I}}$: brotherhood
- $[Brother(Richard, John)]^{\mathcal{I}}$ is true
- $[Leftleg]^{\mathcal{I}}$ maps any individual to his left leg
- ...



(© S. Russell & P. Norwig, AIMA)

- $[f]^{\mathcal{I}}$ total: must provide an output for every input
- e.g.: $[Leftleg(crown)]^{\mathcal{I}}$?
- possible solution: assume "null" object ($[Leftleg(crown) = null]^{\mathcal{I}}$)

Universal Quantification

- $\forall x.\alpha(x, ...)$ (x variable, typically occurs in x)
  - ex: $\forall x.(King(x) \rightarrow Person(x))$ ("all kings are persons")
- $\forall x.\alpha(x, ...)$ true in $\mathcal{M}$ iff α true with all possible interpretations of x in $\mathcal{D}$
  - $\surd$ ex: $King(John) \rightarrow Person(John)$
  - $\surd$ ex: $King(Richard) \rightarrow Person(Richard)$
  - $\surd$ ex: $King(crown) \rightarrow Person(crown)$
  - $\surd$ ex: $King(Leftleg(John)) \rightarrow Person(Leftleg(John))$
- Roughly speaking, can be seen as a conjunction over all (typically infinite) possible instantiations of x in α

$$
\begin{array}{lll}
(King(John) & \rightarrow Person(John) &)\wedge \\
(King(Richard) & \rightarrow Person(Richard) &)\wedge \\
(King(crown) & \rightarrow Person(crown) &)\wedge \\
(King(Leftleg(John)) & \rightarrow Person(Leftleg(John)) &)\wedge \\
(King(Leftleg(Leftleg(John))) & \rightarrow Person(Leftleg(Leftleg(John))) &)\wedge \\
... & ... &
\end{array}
$$

Universal Quantification

- $\forall x.\alpha(x, ...)$ (x variable, typically occurs in x)
  - ex: $\forall x.(King(x) \rightarrow Person(x))$ (“all kings are persons”)
- $\forall x.\alpha(x, ...)$ true in $\mathcal{M}$ iff α true with all possible interpretations of x in $\mathcal{D}$
  - √ ex: $King(John) \rightarrow Person(John)$
  - √ ex: $King(Richard) \rightarrow Person(Richard)$
  - √ ex: $King(crown) \rightarrow Person(crown)$
  - √ ex: $King(Leftleg(John)) \rightarrow Person(Leftleg(John))$
- Roughly speaking, can be seen as a conjunction over all (typically infinite) possible instantiations of x in α

$$
\begin{array}{lll}
(King(John) & \rightarrow Person(John) &)\wedge \\
(King(Richard) & \rightarrow Person(Richard) &)\wedge \\
(King(crown) & \rightarrow Person(crown) &)\wedge \\
(King(Leftleg(John)) & \rightarrow Person(Leftleg(John)) &)\wedge \\
(King(Leftleg(Leftleg(John))) & \rightarrow Person(Leftleg(Leftleg(John))) &)\wedge \\
... & ... &
\end{array}
$$

Universal Quantification

- $\forall x.\alpha(x, ...)$ (x variable, typically occurs in x)
  - ex: $\forall x.(King(x) \rightarrow Person(x))$ ("all kings are persons")
- $\forall x.\alpha(x, ...)$ true in $\mathcal{M}$ iff α true with all possible interpretations of x in $\mathcal{D}$
  - $\surd$ ex: $King(John) \rightarrow Person(John)$
  - $\surd$ ex: $King(Richard) \rightarrow Person(Richard)$
  - $\surd$ ex: $King(crown) \rightarrow Person(crown)$
  - $\surd$ ex: $King(Leftleg(John)) \rightarrow Person(Leftleg(John))$
- Roughly speaking, can be seen as a conjunction over all (typically infinite) possible instantiations of x in α

$$
\begin{array}{lll}
(King(John) & \rightarrow Person(John) &)\wedge \\
(King(Richard) & \rightarrow Person(Richard) &)\wedge \\
(King(crown) & \rightarrow Person(crown) &)\wedge \\
(King(Leftleg(John)) & \rightarrow Person(Leftleg(John)) &)\wedge \\
(King(Leftleg(Leftleg(John))) & \rightarrow Person(Leftleg(Leftleg(John))) &)\wedge \\
... & ... &
\end{array}
$$

Universal Quantification

- $\forall x.\alpha(x, ...)$ (x variable, typically occurs in x)
  - ex: $\forall x.(King(x) \rightarrow Person(x))$ ("all kings are persons")
- $\forall x.\alpha(x, ...)$ true in $\mathcal{M}$ iff α true with all possible interpretations of x in $\mathcal{D}$
  - $\surd$ ex: $King(John) \rightarrow Person(John)$
  - $\surd$ ex: $King(Richard) \rightarrow Person(Richard)$
  - $\surd$ ex: $King(crown) \rightarrow Person(crown)$
  - $\surd$ ex: $King(Leftleg(John)) \rightarrow Person(Leftleg(John))$
- Roughly speaking, can be seen as a conjunction over all (typically infinite) possible instantiations of x in α

$$
\begin{array}{lll}
(King(John) & \rightarrow Person(John) &)\wedge \\
(King(Richard) & \rightarrow Person(Richard) &)\wedge \\
(King(crown) & \rightarrow Person(crown) &)\wedge \\
(King(Leftleg(John)) & \rightarrow Person(Leftleg(John)) &)\wedge \\
(King(Leftleg(Leftleg(John))) & \rightarrow Person(Leftleg(Leftleg(John))) &)\wedge \\
... & ... &
\end{array}
$$

Universal Quantification

- $\forall x.\alpha(x, ...)$ (x variable, typically occurs in x)
  - ex: $\forall x.(King(x) \rightarrow Person(x))$ ("all kings are persons")
- $\forall x.\alpha(x, ...)$ true in $\mathcal{M}$ iff α true with all possible interpretations of x in $\mathcal{D}$
  - $\surd$ ex: $King(John) \rightarrow Person(John)$
  - $\surd$ ex: $King(Richard) \rightarrow Person(Richard)$
  - $\surd$ ex: $King(crown) \rightarrow Person(crown)$
  - $\surd$ ex: $King(Leftleg(John)) \rightarrow Person(Leftleg(John))$
- Roughly speaking, can be seen as a conjunction over all (typically infinite) possible instantiations of x in α

$$
\begin{array}{lll}
(King(John) & \rightarrow Person(John) &)\wedge \\
(King(Richard) & \rightarrow Person(Richard) &)\wedge \\
(King(crown) & \rightarrow Person(crown) &)\wedge \\
(King(Leftleg(John)) & \rightarrow Person(Leftleg(John)) &)\wedge \\
(King(Leftleg(Leftleg(John))) & \rightarrow Person(Leftleg(Leftleg(John))) &)\wedge \\
\ldots & \ldots &
\end{array}
$$

Universal Quantification

- $\forall x.\alpha(x, ...)$ (x variable, typically occurs in x)
  - ex: $\forall x.(King(x) \rightarrow Person(x))$ ("all kings are persons")
- $\forall x.\alpha(x, ...)$ true in $\mathcal{M}$ iff α true with all possible interpretations of x in $\mathcal{D}$
  - $\surd$ ex: $King(John) \rightarrow Person(John)$
  - $\surd$ ex: $King(Richard) \rightarrow Person(Richard)$
  - $\surd$ ex: $King(crown) \rightarrow Person(crown)$
  - $\surd$ ex: $King(Leftleg(John)) \rightarrow Person(Leftleg(John))$
- Roughly speaking, can be seen as a conjunction over all (typically infinite) possible instantiations of x in α

$$
\begin{array}{lll}
(King(John) & \rightarrow Person(John) &)\land \\
(King(Richard) & \rightarrow Person(Richard) &)\land \\
(King(crown) & \rightarrow Person(crown) &)\land \\
(King(Leftleg(John)) & \rightarrow Person(Leftleg(John)) &)\land \\
(King(Leftleg(Leftleg(John))) & \rightarrow Person(Leftleg(Leftleg(John))) &)\land \\
\ldots & \ldots &
\end{array}
$$

Universal Quantification

- $\forall x.\alpha(x,...)$ (x variable, typically occurs in x)
  - ex: $\forall x.(King(x) \rightarrow Person(x))$ ("all kings are persons")
- $\forall x.\alpha(x,...)$ true in $\mathcal{M}$ iff α true with all possible interpretations of x in $\mathcal{D}$
  - $\surd$ ex: $King(John) \rightarrow Person(John)$
  - $\surd$ ex: $King(Richard) \rightarrow Person(Richard)$
  - $\surd$ ex: $King(crown) \rightarrow Person(crown)$
  - $\surd$ ex: $King(Leftleg(John)) \rightarrow Person(Leftleg(John))$
- Roughly speaking, can be seen as a conjunction over all (typically infinite) possible instantiations of x in α

$$
\begin{array}{lll}
(King(John) & \rightarrow Person(John) &)\wedge \\
(King(Richard) & \rightarrow Person(Richard) &)\wedge \\
(King(crown) & \rightarrow Person(crown) &)\wedge \\
(King(Leftleg(John)) & \rightarrow Person(Leftleg(John)) &)\wedge \\
(King(Leftleg(Leftleg(John))) & \rightarrow Person(Leftleg(Leftleg(John))) &)\wedge \\
... & ... &
\end{array}
$$

Universal Quantification

- $\forall x.\alpha(x,...)$ (x variable, typically occurs in x)
  - ex: $\forall x.(King(x) \rightarrow Person(x))$ ("all kings are persons")
- $\forall x.\alpha(x,...)$ true in $\mathcal{M}$ iff α true with all possible interpretations of x in $\mathcal{D}$
  - $\surd$ ex: $King(John) \rightarrow Person(John)$
  - $\surd$ ex: $King(Richard) \rightarrow Person(Richard)$
  - $\surd$ ex: $King(crown) \rightarrow Person(crown)$
  - $\surd$ ex: $King(Leftleg(John)) \rightarrow Person(Leftleg(John))$
- Roughly speaking, can be seen as a conjunction over all (typically infinite) possible instantiations of x in α

$$
\begin{array}{lll}
(King(John) & \rightarrow Person(John) &)\wedge \\
(King(Richard) & \rightarrow Person(Richard) &)\wedge \\
(King(crown) & \rightarrow Person(crown) &)\wedge \\
(King(Leftleg(John)) & \rightarrow Person(Leftleg(John)) &)\wedge \\
(King(Leftleg(Leftleg(John))) & \rightarrow Person(Leftleg(Leftleg(John))) &)\wedge \\
... & ... &
\end{array}
$$

Universal Quantification [cont.]

- One may want to restrict the domain of universal quantification to elements of some kind P
  - ex "forall kings ...", "forall integer numbers..."
- Idea: use an implication, with restrictive predicate as implicant: $\forall x.(P(x) \rightarrow \alpha(x,...))$
  - ex "$\forall x.(King(x) \rightarrow ...)$", "$\forall x.(Integer(x) \rightarrow ...)$",
- Beware of typical mistake: do not use "$\wedge$" instead of "$\rightarrow$"
  - ex: "$\forall x.(King(x) \wedge Person(x))$" means "everything/one is a King and is a Person"
- "$\forall$" distributes with "$\wedge$", but not with "$\vee$"
  - $\forall x.(P(x) \wedge Q(x))$ equivalent to $(\forall x.P(x)) \wedge (\forall x.Q(x))$
  - "Everybody is a king and is a person" same as "Everybody is a king and everybody is a person"
  - $\forall x.(P(x) \vee Q(x))$ not equivalent to $(\forall x.P(x)) \vee (\forall x.Q(x))$
  - "Everybody is a king or is a peasant" much weaker than "Everybody is a king or everybody is a peasant"

Universal Quantification [cont.]

- One may want to restrict the domain of universal quantification to elements of some kind P
  - ex "forall kings ...", "forall integer numbers..."
- Idea: use an implication, with restrictive predicate as implicant: $\forall x.(P(x) \rightarrow \alpha(x, ...))$
  - ex "$\forall x.(King(x) \rightarrow ...)$", "$\forall x.(Integer(x) \rightarrow ...)$",
- Beware of typical mistake: do not use "$\wedge$" instead of "$\rightarrow$"
  - ex: "$\forall x.(King(x) \wedge Person(x))$" means "everything/one is a King and is a Person"
- "$\forall$" distributes with "$\wedge$", but not with "$\vee$"
  - $\forall x.(P(x) \wedge Q(x))$ equivalent to $(\forall x.P(x)) \wedge (\forall x.Q(x))$
  - "Everybody is a king and is a person" same as "Everybody is a king and everybody is a person"
  - $\forall x.(P(x) \vee Q(x))$ not equivalent to $(\forall x.P(x)) \vee (\forall x.Q(x))$
  - "Everybody is a king or is a peasant" much weaker than "Everybody is a king or everybody is a peasant"

Universal Quantification [cont.]

- One may want to restrict the domain of universal quantification to elements of some kind P
  - ex "forall kings ...", "forall integer numbers..."
- Idea: use an implication, with restrictive predicate as implicant: $\forall x.(P(x) \rightarrow \alpha(x, ...))$
  - ex "$\forall x.(King(x) \rightarrow ...)$", "$\forall x.(Integer(x) \rightarrow ...)$",
- Beware of typical mistake: do not use "$\wedge$" instead of "$\rightarrow$"
  - ex: "$\forall x.(King(x) \wedge Person(x))$" means "everything/one is a King and is a Person"
- "$\forall$" distributes with "$\wedge$", but not with "$\vee$"
  - $\forall x.(P(x) \wedge Q(x))$ equivalent to $(\forall x.P(x)) \wedge (\forall x.Q(x))$
  - "Everybody is a king and is a person" same as "Everybody is a king and everybody is a person"
  - $\forall x.(P(x) \vee Q(x))$ not equivalent to $(\forall x.P(x)) \vee (\forall x.Q(x))$
  - "Everybody is a king or is a peasant" much weaker than "Everybody is a king or everybody is a peasant"

Universal Quantification [cont.]

- One may want to restrict the domain of universal quantification to elements of some kind P
  - ex “forall kings ...”, “forall integer numbers...”
- Idea: use an implication, with restrictive predicate as implicant: $\forall x.(P(x) \rightarrow \alpha(x, ...))$
  - ex “$\forall x.(King(x) \rightarrow ...)$”, “$\forall x.(Integer(x) \rightarrow ...)$”,
- Beware of typical mistake: do not use “$\wedge$” instead of “$\rightarrow$”
  - ex: “$\forall x.(King(x) \wedge Person(x))$” means “everything/one is a King and is a Person”
- “$\forall$” distributes with “$\wedge$”, but not with “$\vee$”
  - $\forall x.(P(x) \wedge Q(x))$ equivalent to $(\forall x.P(x)) \wedge (\forall x.Q(x))$
  - “Everybody is a king and is a person” same as “Everybody is a king and everybody is a person”
  - $\forall x.(P(x) \vee Q(x))$ not equivalent to $(\forall x.P(x)) \vee (\forall x.Q(x))$
  - “Everybody is a king or is a peasant” much weaker than “Everybody is a king or everybody is a peasant”

Universal Quantification [cont.]

- One may want to restrict the domain of universal quantification to elements of some kind P
  - ex "forall kings ...", "forall integer numbers..."
- Idea: use an implication, with restrictive predicate as implicant: $\forall x.(P(x) \rightarrow \alpha(x, ...))$
  - ex "$\forall x.(King(x) \rightarrow ...)$", "$\forall x.(Integer(x) \rightarrow ...)$",
- Beware of typical mistake: do not use "$\wedge$" instead of "$\rightarrow$"
  - ex: "$\forall x.(King(x) \wedge Person(x))$" means "everything/one is a King and is a Person"
- "$\forall$" distributes with "$\wedge$", but not with "$\vee$"
  - $\forall x.(P(x) \wedge Q(x))$ equivalent to $(\forall x.P(x)) \wedge (\forall x.Q(x))$
  - "Everybody is a king and is a person" same as "Everybody is a king and everybody is a person"
  - $\forall x.(P(x) \vee Q(x))$ not equivalent to $(\forall x.P(x)) \vee (\forall x.Q(x))$
  - "Everybody is a king or is a peasant" much weaker than "Everybody is a king or everybody is a peasant"

Existential Quantification

- $\exists x.\alpha(x, ...)$ (x variable, typically occurs in x)
  - ex: $\exists x.(King(x) \land Evil(x))$ ("there is one evil king")
  - pronounced "exists x s.t. ..." or "for some x ..."
- $\exists x.\alpha(x, ...)$ true in $\mathcal{M}$ iff α true with some possible interpretations of x in $\mathcal{D}$
  - × ex: $King(Richard) \land Evil(Richard)$
  - √ ex: $King(John) \land Evil(John)$
  - × ex: $King(crown) \land Evil(crown)$
  - × ex: $King(Leftleg(John)) \land Evil(Leftleg(John))$
- Roughly speaking, can be seen as a disjunction over all (typically infinite) possible instantiations of x in α

$$
\begin{array}{lll}
(King(Richard) & \land Evil(Richard) &)\lor \\
(King(John) & \land Evil(John) &)\lor \\
(King(crown) & \land Evil(crown) &)\lor \\
(King(Leftleg(John)) & \land Evil(Leftleg(John)) &)\lor \\
(King(Leftleg(Leftleg(John))) & \land Evil(Leftleg(Leftleg(John))) &)\lor \\
... & ... &
\end{array}
$$

Existential Quantification

- $\exists x.\alpha(x, ...)$ (x variable, typically occurs in x)
  - ex: $\exists x.(King(x) \wedge Evil(x))$ ("there is one evil king")
  - pronounced "exists x s.t. ..." or "for some x ..."
- $\exists x.\alpha(x, ...)$ true in $\mathcal{M}$ iff α true with some possible interpretations of x in $\mathcal{D}$
  - × ex: $King(Richard) \wedge Evil(Richard)$
  - √ ex: $King(John) \wedge Evil(John)$
  - × ex: $King(crown) \wedge Evil(crown)$
  - × ex: $King(Leftleg(John)) \wedge Evil(Leftleg(John))$
- Roughly speaking, can be seen as a disjunction over all (typically infinite) possible instantiations of x in α

$$
\begin{array}{lll}
(King(Richard) & \wedge Evil(Richard) &)\vee \\
(King(John) & \wedge Evil(John) &)\vee \\
(King(crown) & \wedge Evil(crown) &)\vee \\
(King(Leftleg(John)) & \wedge Evil(Leftleg(John)) &)\vee \\
(King(Leftleg(Leftleg(John))) & \wedge Evil(Leftleg(Leftleg(John))) &)\vee \\
\ldots & \ldots &
\end{array}
$$

Existential Quantification

- $\exists x.\alpha(x, ...)$ (x variable, typically occurs in x)
  - ex: $\exists x.(King(x) \wedge Evil(x))$ ("there is one evil king")
  - pronounced "exists x s.t. ..." or "for some x ..."
- $\exists x.\alpha(x, ...)$ true in $\mathcal{M}$ iff α true with some possible interpretations of x in $\mathcal{D}$
  - × ex: $King(Richard) \wedge Evil(Richard)$
  - √ ex: $King(John) \wedge Evil(John)$
  - × ex: $King(crown) \wedge Evil(crown)$
  - × ex: $King(Leftleg(John)) \wedge Evil(Leftleg(John))$
- Roughly speaking, can be seen as a disjunction over all (typically infinite) possible instantiations of x in α

$$
\begin{array}{lll}
(King(Richard) & \wedge Evil(Richard) &)\vee \\
(King(John) & \wedge Evil(John) &)\vee \\
(King(crown) & \wedge Evil(crown) &)\vee \\
(King(Leftleg(John)) & \wedge Evil(Leftleg(John)) &)\vee \\
(King(Leftleg(Leftleg(John))) & \wedge Evil(Leftleg(Leftleg(John))) &)\vee \\
\ldots & \ldots &
\end{array}
$$

Existential Quantification

- $\exists x.\alpha(x, ...)$ (x variable, typically occurs in x)
  - ex: $\exists x.(King(x) \land Evil(x))$ ("there is one evil king")
  - pronounced "exists x s.t. ..." or "for some x ..."
- $\exists x.\alpha(x, ...)$ true in $\mathcal{M}$ iff α true with some possible interpretations of x in $\mathcal{D}$
  - × ex: $King(Richard) \land Evil(Richard)$
  - √ ex: $King(John) \land Evil(John)$
  - × ex: $King(crown) \land Evil(crown)$
  - × ex: $King(Leftleg(John)) \land Evil(Leftleg(John))$
- Roughly speaking, can be seen as a disjunction over all (typically infinite) possible instantiations of x in α

$$
\begin{array}{lll}
(King(Richard) & \land Evil(Richard) &)\lor \\
(King(John) & \land Evil(John) &)\lor \\
(King(crown) & \land Evil(crown) &)\lor \\
(King(Leftleg(John)) & \land Evil(Leftleg(John)) &)\lor \\
(King(Leftleg(Leftleg(John))) & \land Evil(Leftleg(Leftleg(John))) &)\lor \\
\ldots & \ldots &
\end{array}
$$

Existential Quantification

- $\exists x.\alpha(x, ...)$ (x variable, typically occurs in x)
  - ex: $\exists x.(King(x) \wedge Evil(x))$ ("there is one evil king")
  - pronounced "exists x s.t. ..." or "for some x ..."
- $\exists x.\alpha(x, ...)$ true in $\mathcal{M}$ iff α true with some possible interpretations of x in $\mathcal{D}$
  - × ex: $King(Richard) \wedge Evil(Richard)$
  - √ ex: $King(John) \wedge Evil(John)$
  - × ex: $King(crown) \wedge Evil(crown)$
  - × ex: $King(Leftleg(John)) \wedge Evil(Leftleg(John))$
- Roughly speaking, can be seen as a disjunction over all (typically infinite) possible instantiations of x in α

$$
\begin{array}{lll}
(King(Richard) & \wedge Evil(Richard) &)\vee \\
(King(John) & \wedge Evil(John) &)\vee \\
(King(crown) & \wedge Evil(crown) &)\vee \\
(King(Leftleg(John)) & \wedge Evil(Leftleg(John)) &)\vee \\
(King(Leftleg(Leftleg(John))) & \wedge Evil(Leftleg(Leftleg(John))) &)\vee \\
\ldots & \ldots &
\end{array}
$$

Existential Quantification

- $\exists x.\alpha(x, ...)$ (x variable, typically occurs in x)
  - ex: $\exists x.(King(x) \land Evil(x))$ ("there is one evil king")
  - pronounced "exists x s.t. ..." or "for some x ..."
- $\exists x.\alpha(x, ...)$ true in $\mathcal{M}$ iff α true with some possible interpretations of x in $\mathcal{D}$
  - × ex: $King(Richard) \land Evil(Richard)$
  - √ ex: $King(John) \land Evil(John)$
  - × ex: $King(crown) \land Evil(crown)$
  - × ex: $King(Leftleg(John)) \land Evil(Leftleg(John))$
- Roughly speaking, can be seen as a disjunction over all (typically infinite) possible instantiations of x in α

$$
\begin{array}{lll}
(King(Richard) & \land Evil(Richard) &)\lor \\
(King(John) & \land Evil(John) &)\lor \\
(King(crown) & \land Evil(crown) &)\lor \\
(King(Leftleg(John)) & \land Evil(Leftleg(John)) &)\lor \\
(King(Leftleg(Leftleg(John))) & \land Evil(Leftleg(Leftleg(John))) &)\lor \\
\ldots & \ldots &
\end{array}
$$

Existential Quantification

- $\exists x.\alpha(x, ...)$ (x variable, typically occurs in x)
  - ex: $\exists x.(King(x) \land Evil(x))$ ("there is one evil king")
  - pronounced "exists x s.t. ..." or "for some x ..."
- $\exists x.\alpha(x, ...)$ true in $\mathcal{M}$ iff α true with some possible interpretations of x in $\mathcal{D}$
  - × ex: $King(Richard) \land Evil(Richard)$
  - √ ex: $King(John) \land Evil(John)$
  - × ex: $King(crown) \land Evil(crown)$
  - × ex: $King(Leftleg(John)) \land Evil(Leftleg(John))$
- Roughly speaking, can be seen as a disjunction over all (typically infinite) possible instantiations of x in α

$$
\begin{array}{lll}
(King(Richard) & \land Evil(Richard) &)\lor \\
(King(John) & \land Evil(John) &)\lor \\
(King(crown) & \land Evil(crown) &)\lor \\
(King(Leftleg(John)) & \land Evil(Leftleg(John)) &)\lor \\
(King(Leftleg(Leftleg(John))) & \land Evil(Leftleg(Leftleg(John))) &)\lor \\
\ldots & \ldots &
\end{array}
$$

Existential Quantification

- $\exists x.\alpha(x, ...)$ (x variable, typically occurs in x)
  - ex: $\exists x.(King(x) \wedge Evil(x))$ ("there is one evil king")
  - pronounced "exists x s.t. ..." or "for some x ..."
- $\exists x.\alpha(x, ...)$ true in $\mathcal{M}$ iff α true with some possible interpretations of x in $\mathcal{D}$
  - $\times$ ex: $King(Richard) \wedge Evil(Richard)$
  - $\surd$ ex: $King(John) \wedge Evil(John)$
  - $\times$ ex: $King(crown) \wedge Evil(crown)$
  - $\times$ ex: $King(Leftleg(John)) \wedge Evil(Leftleg(John))$
- Roughly speaking, can be seen as a disjunction over all (typically infinite) possible instantiations of x in α

$$
\begin{array}{lll}
(King(Richard) & \wedge Evil(Richard) &)\vee \\
(King(John) & \wedge Evil(John) &)\vee \\
(King(crown) & \wedge Evil(crown) &)\vee \\
(King(Leftleg(John)) & \wedge Evil(Leftleg(John)) &)\vee \\
(King(Leftleg(Leftleg(John))) & \wedge Evil(Leftleg(Leftleg(John))) &)\vee \\
... & ... &
\end{array}
$$

Existential Quantification [cont.]

- One may want to restrict the domain of existential quantification to elements of some kind P
  - ex "exists a king s.t. ...", "for some integer numbers..."
- Idea: use a conjunction with restrictive predicate: $\exists x.(P(x) \land \alpha(x, ...))$
  - ex "$\exists x.(King(x) \land ...)$", "$\exists x.(Integer(x) \land ...)$",
- Beware of typical mistake: do not use "$\rightarrow$" instead of "$\land$"
  - ex: "$\exists x.(King(x) \rightarrow Evil(x))$" means "Someone is not a king King or is evil"
- "$\exists$" distributes with "$\lor$", but not with "$\land$"
  - $\exists x.(P(x) \lor Q(x))$ equivalent to $(\exists x.P(x)) \lor (\exists x.Q(x))$
  - "Somebody is a king or is a knight" same as "Somebody is a king or somebody is a knight"
  - $\exists x.(P(x) \land Q(x))$ not equivalent to $(\exists x.P(x)) \land (\exists x.Q(x))$
  - "Somebody is a king and is evil" much stronger than "Somebody is a king and somebody is evil"

Existential Quantification [cont.]

- One may want to restrict the domain of existential quantification to elements of some kind P
  - ex "exists a king s.t. ...", "for some integer numbers..."
- Idea: use a conjunction with restrictive predicate: $\exists x.(P(x) \wedge \alpha(x, ...))$
  - ex "$\exists x.(King(x) \wedge ...)$", "$\exists x.(Integer(x) \wedge ...)$",
- Beware of typical mistake: do not use "$\rightarrow$" instead of "$\wedge$"
  - ex: "$\exists x.(King(x) \rightarrow Evil(x))$" means "Someone is not a king King or is evil"
- "$\exists$" distributes with "$\vee$", but not with "$\wedge$"
  - $\exists x.(P(x) \vee Q(x))$ equivalent to $(\exists x.P(x)) \vee (\exists x.Q(x))$
  - "Somebody is a king or is a knight" same as "Somebody is a king or somebody is a knight"
  - $\exists x.(P(x) \wedge Q(x))$ not equivalent to $(\exists x.P(x)) \wedge (\exists x.Q(x))$
  - "Somebody is a king and is evil" much stronger than "Somebody is a king and somebody is evil"

Existential Quantification [cont.]

- One may want to restrict the domain of existential quantification to elements of some kind P
  - ex "exists a king s.t. ...", "for some integer numbers..."
- Idea: use a conjunction with restrictive predicate: $\exists x.(P(x) \land \alpha(x, ...))$
  - ex "$\exists x.(King(x) \land ...)$", "$\exists x.(Integer(x) \land ...)$",
- Beware of typical mistake: do not use "$\rightarrow$" instead of "$\land$"
  - ex: "$\exists x.(King(x) \rightarrow Evil(x))$" means "Someone is not a king King or is evil"
- "$\exists$" distributes with "$\lor$", but not with "$\land$"
  - $\exists x.(P(x) \lor Q(x))$ equivalent to $(\exists x.P(x)) \lor (\exists x.Q(x))$
  - "Somebody is a king or is a knight" same as "Somebody is a king or somebody is a knight"
  - $\exists x.(P(x) \land Q(x))$ not equivalent to $(\exists x.P(x)) \land (\exists x.Q(x))$
  - "Somebody is a king and is evil" much stronger than "Somebody is a king and somebody is evil"

Existential Quantification [cont.]

- One may want to restrict the domain of existential quantification to elements of some kind P
  - ex "exists a king s.t. ...", "for some integer numbers..."
- Idea: use a conjunction with restrictive predicate: $\exists x.(P(x) \wedge \alpha(x, ...))$
  - ex "$\exists x.(King(x) \wedge ...)$", "$\exists x.(Integer(x) \wedge ...)$",
- Beware of typical mistake: do not use "$\rightarrow$" instead of "$\wedge$"
  - ex: "$\exists x.(King(x) \rightarrow Evil(x))$" means "Someone is not a king King or is evil"
- "$\exists$" distributes with "$\vee$", but not with "$\wedge$"
  - $\exists x.(P(x) \vee Q(x))$ equivalent to $(\exists x.P(x)) \vee (\exists x.Q(x))$
  - "Somebody is a king or is a knight" same as "Somebody is a king or somebody is a knight"
  - $\exists x.(P(x) \wedge Q(x))$ not equivalent to $(\exists x.P(x)) \wedge (\exists x.Q(x))$
  - "Somebody is a king and is evil" much stronger than "Somebody is a king and somebody is evil"

Existential Quantification [cont.]

- One may want to restrict the domain of existential quantification to elements of some kind P
  - ex "exists a king s.t. ...", "for some integer numbers..."
- Idea: use a conjunction with restrictive predicate: $\exists x.(P(x) \wedge \alpha(x, ...))$
  - ex "$\exists x.(King(x) \wedge ...)$", "$\exists x.(Integer(x) \wedge ...)$",
- Beware of typical mistake: do not use "$\rightarrow$" instead of "$\wedge$"
  - ex: "$\exists x.(King(x) \rightarrow Evil(x))$" means "Someone is not a king King or is evil"
- "$\exists$" distributes with "$\vee$", but not with "$\wedge$"
  - $\exists x.(P(x) \vee Q(x))$ equivalent to $(\exists x.P(x)) \vee (\exists x.Q(x))$
  - "Somebody is a king or is a knight" same as "Somebody is a king or somebody is a knight"
  - $\exists x.(P(x) \wedge Q(x))$ <u>not</u> equivalent to $(\exists x.P(x)) \wedge (\exists x.Q(x))$
  - "Somebody is a king and is evil" much stronger than "Somebody is a king and somebody is evil"

Basic Definitions and Properties

- A model $\mathcal{M} \stackrel{\text{def}}{=} \langle \mathcal{D}, \mathcal{I} \rangle$ satisfies φ ($\mathcal{M} \models \varphi$) iff $[\varphi]^{\mathcal{I}}$ is true
- $M(\varphi) \stackrel{\text{def}}{=} \{\mathcal{M} \mid \mathcal{M} \models \varphi\}$ (the set of models of φ)
- φ is satisfiable iff $\mathcal{M} \models \varphi$ for some $\mathcal{M}$ (i.e. $M(\varphi) \neq \emptyset$)
- α entails β ($\alpha \models \beta$) iff, for all $\mathcal{M}$s, $\mathcal{M} \models \alpha \Longrightarrow \mathcal{M} \models \beta$ (i.e., $M(\alpha) \subseteq M(\beta)$)
- φ is valid ($\models \varphi$) iff $\mathcal{M} \models \varphi$ forall $\mathcal{M}$s (i.e., $\mathcal{M} \in M(\varphi)$ forall $\mathcal{M}$s)
- α, β are equivalent iff $\alpha \models \beta$ and $\beta \models \alpha$

Basic Definitions and Properties

- A model $\mathcal{M} \stackrel{\text{def}}{=} \langle \mathcal{D}, \mathcal{I} \rangle$ satisfies φ ($\mathcal{M} \models \varphi$) iff $[\varphi]^{\mathcal{I}}$ is true
- $M(\varphi) \stackrel{\text{def}}{=} \{\mathcal{M} \mid \mathcal{M} \models \varphi\}$ (the set of models of φ)
- φ is satisfiable iff $\mathcal{M} \models \varphi$ for some $\mathcal{M}$ (i.e. $M(\varphi) \neq \emptyset$)
- α entails β ($\alpha \models \beta$) iff, for all $\mathcal{M}$s, $\mathcal{M} \models \alpha \Longrightarrow \mathcal{M} \models \beta$ (i.e., $M(\alpha) \subseteq M(\beta)$)
- φ is valid ($\models \varphi$) iff $\mathcal{M} \models \varphi$ forall $\mathcal{M}$s (i.e., $\mathcal{M} \in M(\varphi)$ forall $\mathcal{M}$s)
- α, β are equivalent iff $\alpha \models \beta$ and $\beta \models \alpha$

Basic Definitions and Properties

- A model $\mathcal{M} \stackrel{\text{def}}{=} \langle \mathcal{D}, \mathcal{I} \rangle$ satisfies φ ($\mathcal{M} \models \varphi$) iff $[\varphi]^{\mathcal{I}}$ is true
- $M(\varphi) \stackrel{\text{def}}{=} \{\mathcal{M} \mid \mathcal{M} \models \varphi\}$ (the set of models of φ)
- φ is satisfiable iff $\mathcal{M} \models \varphi$ for some $\mathcal{M}$ (i.e. $M(\varphi) \neq \emptyset$)
- α entails β ($\alpha \models \beta$) iff, for all $\mathcal{M}$s, $\mathcal{M} \models \alpha \Longrightarrow \mathcal{M} \models \beta$ (i.e., $M(\alpha) \subseteq M(\beta)$)
- φ is valid ($\models \varphi$) iff $\mathcal{M} \models \varphi$ forall $\mathcal{M}$s (i.e., $\mathcal{M} \in M(\varphi)$ forall $\mathcal{M}$s)
- α, β are equivalent iff $\alpha \models \beta$ and $\beta \models \alpha$

Basic Definitions and Properties

- A model $\mathcal{M} \stackrel{\text{def}}{=} \langle \mathcal{D}, \mathcal{I} \rangle$ satisfies φ ($\mathcal{M} \models \varphi$) iff $[\varphi]^{\mathcal{I}}$ is true
- $M(\varphi) \stackrel{\text{def}}{=} \{\mathcal{M} \mid \mathcal{M} \models \varphi\}$ (the set of models of φ)
- φ is satisfiable iff $\mathcal{M} \models \varphi$ for some $\mathcal{M}$ (i.e. $M(\varphi) \neq \emptyset$)
- α entails β ($\alpha \models \beta$) iff, for all $\mathcal{M}$s, $\mathcal{M} \models \alpha \Longrightarrow \mathcal{M} \models \beta$ (i.e., $M(\alpha) \subseteq M(\beta)$)
- φ is valid ($\models \varphi$) iff $\mathcal{M} \models \varphi$ forall $\mathcal{M}$s (i.e., $\mathcal{M} \in M(\varphi)$ forall $\mathcal{M}$s)
- α, β are equivalent iff $\alpha \models \beta$ and $\beta \models \alpha$

Basic Definitions and Properties

- A model $\mathcal{M} \stackrel{\text{def}}{=} \langle \mathcal{D}, \mathcal{I} \rangle$ satisfies φ ($\mathcal{M} \models \varphi$) iff $[\varphi]^{\mathcal{I}}$ is true
- $M(\varphi) \stackrel{\text{def}}{=} \{\mathcal{M} \mid \mathcal{M} \models \varphi\}$ (the set of models of φ)
- φ is satisfiable iff $\mathcal{M} \models \varphi$ for some $\mathcal{M}$ (i.e. $M(\varphi) \neq \emptyset$)
- α entails β ($\alpha \models \beta$) iff, for all $\mathcal{M}$s, $\mathcal{M} \models \alpha \Longrightarrow \mathcal{M} \models \beta$ (i.e., $M(\alpha) \subseteq M(\beta)$)
- φ is valid ($\models \varphi$) iff $\mathcal{M} \models \varphi$ forall $\mathcal{M}$s (i.e., $\mathcal{M} \in M(\varphi)$ forall $\mathcal{M}$s)
- α, β are equivalent iff $\alpha \models \beta$ and $\beta \models \alpha$

Properties & Results

Property

φ is valid iff $\neg\varphi$ is unsatisfiable

Deduction Theorem

$\alpha \models \beta$ iff $\alpha \rightarrow \beta$ is valid ($\models \alpha \rightarrow \beta$)

Corollary

$\alpha \models \beta$ iff $\alpha \wedge \neg\beta$ is unsatisfiable

Validity and entailment checking can be straightforwardly reduced to (un)satisfiability checking!

Properties & Results

Property

φ is valid iff $\neg\varphi$ is unsatisfiable

Deduction Theorem

$\alpha \models \beta$ iff $\alpha \rightarrow \beta$ is valid ($\models \alpha \rightarrow \beta$)

Corollary

$\alpha \models \beta$ iff $\alpha \wedge \neg\beta$ is unsatisfiable

Validity and entailment checking can be straightforwardly reduced to (un)satisfiability checking!

Properties & Results

Property

φ is valid iff $\neg\varphi$ is unsatisfiable

Deduction Theorem

$\alpha \models \beta$ iff $\alpha \rightarrow \beta$ is valid ($\models \alpha \rightarrow \beta$)

Corollary

$\alpha \models \beta$ iff $\alpha \wedge \neg\beta$ is unsatisfiable

Validity and entailment checking can be straightforwardly reduced to (un)satisfiability checking!

Properties & Results

Property

φ is valid iff $\neg\varphi$ is unsatisfiable

Deduction Theorem

$\alpha \models \beta$ iff $\alpha \to \beta$ is valid ($\models \alpha \to \beta$)

Corollary

$\alpha \models \beta$ iff $\alpha \wedge \neg\beta$ is unsatisfiable

Validity and entailment checking can be straightforwardly reduced to (un)satisfiability checking!

Properties of quantifiers

Notation variants: $\forall x(\forall y.\alpha) \iff \forall x\forall y.\alpha \iff \forall x, y.\alpha \iff \forall xy.\alpha$ (same with $\exists$)

- if x does not occur in φ, $\forall x.\varphi$ equivalent to $\exists x.\varphi$ equivalent to φ
- $\forall xy.P(x,y)$ equivalent to $\forall yx.P(x,y)$
  - ex: $\forall xy.(x < y)$ same as $\forall yx.(x < y)$
- $\exists xy.P(x,y)$ equivalent to $\exists yx.P(x,y)$
  - ex: $\exists xy.Twins(x,y)$ same as $\exists yx.Twins(x,y)$
- $\exists x\forall y.P(x,y)$ not equivalent to $\forall y\exists x.P(x,y)$
  - ex: $\forall y\exists x.Father(x,y)$ much weaker than $\exists x\forall y.Father(x,y)$ "everybody has a father" vs. "exists a father of everybody"
- Quantifier duality: each can be expressed using the other
  - $\forall x.\alpha$ equivalent to $\neg\exists x.\neg\alpha$ ex: $\forall x.Likes(x, Icecream)$ equivalent to $\neg\exists x.\neg Likes(x, Icecream)$
  - $\exists x.\alpha$ equivalent to $\neg\forall x.\neg\alpha$ ex: $\exists x.Likes(x, Broccoli)$ equivalent to $\neg\forall x.\neg Likes(x, Broccoli)$

Properties of quantifiers

Notation variants: $\forall x(\forall y.\alpha) \iff \forall x\forall y.\alpha \iff \forall x, y.\alpha \iff \forall xy.\alpha$ (same with $\exists$)

- if x does not occur in φ, $\forall x.\varphi$ equivalent to $\exists x.\varphi$ equivalent to φ
- $\forall xy.P(x, y)$ equivalent to $\forall yx.P(x, y)$
  - ex: $\forall xy.(x < y)$ same as $\forall yx.(x < y)$
- $\exists xy.P(x, y)$ equivalent to $\exists yx.P(x, y)$
  - ex: $\exists xy.Twins(x, y)$ same as $\exists yx.Twins(x, y)$
- $\exists x\forall y.P(x, y)$ not equivalent to $\forall y\exists x.P(x, y)$
  - ex: $\forall y\exists x.Father(x, y)$ much weaker than $\exists x\forall y.Father(x, y)$ "everybody has a father" vs. "exists a father of everybody"
- Quantifier duality: each can be expressed using the other
  - $\forall x.\alpha$ equivalent to $\neg\exists x.\neg\alpha$ ex: $\forall x.Likes(x, Icecream)$ equivalent to $\neg\exists x.\neg Likes(x, Icecream)$
  - $\exists x.\alpha$ equivalent to $\neg\forall x.\neg\alpha$ ex: $\exists x.Likes(x, Broccoli)$ equivalent to $\neg\forall x.\neg Likes(x, Broccoli)$

Properties of quantifiers

Notation variants: $\forall x(\forall y.\alpha) \iff \forall x \forall y.\alpha \iff \forall x, y.\alpha \iff \forall xy.\alpha$ (same with $\exists$)

- if x does not occur in φ, $\forall x.\varphi$ equivalent to $\exists x.\varphi$ equivalent to φ
- $\forall xy.P(x, y)$ equivalent to $\forall yx.P(x, y)$
  - ex: $\forall xy.(x < y)$ same as $\forall yx.(x < y)$
- $\exists xy.P(x, y)$ equivalent to $\exists yx.P(x, y)$
  - ex: $\exists xy.Twins(x, y)$ same as $\exists yx.Twins(x, y)$
- $\exists x \forall y.P(x, y)$ not equivalent to $\forall y \exists x.P(x, y)$
  - ex: $\forall y \exists x.Father(x, y)$ much weaker than $\exists x \forall y.Father(x, y)$ "everybody has a father" vs. "exists a father of everybody"
- Quantifier duality: each can be expressed using the other
  - $\forall x.\alpha$ equivalent to $\neg \exists x.\neg \alpha$ ex: $\forall x.Likes(x, Icecream)$ equivalent to $\neg \exists x.\neg Likes(x, Icecream)$
  - $\exists x.\alpha$ equivalent to $\neg \forall x.\neg \alpha$ ex: $\exists x.Likes(x, Broccoli)$ equivalent to $\neg \forall x.\neg Likes(x, Broccoli)$

Properties of quantifiers

Notation variants: $\forall x(\forall y.\alpha) \iff \forall x \forall y.\alpha \iff \forall x, y.\alpha \iff \forall xy.\alpha$ (same with $\exists$)

- if x does not occur in φ, $\forall x.\varphi$ equivalent to $\exists x.\varphi$ equivalent to φ
- $\forall xy.P(x, y)$ equivalent to $\forall yx.P(x, y)$
  - ex: $\forall xy.(x < y)$ same as $\forall yx.(x < y)$
- $\exists xy.P(x, y)$ equivalent to $\exists yx.P(x, y)$
  - ex: $\exists xy.Twins(x, y)$ same as $\exists yx.Twins(x, y)$
- $\exists x \forall y.P(x, y)$ not equivalent to $\forall y \exists x.P(x, y)$
  - ex: $\forall y \exists x.Father(x, y)$ much weaker than $\exists x \forall y.Father(x, y)$ "everybody has a father" vs. "exists a father of everybody"
- Quantifier duality: each can be expressed using the other
  - $\forall x.\alpha$ equivalent to $\neg\exists x.\neg\alpha$
    ex: $\forall x.Likes(x, Icecream)$ equivalent to $\neg\exists x.\neg Likes(x, Icecream)$
  - $\exists x.\alpha$ equivalent to $\neg\forall x.\neg\alpha$
    ex: $\exists x.Likes(x, Broccoli)$ equivalent to $\neg\forall x.\neg Likes(x, Broccoli)$

Properties of quantifiers

Notation variants: $\forall x(\forall y.\alpha) \iff \forall x \forall y.\alpha \iff \forall x, y.\alpha \iff \forall xy.\alpha$ (same with $\exists$)

- if x does not occur in φ, $\forall x.\varphi$ equivalent to $\exists x.\varphi$ equivalent to φ
- $\forall xy.P(x, y)$ equivalent to $\forall yx.P(x, y)$
  - ex: $\forall xy.(x < y)$ same as $\forall yx.(x < y)$
- $\exists xy.P(x, y)$ equivalent to $\exists yx.P(x, y)$
  - ex: $\exists xy.Twins(x, y)$ same as $\exists yx.Twins(x, y)$
- $\exists x \forall y.P(x, y)$ not equivalent to $\forall y \exists x.P(x, y)$
  - ex: $\forall y \exists x.Father(x, y)$ much weaker than $\exists x \forall y.Father(x, y)$ "everybody has a father" vs. "exists a father of everybody"
- Quantifier duality: each can be expressed using the other
  - $\forall x.\alpha$ equivalent to $\neg \exists x.\neg \alpha$
    ex: $\forall x.Likes(x, Icecream)$ equivalent to $\neg \exists x.\neg Likes(x, Icecream)$
  - $\exists x.\alpha$ equivalent to $\neg \forall x.\neg \alpha$
    ex: $\exists x.Likes(x, Broccoli)$ equivalent to $\neg \forall x.\neg Likes(x, Broccoli)$

Properties of quantifiers

Notation variants: $\forall x(\forall y.\alpha) \iff \forall x \forall y.\alpha \iff \forall x, y.\alpha \iff \forall xy.\alpha$ (same with $\exists$)

- if x does not occur in φ, $\forall x.\varphi$ equivalent to $\exists x.\varphi$ equivalent to φ
- $\forall xy.P(x, y)$ equivalent to $\forall yx.P(x, y)$
  - ex: $\forall xy.(x < y)$ same as $\forall yx.(x < y)$
- $\exists xy.P(x, y)$ equivalent to $\exists yx.P(x, y)$
  - ex: $\exists xy.Twins(x, y)$ same as $\exists yx.Twins(x, y)$
- $\exists x \forall y.P(x, y)$ not equivalent to $\forall y \exists x.P(x, y)$
  - ex: $\forall y \exists x.Father(x, y)$ much weaker than $\exists x \forall y.Father(x, y)$ "everybody has a father" vs. "exists a father of everybody"
- Quantifier duality: each can be expressed using the other
  - $\forall x.\alpha$ equivalent to $\neg \exists x.\neg \alpha$
    ex: $\forall x.Likes(x, Icecream)$ equivalent to $\neg \exists x.\neg Likes(x, Icecream)$
  - $\exists x.\alpha$ equivalent to $\neg \forall x.\neg \alpha$
    ex: $\exists x.Likes(x, Broccoli)$ equivalent to $\neg \forall x.\neg Likes(x, Broccoli)$

Semi-decidability of FOL

Theorem

Entailment (validity, unsatisfiability) in FOL is only semi-decidable:

- if $KB \models \alpha$, this can be checked in finite time
- if $KB \not\models \alpha$, no algorithm is guaranteed to check it in finite time

Semi-decidability of FOL

Theorem

Entailment (validity, unsatisfiability) in FOL is only semi-decidable:

- if $KB \models \alpha$, this can be checked in finite time
- if $KB \not\models \alpha$, no algorithm is guaranteed to check it in finite time

©Munch Museum, Oslo

Outline

[Recall:] Knowledge-Based Agent: General Schema

- Given a percept, the agent
  - Tells the KB of the percept at time step t
  - ASKs the KB for the best action to do at time step t
  - Tells the KB that it has in fact taken that action
- Details hidden in three functions: MAKE-PERCEPT-SENTENCE, MAKE-ACTION-QUERY, MAKE-ACTION-SENTENCE
  - construct logic sentences
  - implement the interface between sensors/actuators and KRR core
- Tell and Ask may require complex logical inference

```
function KB-AGENT(percept) returns an action
   persistent: KB, a knowledge base
               t, a counter, initially 0, indicating time

   TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))
   action ← ASK(KB, MAKE-ACTION-QUERY(t))
   TELL(KB, MAKE-ACTION-SENTENCE(action, t))
   t ← t + 1
   return action
```

FOL Knowledge-Based Agent

- We can assert FOL sentences (assertions) into the KB. Ex:
  - ex: $\text{Tell}(KB, King(John))$
  - ex: $\text{Tell}(KB, Person(Richard))$
  - ex: $\text{Tell}(KB, \forall x.(King(x) \rightarrow Person(x)))$
- We can ask queries (aka goals) to the KB. Ex:
  - ex: $\text{Ask}(KB, King(John))$
  - ex: $\text{Ask}(KB, Person(John))$
  - ex: $\text{Ask}(KB, \exists x.Person(x))$

$\Longrightarrow$ Ask(KB,α) returns true only if $KB \models \alpha$

- Other queries: AskVars, asking for variable values
  $\Longrightarrow$ returns one (or more) binding lists (aka substitutions)
  $\{var/term; var/term, ...\}$
  - ex: $\text{AskVars}(KB, \exists x.Person(x)) \Longrightarrow \{x/John\}; \{x/Richard\}$
  - typical for Horn clauses
    (e.g. $King(John) \vee King(Richard)$ would not cause a binding list)

FOL Knowledge-Based Agent

- We can assert FOL sentences (assertions) into the KB. Ex:
  - ex: Tell($KB, King(John)$)
  - ex: Tell($KB, Person(Richard)$)
  - ex: Tell($KB, \forall x.(King(x) \rightarrow Person(x))$)
- We can ask queries (aka goals) to the KB. Ex:
  - ex: Ask($KB, King(John)$)
  - ex: Ask($KB, Person(John)$)
  - ex: Ask($KB, \exists x.Person(x)$)

$\Longrightarrow$ Ask(KB,α) returns true only if $KB \models \alpha$

- Other queries: AskVars, asking for variable values
  $\Longrightarrow$ returns one (or more) binding lists (aka substitutions)
  $\{var/term; var/term, ...\}$
  - ex: AskVars($KB, \exists x.Person(x)$) $\Longrightarrow \{x/John\}; \{x/Richard\}$
  - typical for Horn clauses
    (e.g. $King(John) \vee King(Richard)$ would not cause a binding list)

FOL Knowledge-Based Agent

- We can assert FOL sentences (assertions) into the KB. Ex:
  - ex: Tell(KB, $King(John)$)
  - ex: Tell(KB, $Person(Richard)$)
  - ex: Tell(KB, $\forall x.(King(x) \rightarrow Person(x))$)
- We can ask queries (aka goals) to the KB. Ex:
  - ex: Ask(KB, $King(John)$)
  - ex: Ask(KB, $Person(John)$)
  - ex: Ask(KB, $\exists x.Person(x)$)

$\Longrightarrow$ Ask(KB,α) returns true only if $KB \models \alpha$

- Other queries: AskVars, asking for variable values
  $\Longrightarrow$ returns one (or more) binding lists (aka substitutions)
  $\{var/term; var/term, ...\}$
  - ex: AskVars(KB, $\exists x.Person(x)$) $\Longrightarrow \{x/John\}; \{x/Richard\}$
  - typical for Horn clauses
    (e.g. $King(John) \vee King(Richard)$ would not cause a binding list)

FOL Knowledge-Based Agent

- We can assert FOL sentences (assertions) into the KB. Ex:
  - ex: $\text{Tell}(KB, King(John))$
  - ex: $\text{Tell}(KB, Person(Richard))$
  - ex: $\text{Tell}(KB, \forall x.(King(x) \rightarrow Person(x)))$
- We can ask queries (aka goals) to the KB. Ex:
  - ex: $\text{Ask}(KB, King(John))$
  - ex: $\text{Ask}(KB, Person(John))$
  - ex: $\text{Ask}(KB, \exists x.Person(x))$

$\Longrightarrow$ Ask(KB,α) returns true only if $KB \models \alpha$

- Other queries: AskVars, asking for variable values
  $\Longrightarrow$ returns one (or more) binding lists (aka substitutions)
  $\{var/term; var/term, ...\}$
  - ex: $\text{AskVars}(KB, \exists x.Person(x)) \Longrightarrow \{x/John\}; \{x/Richard\}$
  - typical for Horn clauses
    (e.g. $King(John) \vee King(Richard)$ would not cause a binding list)

Example: The Kinship Domain

Domain of family relationships

Notation: "$t \neq s$" shortcut for "$\neg(t = s)$"

- Binary predicate symbols (family relationships):
  - Parent , Sibling, Brother, Sister, Child , Daughter, Son, Spouse, Wife, Husband, Grandparent, Grandchild, Cousin, Aunt, Uncle
- function symbols:
  - Mother, Father
- Knowledge base KB:
  1. $\forall x, y.(x = Mother(y) \leftrightarrow (Female(x) \wedge Parent(x, y)))$
  2. $\forall x, y.(Brother(x, y) \leftrightarrow (Male(x) \wedge Sibling(x, y)))$
  3. $\forall x, y.(Grandparent(x, y) \leftrightarrow \exists z.(Parent(x, z) \wedge Parent(z, y)))$
  4. $\forall x, y.(Sibling(x, y) \leftrightarrow ((x \neq y) \wedge \exists m, f.((m \neq f) \wedge Parent(m, x) \wedge Parent(m, y) \wedge (Parent(f, x) \wedge Parent(f, y))))$
  5. ...
- Queries inferred from KB
  - ex: (4) $\models \forall x, y.(Sibling(x, y) \leftrightarrow Sibling(y, x))$

Example: The Kinship Domain

Domain of family relationships

Notation: "$t \neq s$" shortcut for "$\neg(t = s)$"

- Binary predicate symbols (family relationships):
  - Parent , Sibling, Brother, Sister, Child , Daughter, Son, Spouse, Wife, Husband, Grandparent, Grandchild, Cousin, Aunt, Uncle
- function symbols:
  - Mother, Father
- Knowledge base KB:
  1. $\forall x, y.(x = Mother(y) \leftrightarrow (Female(x) \wedge Parent(x, y)))$
  2. $\forall x, y.(Brother(x, y) \leftrightarrow (Male(x) \wedge Sibling(x, y)))$
  3. $\forall x, y.(Grandparent(x, y) \leftrightarrow \exists z.(Parent(x, z) \wedge Parent(z, y)))$
  4. $\forall x, y.(Sibling(x, y) \leftrightarrow ((x \neq y) \wedge \exists m, f.((m \neq f) \wedge Parent(m, x) \wedge Parent(m, y) \wedge (Parent(f, x) \wedge Parent(f, y))))$
  5. ...
- Queries inferred from KB
  - ex: (4) $\models \forall x, y.(Sibling(x, y) \leftrightarrow Sibling(y, x))$

Example: The Kinship Domain

Domain of family relationships

Notation: "$t \neq s$" shortcut for "$\neg(t = s)$"

- Binary predicate symbols (family relationships):
  - Parent , Sibling, Brother, Sister, Child , Daughter, Son, Spouse, Wife, Husband, Grandparent, Grandchild, Cousin, Aunt, Uncle
- function symbols:
  - Mother, Father
- Knowledge base KB:
  1. $\forall x, y.(x = Mother(y) \leftrightarrow (Female(x) \wedge Parent(x, y)))$
  2. $\forall x, y.(Brother(x, y) \leftrightarrow (Male(x) \wedge Sibling(x, y)))$
  3. $\forall x, y.(Grandparent(x, y) \leftrightarrow \exists z.(Parent(x, z) \wedge Parent(z, y)))$
  4. $\forall x, y.(Sibling(x, y) \leftrightarrow ((x \neq y) \wedge \exists m, f.((m \neq f) \wedge Parent(m, x) \wedge Parent(m, y) \wedge (Parent(f, x) \wedge Parent(f, y))))$
  5. ...
- Queries inferred from KB
  - ex: (4) $\models \forall x, y.(Sibling(x, y) \leftrightarrow Sibling(y, x))$

Example: The Kinship Domain

Domain of family relationships

Notation: "$t \neq s$" shortcut for "$\neg(t = s)$"

- Binary predicate symbols (family relationships):
  - Parent , Sibling, Brother, Sister, Child , Daughter, Son, Spouse, Wife, Husband, Grandparent, Grandchild, Cousin, Aunt, Uncle
- function symbols:
  - Mother, Father
- Knowledge base KB:
  1. $\forall x, y.(x = Mother(y) \leftrightarrow (Female(x) \wedge Parent(x, y)))$
  2. $\forall x, y.(Brother(x, y) \leftrightarrow (Male(x) \wedge Sibling(x, y)))$
  3. $\forall x, y.(Grandparent(x, y) \leftrightarrow \exists z.(Parent(x, z) \wedge Parent(z, y)))$
  4. $\forall x, y.(Sibling(x, y) \leftrightarrow ((x \neq y) \wedge \exists m, f.((m \neq f) \wedge$ $Parent(m, x) \wedge Parent(m, y) \wedge (Parent(f, x) \wedge Parent(f, y))))$
  5. ...
- Queries inferred from KB
  - ex: (4) $\models \forall x, y.(Sibling(x, y) \leftrightarrow Sibling(y, x))$

Example: The Kinship Domain

Domain of family relationships

Notation: "$t \neq s$" shortcut for "$\neg(t = s)$"

- Binary predicate symbols (family relationships):
  - Parent , Sibling, Brother, Sister, Child , Daughter, Son, Spouse, Wife, Husband, Grandparent, Grandchild, Cousin, Aunt, Uncle
- function symbols:
  - Mother, Father
- Knowledge base KB:
  1. $\forall x, y.(x = Mother(y) \leftrightarrow (Female(x) \land Parent(x, y)))$
  2. $\forall x, y.(Brother(x, y) \leftrightarrow (Male(x) \land Sibling(x, y)))$
  3. $\forall x, y.(Grandparent(x, y) \leftrightarrow \exists z.(Parent(x, z) \land Parent(z, y)))$
  4. $\forall x, y.(Sibling(x, y) \leftrightarrow ((x \neq y) \land \exists m, f.((m \neq f) \land$ $Parent(m, x) \land Parent(m, y) \land (Parent(f, x) \land Parent(f, y))))$
  5. ...
- Queries inferred from KB
  - ex: (4) $\models \forall x, y.(Sibling(x, y) \leftrightarrow Sibling(y, x))$

Example: Integer Numbers

Peano Arithmetic

- Basic symbols
  - Unary predicate symbol: NatNum (natural number)
  - Unary function symbol: S (Successor)
  - Constant symbol: 0
- Defined symbols:
  - Binary function symbols: +,* (infix)
  - Constant symbols: 1,2,3,4,5,6,...
- Knowledge base KB:
  1. $NatNum(0)$
  2. $\forall x.(NatNum(x) \rightarrow NatNum(S(x)))$
  3. $\forall x.(NatNum(x) \rightarrow (0 \neq S(x)))$
  4. $\forall x, y.((NatNum(x) \wedge NatNum(y)) \rightarrow ((x \neq y) \rightarrow (S(x) \neq S(y))))$
  5. $\forall x.(NatNum(x) \rightarrow (x = (0 + x)))$
  6. $\forall x, y.((NatNum(x) \wedge NatNum(y)) \rightarrow (S(x) + y) = S(x + y))$
  7. $1 = S(0), 2 = S(1), 3 = S(2), \ldots$
- Queries inferred from KB
  - ex: $(4) \models \forall x, y.((x + y) = (y + x))$

Example: Integer Numbers

Peano Arithmetic

- Basic symbols
  - Unary predicate symbol: NatNum (natural number)
  - Unary function symbol: S (Successor)
  - Constant symbol: 0
- Defined symbols:
  - Binary function symbols: +,* (infix)
  - Constant symbols: 1,2,3,4,5,6,...
- Knowledge base KB:
  1. $NatNum(0)$
  2. $\forall x.(NatNum(x) \to NatNum(S(x)))$
  3. $\forall x.(NatNum(x) \to (0 \neq S(x)))$
  4. $\forall x, y.((NatNum(x) \wedge NatNum(y)) \to ((x \neq y) \to (S(x) \neq S(y))))$
  5. $\forall x.(NatNum(x) \to (x = (0 + x)))$
  6. $\forall x, y.((NatNum(x) \wedge NatNum(y)) \to (S(x) + y) = S(x + y))$
  7. $1 = S(0), 2 = S(1), 3 = S(2), \ldots$
- Queries inferred from KB
  - ex: (4) $\models \forall x, y.((x + y) = (y + x))$

Example: Integer Numbers

Peano Arithmetic

- Basic symbols
  - Unary predicate symbol: NatNum (natural number)
  - Unary function symbol: S (Successor)
  - Constant symbol: 0
- Defined symbols:
  - Binary function symbols: +,* (infix)
  - Constant symbols: 1,2,3,4,5,6,...
- Knowledge base KB:
  1. $NatNum(0)$
  2. $\forall x.(NatNum(x) \rightarrow NatNum(S(x)))$
  3. $\forall x.(NatNum(x) \rightarrow (0 \neq S(x)))$
  4. $\forall x, y.((NatNum(x) \wedge NatNum(y)) \rightarrow ((x \neq y) \rightarrow (S(x) \neq S(y))))$
  5. $\forall x.(NatNum(x) \rightarrow (x = (0 + x)))$
  6. $\forall x, y.((NatNum(x) \wedge NatNum(y)) \rightarrow (S(x) + y) = S(x + y))$
  7. $1 = S(0), 2 = S(1), 3 = S(2), \ldots$
- Queries inferred from KB
  - ex: $(4) \models \forall x, y.((x + y) = (y + x))$

Example: Integer Numbers

Peano Arithmetic

- Basic symbols
  - Unary predicate symbol: NatNum (natural number)
  - Unary function symbol: S (Successor)
  - Constant symbol: 0
- Defined symbols:
  - Binary function symbols: +,* (infix)
  - Constant symbols: 1,2,3,4,5,6,...
- Knowledge base KB:
  1. $NatNum(0)$
  2. $\forall x.(NatNum(x) \rightarrow NatNum(S(x)))$
  3. $\forall x.(NatNum(x) \rightarrow (0 \neq S(x)))$
  4. $\forall x, y.((NatNum(x) \wedge NatNum(y)) \rightarrow ((x \neq y) \rightarrow (S(x) \neq S(y))))$
  5. $\forall x.(NatNum(x) \rightarrow (x = (0 + x)))$
  6. $\forall x, y.((NatNum(x) \wedge NatNum(y)) \rightarrow (S(x) + y) = S(x + y))$
  7. $1 = S(0), 2 = S(1), 3 = S(2), ...$
- Queries inferred from KB
  - ex: $(4) \models \forall x, y.((x + y) = (y + x))$

Example: Integer Numbers

Peano Arithmetic

- Basic symbols
  - Unary predicate symbol: NatNum (natural number)
  - Unary function symbol: S (Successor)
  - Constant symbol: 0
- Defined symbols:
  - Binary function symbols: +,* (infix)
  - Constant symbols: 1,2,3,4,5,6,...
- Knowledge base KB:
  1. $NatNum(0)$
  2. $\forall x.(NatNum(x) \rightarrow NatNum(S(x)))$
  3. $\forall x.(NatNum(x) \rightarrow (0 \neq S(x)))$
  4. $\forall x, y.((NatNum(x) \wedge NatNum(y)) \rightarrow ((x \neq y) \rightarrow (S(x) \neq S(y))))$
  5. $\forall x.(NatNum(x) \rightarrow (x = (0 + x)))$
  6. $\forall x, y.((NatNum(x) \wedge NatNum(y)) \rightarrow (S(x) + y) = S(x + y))$
  7. $1 = S(0), 2 = S(1), 3 = S(2), \ldots$
- Queries inferred from KB
  - ex: $(4) \models \forall x, y.((x + y) = (y + x))$

Exercises

About the Kinship domain

- Try to add the axioms defining other predicates or functions (e.g. Brother, Sister, Child , Daughter, Son, Spouse, Wife, Husband, Grandparent, Grandchild, Cousin, Aunt, Uncle, ...)
- Add some ground atom or its negation to the KB (ex: Brother(Steve,Mary), Mary=Mother(Paul),...)
- Try to solve some query by entailment (e.g. Uncle(Steve,Paul), $\exists x.Uncle(x, Paul)$, ...)

About the Peano Arithmetic domain

- Try to add the axioms defining other predicate or functions (e.g. "$n \leq m$" or "$m * n$"", n^m)
- Add some ground atom or its negation to the KB (ex: $1 = S(0), 2 = S(1), ...$)
- Try to solve some query by entailment (e.g. $3 + 2 = 5$, $2 * 3 = 6$, ...)

Exercises

About the Kinship domain

- Try to add the axioms defining other predicates or functions (e.g. Brother, Sister, Child , Daughter, Son, Spouse, Wife, Husband, Grandparent, Grandchild, Cousin, Aunt, Uncle, ...)
- Add some ground atom or its negation to the KB (ex: Brother(Steve,Mary), Mary=Mother(Paul),...)
- Try to solve some query by entailment (e.g. Uncle(Steve,Paul), $\exists x.Uncle(x, Paul)$, ...)

About the Peano Arithmetic domain

- Try to add the axioms defining other predicate or functions (e.g. "$n \leq m$" or "$m * n$"", n^m)
- Add some ground atom or its negation to the KB (ex: $1 = S(0), 2 = S(1), ...$)
- Try to solve some query by entailment (e.g. $3 + 2 = 5$, $2 * 3 = 6$, ...)

Example: The Wumpus World

The FOL KB

- Perception: binary predicate Percept([s, b, g, b, sc],t)
  - (recall: perception is [Stench,Breeze,Glitter,Bump,Scream])
  - Stench, Breeze, Glitter, Bump, Scream constant symbols
  - time step t represented as integer
- Percepts imply facts about the current state.
  - $\forall t, s, g, m, c.(Percept([s, Breeze, g, m, c], t) \rightarrow Breeze(t))$
  - $\forall t, s, g, m, c.(Percept([s, Null, g, m, c], t) \rightarrow \neg Breeze(t))$
  - ...
- Environment:
  - Square: term (pair of integers): $[1, 2]$
  - Adjacency: binary predicate Adjacent:
    $\forall x, y, a, b.(Adjacent([x, y], [a, b]) \leftrightarrow$
    $(x = a \wedge (y = b - 1 \vee y = b + 1)) \vee (y = b \wedge (x = a - 1 \vee x = a + 1)))$
  - Position: predicate $At(Agent, s, t)$, ex: $At(Agent, [1, 1], 1)$
  - Unique position: $\forall x, s_1, s_2, t.((At(x, s_1, t) \wedge At(x, s_2, t)) \rightarrow s_1 = s_2)$
  - Wumpus: predicate $Wumpus(s)$, ex: $Wumpus([3, 1])$
  - Pits: predicate $Pit(s)$, ex: $Pit([3, 1])$

Example: The Wumpus World

The FOL KB

- Perception: binary predicate Percept([s, b, g, b, sc],t)
  - (recall: perception is [Stench,Breeze,Glitter,Bump,Scream])
  - Stench, Breeze, Glitter, Bump, Scream constant symbols
  - time step *t* represented as integer
- Percepts imply facts about the current state.
  - $\forall t, s, g, m, c.(Percept([s, Breeze, g, m, c], t) \rightarrow Breeze(t))$
  - $\forall t, s, g, m, c.(Percept([s, Null, g, m, c], t) \rightarrow \neg Breeze(t))$
  - ...
- Environment:
  - Square: term (pair of integers): $[1, 2]$
  - Adjacency: binary predicate Adjacent:
    $\forall x, y, a, b.(Adjacent([x, y], [a, b]) \leftrightarrow$
    $(x = a \wedge (y = b - 1 \vee y = b + 1)) \vee (y = b \wedge (x = a - 1 \vee x = a + 1)))$
  - Position: predicate $At(Agent, s, t)$, ex: $At(Agent, [1, 1], 1)$
  - Unique position: $\forall x, s_1, s_2, t.((At(x, s_1, t) \wedge At(x, s_2, t)) \rightarrow s_1 = s_2)$
  - Wumpus: predicate $Wumpus(s)$, ex: $Wumpus([3, 1])$
  - Pits: predicate $Pit(s)$, ex: $Pit([3, 1])$

Example: The Wumpus World

The FOL KB

- Perception: binary predicate Percept([s, b, g, b, sc],t)
  - (recall: perception is [Stench,Breeze,Glitter,Bump,Scream])
  - Stench, Breeze, Glitter, Bump, Scream constant symbols
  - time step t represented as integer
- Percepts imply facts about the current state.
  - $\forall t, s, g, m, c.(Percept([s, Breeze, g, m, c], t) \rightarrow Breeze(t))$
  - $\forall t, s, g, m, c.(Percept([s, Null, g, m, c], t) \rightarrow \neg Breeze(t))$
  - ...
- Environment:
  - Square: term (pair of integers): $[1, 2]$
  - Adjacency: binary predicate Adjacent:
    $\forall x, y, a, b.(Adjacent([x, y], [a, b]) \leftrightarrow$
    $(x = a \wedge (y = b - 1 \vee y = b + 1)) \vee (y = b \wedge (x = a - 1 \vee x = a + 1)))$
  - Position: predicate $At(Agent, s, t)$, ex: $At(Agent, [1, 1], 1)$
  - Unique position: $\forall x, s_1, s_2, t.((At(x, s_1, t) \wedge At(x, s_2, t)) \rightarrow s_1 = s_2)$
  - Wumpus: predicate $Wumpus(s)$, ex: $Wumpus([3, 1])$
  - Pits: predicate $Pit(s)$, ex: $Pit([3, 1])$

Example: The Wumpus World

The FOL KB

- Perception: binary predicate Percept([s, b, g, b, sc],t)
  - (recall: perception is [Stench,Breeze,Glitter,Bump,Scream])
  - Stench, Breeze, Glitter, Bump, Scream constant symbols
  - time step t represented as integer
- Percepts imply facts about the current state.
  - $\forall t, s, g, m, c.(Percept([s, Breeze, g, m, c], t) \rightarrow Breeze(t))$
  - $\forall t, s, g, m, c.(Percept([s, Null, g, m, c], t) \rightarrow \neg Breeze(t))$
  - ...
- Environment:
  - Square: term (pair of integers): $[1, 2]$
  - Adjacency: binary predicate Adjacent: $\forall x, y, a, b.(Adjacent([x, y], [a, b]) \leftrightarrow$ $(x = a \wedge (y = b - 1 \vee y = b + 1)) \vee (y = b \wedge (x = a - 1 \vee x = a + 1)))$
  - Position: predicate $At(Agent, s, t)$, ex: $At(Agent, [1, 1], 1)$
  - Unique position: $\forall x, s_1, s_2, t.((At(x, s_1, t) \wedge At(x, s_2, t)) \rightarrow s_1 = s_2)$
  - Wumpus: predicate $Wumpus(s)$, ex: $Wumpus([3, 1])$
  - Pits: predicate $Pit(s)$, ex: $Pit([3, 1])$

Personal Remark

- For Wumpus, AIMA suggests;
  - Wumpus: constant, ex $\forall t. At(Wumpus, [2, 2], t)$
- Simplification: assume Wumpus status does not evolve with time
  - predicate $Wumpus(s)$, ex: $Wumpus([3, 1])$
  - $\Longrightarrow$ makes inference much easier
  - if we consider the case the Wumpus is killed by arrow, then we need reintroducing the “At” formalization

Personal Remark

- For Wumpus, AIMA suggests;
  - Wumpus: constant, ex $\forall t.At(Wumpus, [2,2], t)$
- Simplification: assume Wumpus status does not evolve with time
  - predicate $Wumpus(s)$, ex: $Wumpus([3,1])$

  $\Longrightarrow$ makes inference much easier
  - if we consider the case the Wumpus is killed by arrow, then we need reintroducing the “At” formalization

Example: The Wumpus World [cont.]

The FOL KB [cont.]

- Infer properties from percepts:
  - $\forall s, t.((At(Agent, s, t) \wedge Breeze(t)) \rightarrow Breezy(s))$
  - $\forall s, t.((At(Agent, s, t) \wedge \neg Breeze(t)) \rightarrow \neg Breezy(s))$
- Infer information about pits & Wumpus
  - $\forall s.\ (Breezy(s) \leftrightarrow \exists r.(Adjacent(r, s) \wedge Pit(r)))$
  - $\forall s.\ (Stench(s) \leftrightarrow \exists r.(Adjacent(r, s) \wedge Wumpus(r)))$
- Actions: terms Turn(Right),Turn(Left),Forward,Shoot,Grab,Climb
  - reflex with internal state (ex: do we have the gold already?) $\forall t.((Glitter(t) \wedge \neg Holding(Gold, t)) \rightarrow BestAction(Grab, t))$
  - Query: $AskVars(\exists a.BestAction(a, 5)) \Longrightarrow \{a/Grab\}$
- Evolution on time: successor states:
  - $\forall t.(HaveArrow(t + 1) \leftrightarrow (HaveArrow(t) \wedge \neg Action(Shoot, t)))$

Note

$Holding(Gold, t)$ cannot be observed

$\Longrightarrow$ keeping track of change is essential (see Ch. 10-12)

Example: The Wumpus World [cont.]

The FOL KB [cont.]

- Infer properties from percepts:
  - $\forall s, t.((At(Agent, s, t) \wedge Breeze(t)) \rightarrow Breezy(s))$
  - $\forall s, t.((At(Agent, s, t) \wedge \neg Breeze(t)) \rightarrow \neg Breezy(s))$
- Infer information about pits & Wumpus
  - $\forall s.\ (Breezy(s) \leftrightarrow \exists r.(Adjacent(r, s) \wedge Pit(r)))$
  - $\forall s.\ (Stench(s) \leftrightarrow \exists r.(Adjacent(r, s) \wedge Wumpus(r)))$
- Actions: terms Turn(Right),Turn(Left),Forward,Shoot,Grab,Climb
  - reflex with internal state (ex: do we have the gold already?) $\forall t.((Glitter(t) \wedge \neg Holding(Gold, t)) \rightarrow BestAction(Grab, t))$
  - Query: $AskVars(\exists a.BestAction(a, 5)) \Longrightarrow \{a/Grab\}$
- Evolution on time: successor states:
  - $\forall t.(HaveArrow(t+1) \leftrightarrow (HaveArrow(t) \wedge \neg Action(Shoot, t)))$

Note

$Holding(Gold, t)$ cannot be observed
$\Longrightarrow$ keeping track of change is essential (see Ch. 10-12)

Example: The Wumpus World [cont.]

The FOL KB [cont.]

- Infer properties from percepts:
  - $\forall s, t.((At(Agent, s, t) \land Breeze(t)) \to Breezy(s))$
  - $\forall s, t.((At(Agent, s, t) \land \neg Breeze(t)) \to \neg Breezy(s))$
- Infer information about pits & Wumpus
  - $\forall s.\ (Breezy(s) \leftrightarrow \exists r.(Adjacent(r, s) \land Pit(r)))$
  - $\forall s.\ (Stench(s) \leftrightarrow \exists r.(Adjacent(r, s) \land Wumpus(r)))$
- Actions: terms Turn(Right),Turn(Left),Forward,Shoot,Grab,Climb
  - reflex with internal state (ex: do we have the gold already?)
    $\forall t.((Glitter(t) \land \neg Holding(Gold, t)) \to BestAction(Grab, t))$
  - Query: $AskVars(\exists a.BestAction(a, 5)) \Longrightarrow \{a/Grab\}$
- Evolution on time: successor states:
  - $\forall t.(HaveArrow(t+1) \leftrightarrow (HaveArrow(t) \land \neg Action(Shoot, t)))$

Note

$Holding(Gold, t)$ cannot be observed
$\Longrightarrow$ keeping track of change is essential (see Ch. 10-12)

Example: The Wumpus World [cont.]

The FOL KB [cont.]

- Infer properties from percepts:
  - $\forall s, t.((At(Agent, s, t) \wedge Breeze(t)) \rightarrow Breezy(s))$
  - $\forall s, t.((At(Agent, s, t) \wedge \neg Breeze(t)) \rightarrow \neg Breezy(s))$
- Infer information about pits & Wumpus
  - $\forall s.\ (Breezy(s) \leftrightarrow \exists r.(Adjacent(r, s) \wedge Pit(r)))$
  - $\forall s.\ (Stench(s) \leftrightarrow \exists r.(Adjacent(r, s) \wedge Wumpus(r)))$
- Actions: terms Turn(Right),Turn(Left),Forward,Shoot,Grab,Climb
  - reflex with internal state (ex: do we have the gold already?)
    $\forall t.((Glitter(t) \wedge \neg Holding(Gold, t)) \rightarrow BestAction(Grab, t))$
  - Query: $AskVars(\exists a.BestAction(a, 5)) \Longrightarrow \{a/Grab\}$
- Evolution on time: successor states:
  - $\forall t.(HaveArrow(t + 1) \leftrightarrow (HaveArrow(t) \wedge \neg Action(Shoot, t)))$

Note

$Holding(Gold, t)$ cannot be observed

$\Longrightarrow$ keeping track of change is essential (see Ch. 10-12)

Example: The Wumpus World [cont.]

The FOL KB [cont.]

- Infer properties from percepts:
  - $\forall s, t.((At(Agent, s, t) \land Breeze(t)) \rightarrow Breezy(s))$
  - $\forall s, t.((At(Agent, s, t) \land \neg Breeze(t)) \rightarrow \neg Breezy(s))$
- Infer information about pits & Wumpus
  - $\forall s.\ (Breezy(s) \leftrightarrow \exists r.(Adjacent(r, s) \land Pit(r)))$
  - $\forall s.\ (Stench(s) \leftrightarrow \exists r.(Adjacent(r, s) \land Wumpus(r)))$
- Actions: terms Turn(Right),Turn(Left),Forward,Shoot,Grab,Climb
  - reflex with internal state (ex: do we have the gold already?)
    $\forall t.((Glitter(t) \land \neg Holding(Gold, t)) \rightarrow BestAction(Grab, t))$
  - Query: $AskVars(\exists a.BestAction(a, 5)) \Longrightarrow \{a/Grab\}$
- Evolution on time: successor states:
  - $\forall t.(HaveArrow(t + 1) \leftrightarrow (HaveArrow(t) \land \neg Action(Shoot, t)))$

Note

$Holding(Gold, t)$ cannot be observed
$\Longrightarrow$ keeping track of change is essential (see Ch. 10-12)

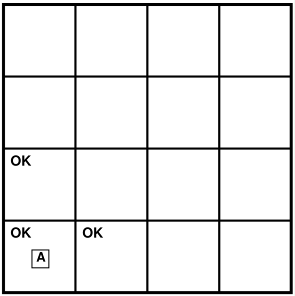
Example: Exploring the Wumpus World

KB initially contains:

$\forall x, y, a, b.(Adjacent([x, y], [a, b]) \leftrightarrow$
$(x = a \wedge (y = b - 1 \vee y = b + 1)) \vee (y = b \wedge (x = a - 1 \vee x = a + 1)))$
$\forall t, s, g, m, c.(Percept([s, Null, g, m, c], t) \rightarrow \neg Breeze(t))$
$\forall t, b, g, m, c.(Percept([Null, b, g, m, c], t) \rightarrow \neg Stench(t))$
$\forall s, t.((At(Agent, s, t) \wedge \neg Breeze(t)) \rightarrow \neg Breezy(s))$
$\forall s, t.((At(Agent, s, t) \wedge \neg Stench(t)) \rightarrow \neg Stenchy(s))$
$\forall s.\ (Breezy(s) \leftrightarrow \exists r.(Adjacent(r, s) \wedge Pit(r)))$
$\forall s.\ (Stench(s) \leftrightarrow \exists r.(Adjacent(r, s) \wedge Wumpus(r)))$
$\forall s.(Ok(s) \leftrightarrow (\neg Stenchy(s) \wedge \neg Breezy(s)))$

- A is initially in 1,1: $At(A, [1, 1], 0)$
- Perceives no stench, no breeze:

  $Tell(KB, Percept([Null, Null, Null, Null, Null], 0))$
  $\Longrightarrow \neg Breeze(0), \neg Stench(0),$
  $\Longrightarrow \neg Breezy([1, 1]), \neg Stenchy([1, 1]),$
  $\Longrightarrow \neg Pit([1, 2]), \neg Pit([2, 1]$
  $\neg Wumpus([1, 2]), \neg Wumpus([2, 1]),$
  $\Longrightarrow Ok([1, 2]), Ok([2, 1])$
  $AskVars(KB, \exists a.Action(a, 0))$
  $\Longrightarrow \{a/Move([1, 2])\}, \{a/Move([2, 1])\}$



Example: Exploring the Wumpus World

KB initially contains: $\neg Pit([1,1]), \neg Wumpus([1,1]), \ldots$

$\forall x, y, a, b.(Adjacent([x,y],[a,b]) \leftrightarrow$

$(x = a \wedge (y = b-1 \vee y = b+1)) \vee (y = b \wedge (x = a-1 \vee x = a+1)))$

$\forall t, s, g, m, c.(Percept([s, Breeze, g, m, c], t) \rightarrow Breeze(t))$

$\forall t, b, g, m, c.(Percept([Null, b, g, m, c], t) \rightarrow \neg Stench(t))$

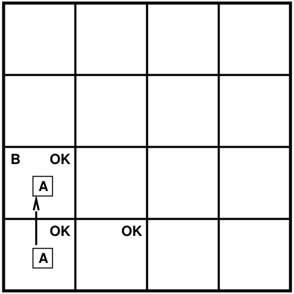
$\forall s, t.((At(Agent, s, t) \wedge Breeze(t)) \rightarrow Breezy(s))$

$\forall s, t.((At(Agent, s, t) \wedge \neg Stench(t)) \rightarrow \neg Stenchy(s))$

$\forall s.\ (Breezy(s) \leftrightarrow \exists r.(Adjacent(r, s) \wedge Pit(r)))$

$\forall s.\ (Stench(s) \leftrightarrow \exists r.(Adjacent(r, s) \wedge Wumpus(r)))$

- Agent moves to [2,1]: $At(A, [2,1], 1)$
- Perceives a breeze and no stench:
  $Tell(KB, Percept([Null, Breeze, Null, Null, Null], 1))$
  $\Longrightarrow Breeze(1), \neg Stench(1),$
  $\Longrightarrow Breezy([2,1]), \neg Stenchy([2,1]),$
  $\Longrightarrow \exists r.(Adjacent(r, [2,1]) \wedge Pit(r)),$
  $\neg Wumpus([3,1]), \neg Wumpus([2,2]),$
  $\Longrightarrow (Pit([3,1]) \vee Pit([2,2]))$
  $AskVars(KB, \exists a.Action(a, 1)) \Longrightarrow$
  $\{a/Move([1,1])\}$



Example: Exploring the Wumpus World

KB initially contains: $\neg Pit([1,1]), \neg Wumpus([1,1]), \ldots$

$\forall x, y, a, b.(Adjacent([x,y],[a,b]) \leftrightarrow$
$(x = a \wedge (y = b - 1 \vee y = b + 1)) \vee (y = b \wedge (x = a - 1 \vee x = a + 1)))$

$\forall t, s, g, m, c.(Percept([s, Breeze, g, m, c], t) \rightarrow Breeze(t))$

$\forall t, b, g, m, c.(Percept([Null, b, g, m, c], t) \rightarrow \neg Stench(t))$

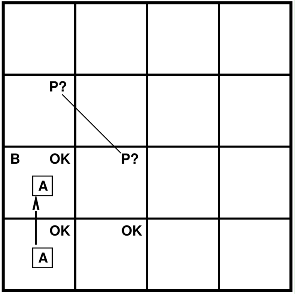
$\forall s, t.((At(Agent, s, t) \wedge Breeze(t)) \rightarrow Breezy(s))$

$\forall s, t.((At(Agent, s, t) \wedge \neg Stench(t)) \rightarrow \neg Stenchy(s))$

$\forall s.\ (Breezy(s) \leftrightarrow \exists r.(Adjacent(r, s) \wedge Pit(r)))$

$\forall s.\ (Stench(s) \leftrightarrow \exists r.(Adjacent(r, s) \wedge Wumpus(r)))$

- Agent moves to [2,1]: $At(A, [2,1], 1)$
- Perceives a breeze and no stench:
  $Tell(KB, Percept([Null, Breeze, Null, Null, Null], 1))$
  $\Longrightarrow Breeze(1), \neg Stench(1),$
  $\Longrightarrow Breezy([2,1]), \neg Stenchy([2,1]),$
  $\Longrightarrow \exists r.(Adjacent(r, [2,1]) \wedge Pit(r)),$
  $\neg Wumpus([3,1]), \neg Wumpus([2,2]),$
  $\Longrightarrow (Pit([3,1]) \vee Pit([2,2]))$
  $AskVars(KB, \exists a.Action(a, 1)) \Longrightarrow$
  $\{a/Move([1,1])\}$



Exercise

Complete the example in the FOL case (see the PL case).

Outline

Knowledge Engineering in FOL

The knowledge-engineering process

1. Identify the task (analogous to PEAS process to design agents)
   - determine what knowledge must be represented in order to connect problem instances to answers
2. Assemble the relevant knowledge (aka knowledge acquisition)
   (either by own domain knowledge or by experts interviews)
   - understand the scope of the knowledge base
   - understand how the domain actually works
3. Decide on a vocabulary of predicates, functions, and constants
   - translate relevant domain-level concepts into logic-level names
   - what should be represented as predicate/function/constant?

   $\Longrightarrow$ define the ontology of the domain
4. Encode into FOL general knowledge about the domain
   - write down the axioms for all the vocabulary terms

   $\Longrightarrow$ should enable the domain expert to check the content
5. ...

Knowledge Engineering in FOL

The knowledge-engineering process

1. Identify the task (analogous to PEAS process to design agents)
   - determine what knowledge must be represented in order to connect problem instances to answers
2. Assemble the relevant knowledge (aka knowledge acquisition)
   (either by own domain knowledge or by experts interviews)
   - understand the scope of the knowledge base
   - understand how the domain actually works
3. Decide on a vocabulary of predicates, functions, and constants
   - translate relevant domain-level concepts into logic-level names
   - what should be represented as predicate/function/constant?

   $\Longrightarrow$ define the ontology of the domain
4. Encode into FOL general knowledge about the domain
   - write down the axioms for all the vocabulary terms

   $\Longrightarrow$ should enable the domain expert to check the content
5. ...

Knowledge Engineering in FOL

The knowledge-engineering process

1. Identify the task (analogous to PEAS process to design agents)
   - determine what knowledge must be represented in order to connect problem instances to answers
2. Assemble the relevant knowledge (aka knowledge acquisition)

   (either by own domain knowledge or by experts interviews)
   - understand the scope of the knowledge base
   - understand how the domain actually works
3. Decide on a vocabulary of predicates, functions, and constants
   - translate relevant domain-level concepts into logic-level names
   - what should be represented as predicate/function/constant?

   $\Longrightarrow$ define the ontology of the domain
4. Encode into FOL general knowledge about the domain
   - write down the axioms for all the vocabulary terms

   $\Longrightarrow$ should enable the domain expert to check the content
5. ...

Knowledge Engineering in FOL

The knowledge-engineering process

1. Identify the task (analogous to PEAS process to design agents)
   - determine what knowledge must be represented in order to connect problem instances to answers
2. Assemble the relevant knowledge (aka knowledge acquisition)

   (either by own domain knowledge or by experts interviews)
   - understand the scope of the knowledge base
   - understand how the domain actually works
3. Decide on a vocabulary of predicates, functions, and constants
   - translate relevant domain-level concepts into logic-level names
   - what should be represented as predicate/function/constant?

   $\Longrightarrow$ define the ontology of the domain
4. Encode into FOL general knowledge about the domain
   - write down the axioms for all the vocabulary terms

   $\Longrightarrow$ should enable the domain expert to check the content
5. ...

Knowledge Engineering in FOL

The knowledge-engineering process

1. Identify the task (analogous to PEAS process to design agents)
   - determine what knowledge must be represented in order to connect problem instances to answers
2. Assemble the relevant knowledge (aka knowledge acquisition)

   (either by own domain knowledge or by experts interviews)
   - understand the scope of the knowledge base
   - understand how the domain actually works
3. Decide on a vocabulary of predicates, functions, and constants
   - translate relevant domain-level concepts into logic-level names
   - what should be represented as predicate/function/constant?

   $\Longrightarrow$ define the ontology of the domain
4. Encode into FOL general knowledge about the domain
   - write down the axioms for all the vocabulary terms

   $\Longrightarrow$ should enable the domain expert to check the content
5. ...

Knowledge Engineering in FOL [cont.]

The knowledge-engineering process [cont.]

4. ...
5. Encode into FOL a description of the specific problem instance
   (straightforward iff the ontology is well-conceived)
   - mostly assertions of (possibly negated) ground atomic formulas
   - for a logical agent, problem instances are supplied by the sensors
   - general knowledge base is supplied with additional sentences
6. Pose queries to the inference procedure and get answers
   - the final outcome
   - check the queries
7. Debug the knowledge base
   - detect un-answered/wrong queries
   - identify too-weak or missing axioms by backward-analysis

No need for writing an application-specific solution algorithm!

Knowledge Engineering in FOL [cont.]

The knowledge-engineering process [cont.]

4. ...
5. Encode into FOL a description of the specific problem instance
   (straightforward iff the ontology is well-conceived)
   - mostly assertions of (possibly negated) ground atomic formulas
   - for a logical agent, problem instances are supplied by the sensors
   - general knowledge base is supplied with additional sentences
6. Pose queries to the inference procedure and get answers
   - the final outcome
   - check the queries
7. Debug the knowledge base
   - detect un-answered/wrong queries
   - identify too-weak or missing axioms by backward-analysis

No need for writing an application-specific solution algorithm!

Knowledge Engineering in FOL [cont.]

The knowledge-engineering process [cont.]

4. ...
5. Encode into FOL a description of the specific problem instance
   (straightforward iff the ontology is well-conceived)
   - mostly assertions of (possibly negated) ground atomic formulas
   - for a logical agent, problem instances are supplied by the sensors
   - general knowledge base is supplied with additional sentences
6. Pose queries to the inference procedure and get answers
   - the final outcome
   - check the queries
7. Debug the knowledge base
   - detect un-answered/wrong queries
   - identify too-weak or missing axioms by backward-analysis

No need for writing an application-specific solution algorithm!

Knowledge Engineering in FOL [cont.]

The knowledge-engineering process [cont.]

4. ...
5. Encode into FOL a description of the specific problem instance
   (straightforward iff the ontology is well-conceived)
   - mostly assertions of (possibly negated) ground atomic formulas
   - for a logical agent, problem instances are supplied by the sensors
   - general knowledge base is supplied with additional sentences
6. Pose queries to the inference procedure and get answers
   - the final outcome
   - check the queries
7. Debug the knowledge base
   - detect un-answered/wrong queries
   - identify too-weak or missing axioms by backward-analysis

No need for writing an application-specific solution algorithm!

Knowledge Engineering in FOL [cont.]

The knowledge-engineering process [cont.]

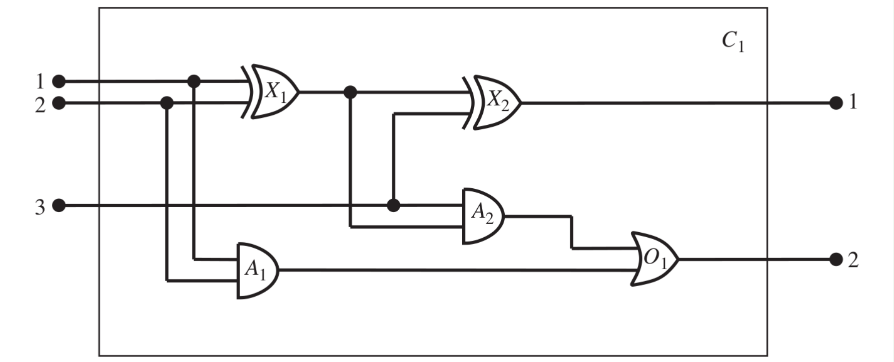
4. ...
5. Encode into FOL a description of the specific problem instance
   (straightforward iff the ontology is well-conceived)
   - mostly assertions of (possibly negated) ground atomic formulas
   - for a logical agent, problem instances are supplied by the sensors
   - general knowledge base is supplied with additional sentences
6. Pose queries to the inference procedure and get answers
   - the final outcome
   - check the queries
7. Debug the knowledge base
   - detect un-answered/wrong queries
   - identify too-weak or missing axioms by backward-analysis

No need for writing an application-specific solution algorithm!

Example: The Electronic Circuits Domain

Task: Develop (an ontology and) a knowledge base allowing to reason about digital circuits (e.g., that shown in Figure)

- Ex: One-bit full adder:
  - first two inputs are to be added, the third input is a carry bit
  - first output is the sum, the second output is a carry bit

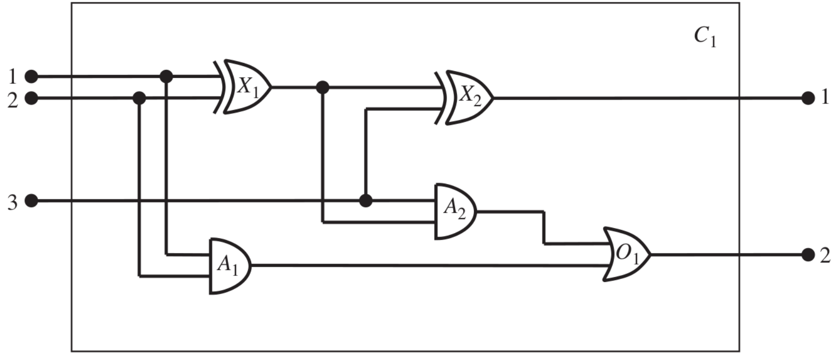


(© S. Russell & P. Norwig, AIMA)

Example: The Electronic Circuits Domain

Task: Develop (an ontology and) a knowledge base allowing to reason about digital circuits (e.g., that shown in Figure)

- Ex: One-bit full adder:
  - first two inputs are to be added, the third input is a carry bit
  - first output is the sum, the second output is a carry bit



(© S. Russell & P. Norwig, AIMA)

Example: The Electronic Circuits Domain [cont.]

1. Identify the task
   - At the highest level, analyze the circuit's functionality
   - ex: does the circuit contain feedback loops?
   - ...
2. Assemble the relevant knowledge
   - signals flow along wires to the input terminals of gates
   - each gate produces a signal on the output
   - AND, OR, XOR gates have two inputs, NOT gates have one
   - ...
3. Decide on a vocabulary of predicates, functions, and constants
   - e.g. each gate instance represented as constant (ex "X_1")
   - each gate type represented as constant (ex "*AND*")
   - a function Type (ex: $Type(X_1) = XOR$)
   - gate terminals represented as integer constants,
   - two functions In, Out, and one predicate *Connected* (ex: $Connected(In(1, X_1), In(1, A_2))$),
   - two values 0,1, a predicate Signal(t) (ex: $Signal(In(1, X_1)) = 1$)
   - ...

Example: The Electronic Circuits Domain [cont.]

1. Identify the task
   - At the highest level, analyze the circuit's functionality
   - ex: does the circuit contain feedback loops?
   - ...
2. Assemble the relevant knowledge
   - signals flow along wires to the input terminals of gates
   - each gate produces a signal on the output
   - AND, OR, XOR gates have two inputs, NOT gates have one
   - ...
3. Decide on a vocabulary of predicates, functions, and constants
   - e.g. each gate instance represented as constant (ex "X_1")
   - each gate type represented as constant (ex "*AND*")
   - a function Type (ex: $Type(X_1) = XOR$)
   - gate terminals represented as integer constants,
   - two functions In, Out, and one predicate *Connected* (ex: $Connected(In(1, X_1), In(1, A_2))$),
   - two values 0,1, a predicate Signal(t) (ex: $Signal(In(1, X_1)) = 1$)
   - ...

Example: The Electronic Circuits Domain [cont.]

1. Identify the task
   - At the highest level, analyze the circuit's functionality
   - ex: does the circuit contain feedback loops?
   - ...
2. Assemble the relevant knowledge
   - signals flow along wires to the input terminals of gates
   - each gate produces a signal on the output
   - AND, OR, XOR gates have two inputs, NOT gates have one
   - ...
3. Decide on a vocabulary of predicates, functions, and constants
   - e.g. each gate instance represented as constant (ex "X_1")
   - each gate type represented as constant (ex "*AND*")
   - a function Type (ex: $Type(X_1) = XOR$)
   - gate terminals represented as integer constants,
   - two functions In, Out, and one predicate *Connected* (ex: $Connected(In(1, X_1), In(1, A_2))$),
   - two values 0,1, a predicate Signal(t) (ex: $Signal(In(1, X_1)) = 1$)
   - ...

Example: The Electronic Circuits Domain [cont.]

1. Identify the task
   - At the highest level, analyze the circuit's functionality
   - ex: does the circuit contain feedback loops?
   - ...
2. Assemble the relevant knowledge
   - signals flow along wires to the input terminals of gates
   - each gate produces a signal on the output
   - AND, OR, XOR gates have two inputs, NOT gates have one
   - ...
3. Decide on a vocabulary of predicates, functions, and constants
   - e.g. each gate instance represented as constant (ex "X_1")
   - each gate type represented as constant (ex "*AND*")
   - a function Type (ex: $Type(X_1) = XOR$)
   - gate terminals represented as integer constants,
   - two functions In, Out, and one predicate *Connected* (ex: $Connected(In(1, X_1), In(1, A_2))$),
   - two values 0,1, a predicate Signal(t) (ex: $Signal(In(1, X_1)) = 1$)
   - ...

Example: The Electronic Circuits Domain [cont.]

4. Encode general knowledge about the domain

$$\forall t_1, t_2.((Terminal(t_1) \land Terminal(t_2) \land Connected(t_1, t_2))) \rightarrow (Signal(t_1) = Signal(t_2))$$

$$\forall t.(Terminal(t) \rightarrow ((Signal(t) = 1) \lor (Signal(t) = 0)))$$

$$\forall t_1, t_2.(Connected(t_1, t_2) \leftrightarrow Connected(t_2, t_1))$$

$$\forall g.(Gate(g) \rightarrow ((Type(g) = AND) \lor (Type(g) = OR) \lor (Type(g) = XOR) \lor (Type(g) = NOT)))$$

$$\forall g.((Gate(g) \land Type(g) = AND) \rightarrow ((Signal(Out(1, g)) = 0) \leftrightarrow \exists n.(Signal(In(n, g)) = 0)))$$

... analogous definitions for OR, XOR, NOT

$$\forall g.((Gate(g) \land (Type(g) = NOT)) \rightarrow Arity(g, 1, 1))$$

$$\forall g.((Gate(g) \land ((Type(g) = AND) \lor (Type(g) = OR) \lor (Type(g) = XOR))) \rightarrow Arity(g, 2, 1))$$

$$\forall c, i, j.((Circuit(c) \land Arity(c, i, j)) \rightarrow \forall n.((n \leq i \rightarrow Terminal(In(c, n))) \land (n > i \rightarrow In(c, n) = Nothing)) \land \forall n.((n \leq j \rightarrow Terminal(Out(c, n))) \land (n > j \rightarrow Out(c, n) = Nothing)))$$

$$\forall g, t.((Gate(g) \land Terminal(t)) \rightarrow (g \neq t \neq 1 \neq 0 \neq OR \neq AND \neq XOR \neq NOT \neq Nothing))$$

Example: The Electronic Circuits Domain [cont.]

5. Encode a description of the specific problem instance
   - $Circuit(C_1) \land Arity(C_1, 3, 2) \land$
     $Gate(X_1) \land Type(X_1) = XOR \land Gate(X_2) \land Type(X_2) = XOR \land ... \land$
     $Gate(O_1) \land Type(O_1) = OR$
   - $Connected(Out(1, X_1), In(1, X_2)) \land$
     $... \land$
     $Connected(In(3, C_1), In(1, A_2))$
6. Pose queries to the inference procedure and get answers
   - Ex: Which inputs would cause the first output of C_1 (the sum bit) to be 0 and the second output of C_1 (the carry bit) to be 1?
     $AskVars(KB, \exists i_1, i_2, i_3.(Signal(In(1, C_1)) = i_1 \land$
     $Signal(In(2, C_1)) = i_2 \land Signal(In(3, C_1)) = i_3 \land$
     $Signal(Out(1, C_1)) = 0 \land Signal(Out(2, C_1)) = 1))$
     $\Longrightarrow \{i_1/1, i_2/1, i_3/0\}$ or $\{i_1/1, i_2/0, i_3/1\}$ or $\{i_1/0, i_2/1, i_3/1\}$
   - What are the possible value sets of all terminals?
     $AskVars(KB, \exists i_1, i_2, i_3, o_1, o_2.(Signal(In(1, C_1)) = i_1 \land$
     $Signal(In(2, C_1)) = i_2 \land Signal(In(3, C_1)) = i_3 \land$
     $Signal(Out(1, C_1)) = o_1 \land Signal(Out(2, C_1)) = o_2))$
     $\Longrightarrow \{i_1/1, i_2/1, i_3/1, o_1/1, o_2/1\}$ or $\{i_1/1, i_2/1, i_3/0, o_1/0, o_2/1\}$ or ...

Example: The Electronic Circuits Domain [cont.]

5. Encode a description of the specific problem instance
   - $Circuit(C_1) \land Arity(C_1, 3, 2) \land$
     $Gate(X_1) \land Type(X_1) = XOR \land Gate(X_2) \land Type(X_2) = XOR \land ... \land$
     $Gate(O_1) \land Type(O_1) = OR$
   - $Connected(Out(1, X_1), In(1, X_2)) \land$
     $... \land$
     $Connected(In(3, C_1), In(1, A_2))$
6. Pose queries to the inference procedure and get answers
   - Ex: Which inputs would cause the first output of C_1 (the sum bit) to be 0 and the second output of C_1 (the carry bit) to be 1?
     $$AskVars(KB, \exists i_1, i_2, i_3.(Signal(In(1, C_1)) = i_1 \land Signal(In(2, C_1)) = i_2 \land Signal(In(3, C_1)) = i_3 \land Signal(Out(1, C_1)) = 0 \land Signal(Out(2, C_1)) = 1))$$
     $\Longrightarrow \{i_1/1, i_2/1, i_3/0\}$ or $\{i_1/1, i_2/0, i_3/1\}$ or $\{i_1/0, i_2/1, i_3/1\}$
   - What are the possible value sets of all terminals?
     $$AskVars(KB, \exists i_1, i_2, i_3, o_1, o_2.(Signal(In(1, C_1)) = i_1 \land Signal(In(2, C_1)) = i_2 \land Signal(In(3, C_1)) = i_3 \land Signal(Out(1, C_1)) = o_1 \land Signal(Out(2, C_1)) = o_2))$$
     $\Longrightarrow \{i_1/1, i_2/1, i_3/1, o_1/1, o_2/1\}$ or $\{i_1/1, i_2/1, i_3/0, o_1/0, o_2/1\}$ or ...

Example: The Electronic Circuits Domain [cont.]

5. Encode a description of the specific problem instance
   - $Circuit(C_1) \land Arity(C_1, 3, 2) \land$
     $Gate(X_1) \land Type(X_1) = XOR \land Gate(X_2) \land Type(X_2) = XOR \land ... \land$
     $Gate(O_1) \land Type(O_1) = OR$
   - $Connected(Out(1, X_1), In(1, X_2)) \land$
     $... \land$
     $Connected(In(3, C_1), In(1, A_2))$
6. Pose queries to the inference procedure and get answers
   - Ex: Which inputs would cause the first output of C_1 (the sum bit) to be 0 and the second output of C_1 (the carry bit) to be 1?
     $AskVars(KB, \exists i_1, i_2, i_3.(Signal(In(1, C_1)) = i_1 \land$
     $Signal(In(2, C_1)) = i_2 \land Signal(In(3, C_1)) = i_3 \land$
     $Signal(Out(1, C_1)) = 0 \land Signal(Out(2, C_1)) = 1))$
     $\Longrightarrow \{i_1/1, i_2/1, i_3/0\}$ or $\{i_1/1, i_2/0, i_3/1\}$ or $\{i_1/0, i_2/1, i_3/1\}$
   - What are the possible value sets of all terminals?
     $AskVars(KB, \exists i_1, i_2, i_3, o_1, o_2.(Signal(In(1, C_1)) = i_1 \land$
     $Signal(In(2, C_1)) = i_2 \land Signal(In(3, C_1)) = i_3 \land$
     $Signal(Out(1, C_1)) = o_1 \land Signal(Out(2, C_1)) = o_2))$
     $\Longrightarrow \{i_1/1, i_2/1, i_3/1, o_1/1, o_2/1\}$ or $\{i_1/1, i_2/1, i_3/0, o_1/0, o_2/1\}$ or ...

Example: The Electronic Circuits Domain [cont.]

5. Encode a description of the specific problem instance
   - $Circuit(C_1) \land Arity(C_1, 3, 2) \land$
     $Gate(X_1) \land Type(X_1) = XOR \land Gate(X_2) \land Type(X_2) = XOR \land ... \land$
     $Gate(O_1) \land Type(O_1) = OR$
   - $Connected(Out(1, X_1), In(1, X_2)) \land$
     $... \land$
     $Connected(In(3, C_1), In(1, A_2))$
6. Pose queries to the inference procedure and get answers
   - Ex: Which inputs would cause the first output of C_1 (the sum bit) to be 0 and the second output of C_1 (the carry bit) to be 1?
     $AskVars(KB, \exists i_1, i_2, i_3.(Signal(In(1, C_1)) = i_1 \land$
     $Signal(In(2, C_1)) = i_2 \land Signal(In(3, C_1)) = i_3 \land$
     $Signal(Out(1, C_1)) = 0 \land Signal(Out(2, C_1)) = 1))$

   $\Longrightarrow \{i_1/1, i_2/1, i_3/0\}$ or $\{i_1/1, i_2/0, i_3/1\}$ or $\{i_1/0, i_2/1, i_3/1\}$
   - What are the possible value sets of all terminals?
     $AskVars(KB, \exists i_1, i_2, i_3, o_1, o_2.(Signal(In(1, C_1)) = i_1 \land$
     $Signal(In(2, C_1)) = i_2 \land Signal(In(3, C_1)) = i_3 \land$
     $Signal(Out(1, C_1)) = o_1 \land Signal(Out(2, C_1)) = o_2))$

   $\Longrightarrow \{i_1/1, i_2/1, i_3/1, o_1/1, o_2/1\}$ or $\{i_1/1, i_2/1, i_3/0, o_1/0, o_2/1\}$ or ...

Example: The Electronic Circuits Domain [cont.]

5. Encode a description of the specific problem instance
   - $Circuit(C_1) \land Arity(C_1, 3, 2) \land$
     $Gate(X_1) \land Type(X_1) = XOR \land Gate(X_2) \land Type(X_2) = XOR \land ... \land$
     $Gate(O_1) \land Type(O_1) = OR$
   - $Connected(Out(1, X_1), In(1, X_2)) \land$
     $... \land$
     $Connected(In(3, C_1), In(1, A_2))$
6. Pose queries to the inference procedure and get answers
   - Ex: Which inputs would cause the first output of C_1 (the sum bit) to be 0 and the second output of C_1 (the carry bit) to be 1?
     $AskVars(KB, \exists i_1, i_2, i_3.(Signal(In(1, C_1)) = i_1 \land$
     $Signal(In(2, C_1)) = i_2 \land Signal(In(3, C_1)) = i_3 \land$
     $Signal(Out(1, C_1)) = 0 \land Signal(Out(2, C_1)) = 1))$
   $\Longrightarrow \{i_1/1, i_2/1, i_3/0\}$ or $\{i_1/1, i_2/0, i_3/1\}$ or $\{i_1/0, i_2/1, i_3/1\}$
   - What are the possible value sets of all terminals?
     $AskVars(KB, \exists i_1, i_2, i_3, o_1, o_2.(Signal(In(1, C_1)) = i_1 \land$
     $Signal(In(2, C_1)) = i_2 \land Signal(In(3, C_1)) = i_3 \land$
     $Signal(Out(1, C_1)) = o_1 \land Signal(Out(2, C_1)) = o_2))$
   $\Longrightarrow \{i_1/1, i_2/1, i_3/1, o_1/1, o_2/1\}$ or $\{i_1/1, i_2/1, i_3/0, o_1/0, o_2/1\}$ or ...

Example: The Electronic Circuits Domain [cont.]

5. Encode a description of the specific problem instance
   - $Circuit(C_1) \wedge Arity(C_1, 3, 2) \wedge$
     $Gate(X_1) \wedge Type(X_1) = XOR \wedge Gate(X_2) \wedge Type(X_2) = XOR \wedge ... \wedge$
     $Gate(O_1) \wedge Type(O_1) = OR$
   - $Connected(Out(1, X_1), In(1, X_2)) \wedge$
     $... \wedge$
     $Connected(In(3, C_1), In(1, A_2))$
6. Pose queries to the inference procedure and get answers
   - Ex: Which inputs would cause the first output of C_1 (the sum bit) to be 0 and the second output of C_1 (the carry bit) to be 1?
     $AskVars(KB, \exists i_1, i_2, i_3.(Signal(In(1, C_1)) = i_1 \wedge$
     $Signal(In(2, C_1)) = i_2 \wedge Signal(In(3, C_1)) = i_3 \wedge$
     $Signal(Out(1, C_1)) = 0 \wedge Signal(Out(2, C_1)) = 1))$

   $\Longrightarrow \{i_1/1, i_2/1, i_3/0\}$ or $\{i_1/1, i_2/0, i_3/1\}$ or $\{i_1/0, i_2/1, i_3/1\}$
   - What are the possible value sets of all terminals?
     $AskVars(KB, \exists i_1, i_2, i_3, o_1, o_2.(Signal(In(1, C_1)) = i_1 \wedge$
     $Signal(In(2, C_1)) = i_2 \wedge Signal(In(3, C_1)) = i_3 \wedge$
     $Signal(Out(1, C_1)) = o_1 \wedge Signal(Out(2, C_1)) = o_2))$

   $\Longrightarrow \{i_1/1, i_2/1, i_3/1, o_1/1, o_2/1\}$ or $\{i_1/1, i_2/1, i_3/0, o_1/0, o_2/1\}$ or ...

Example: The Electronic Circuits Domain [cont.]

5. Encode a description of the specific problem instance
   - $Circuit(C_1) \land Arity(C_1, 3, 2) \land$
     $Gate(X_1) \land Type(X_1) = XOR \land Gate(X_2) \land Type(X_2) = XOR \land ... \land$
     $Gate(O_1) \land Type(O_1) = OR$
   - $Connected(Out(1, X_1), In(1, X_2)) \land$
     $... \land$
     $Connected(In(3, C_1), In(1, A_2))$
6. Pose queries to the inference procedure and get answers
   - Ex: Which inputs would cause the first output of C_1 (the sum bit) to be 0 and the second output of C_1 (the carry bit) to be 1?
     $AskVars(KB, \exists i_1, i_2, i_3.(Signal(In(1, C_1)) = i_1 \land$
     $Signal(In(2, C_1)) = i_2 \land Signal(In(3, C_1)) = i_3 \land$
     $Signal(Out(1, C_1)) = 0 \land Signal(Out(2, C_1)) = 1))$

   $\Longrightarrow \{i_1/1, i_2/1, i_3/0\}$ or $\{i_1/1, i_2/0, i_3/1\}$ or $\{i_1/0, i_2/1, i_3/1\}$
   - What are the possible value sets of all terminals?
     $AskVars(KB, \exists i_1, i_2, i_3, o_1, o_2.(Signal(In(1, C_1)) = i_1 \land$
     $Signal(In(2, C_1)) = i_2 \land Signal(In(3, C_1)) = i_3 \land$
     $Signal(Out(1, C_1)) = o_1 \land Signal(Out(2, C_1)) = o_2))$

   $\Longrightarrow \{i_1/1, i_2/1, i_3/1, o_1/1, o_2/1\}$ or $\{i_1/1, i_2/1, i_3/0, o_1/0, o_2/1\}$ or ...

Example: The Electronic Circuits Domain [cont.]

5. Encode a description of the specific problem instance
   - $Circuit(C_1) \land Arity(C_1, 3, 2) \land$
     $Gate(X_1) \land Type(X_1) = XOR \land Gate(X_2) \land Type(X_2) = XOR \land ... \land$
     $Gate(O_1) \land Type(O_1) = OR$
   - $Connected(Out(1, X_1), In(1, X_2)) \land$
     $... \land$
     $Connected(In(3, C_1), In(1, A_2))$
6. Pose queries to the inference procedure and get answers
   - Ex: Which inputs would cause the first output of C_1 (the sum bit) to be 0 and the second output of C_1 (the carry bit) to be 1?
     $AskVars(KB, \exists i_1, i_2, i_3.(Signal(In(1, C_1)) = i_1 \land$
     $Signal(In(2, C_1)) = i_2 \land Signal(In(3, C_1)) = i_3 \land$
     $Signal(Out(1, C_1)) = 0 \land Signal(Out(2, C_1)) = 1))$
     $\Longrightarrow \{i_1/1, i_2/1, i_3/0\}$ or $\{i_1/1, i_2/0, i_3/1\}$ or $\{i_1/0, i_2/1, i_3/1\}$
   - What are the possible value sets of all terminals?
     $AskVars(KB, \exists i_1, i_2, i_3, o_1, o_2.(Signal(In(1, C_1)) = i_1 \land$
     $Signal(In(2, C_1)) = i_2 \land Signal(In(3, C_1)) = i_3 \land$
     $Signal(Out(1, C_1)) = o_1 \land Signal(Out(2, C_1)) = o_2))$
     $\Longrightarrow \{i_1/1, i_2/1, i_3/1, o_1/1, o_2/1\}$ or $\{i_1/1, i_2/1, i_3/0, o_1/0, o_2/1\}$ or ...

Example: The Electronic Circuits Domain [cont.]

7. Debug the knowledge base
   - Suppose no output produced by previous query
   - We progressively try to restrict our analysis my more local queries, until we pinpoint the problem.
   - Ex: $\exists i_1, i_2, o.(Signal(In(1, C1)) = i_1 \land Signal(In(2, C1)) = i_2 \land Signal(Out(1, X1)) = o)$

     (see AIMA book for a detailed example)