Using First-Order Logic

Outline

I. Assertions and queries

II. Numbers and sets in FOL

III. The wumpus world in FOL

* Figures are from the textbook site unless a source is specifically cited.

I. Knowledge Engineering

A field of AI dedicated to representing information about the world in a form that can be utilized by a computer to solve complex tasks such as:

- medical diagnosis
- dialog in a natural language
- etc.

♦ Knowledge representation (logical rules, semantic nets, frames, etc.)

♦ Automated reasoning (inference engines, theorem provers, etc.)

I. Knowledge Engineering

A field of AI dedicated to representing information about the world in a form that can be utilized by a computer to solve complex tasks such as:

- medical diagnosis
- dialog in a natural language
- etc.

♦ Knowledge representation (logical rules, semantic nets, frames, etc.)

♦ Automated reasoning (inference engines, theorem provers, etc.)

A *domain* is some part of the world about which we wish to express some knowledge.

Assertions and Queries in FOL

- Add sentences, called *assertions*, to a KB using TELL.

  TELL(*KB*, *Likes*(John, *Icecream*))

  TELL(*KB*, *Father*(Zeus, Athena))

  TELL(*KB*, $\forall x \exists y$ *Brother*$(x, y) \Rightarrow$ *Sibiling*(x, y))

Assertions and Queries in FOL

- Add sentences, called *assertions*, to a KB using TELL.

  TELL(*KB*, *Likes*(John, *Icecream*))

  TELL(*KB*, *Father*(Zeus, Athena))

  TELL(*KB*, $\forall x \exists y$ *Brother*$(x, y) \Rightarrow$ *Sibiling*(x, y))

- Ask the KB questions using ASK.

  ASK(*KB*, *Likes*(John, *Icecream*))

Assertions and Queries in FOL

- Add sentences, called *assertions*, to a KB using TELL.

  TELL(*KB*, *Likes*(John, *Icecream*))

  TELL(*KB*, *Father*(Zeus, Athena))

  TELL(*KB*, $\forall x \exists y$ *Brother*$(x, y) \Rightarrow$ *Sibiling*(x, y))

- Ask the KB questions using ASK.

  ASK(*KB*, $\underbrace{\textit{Likes}(\text{John}, \textit{Icecream})}_{\textit{Query}\text{: question asked}}$)

Assertions and Queries in FOL

- Add sentences, called *assertions*, to a KB using TELL.

  TELL(*KB*, *Likes*(John, *Icecream*))

  TELL(*KB*, *Father*(Zeus, Athena))

  TELL(*KB*, $\forall x \exists y$ *Brother*$(x, y) \Rightarrow$ *Sibling*(x, y))

- Ask the KB questions using ASK.

  $\text{ASK}(KB, \underbrace{Likes(\text{John}, Icecream))}_{\textit{Query}\text{: question asked}}$

Any query is entailed by the KB should be answered affirmatively.

Substitution

Suppose another KB has the following predicates:

Bird(Swan), *Bird*(Crane), *Bird*(Parrot),

- Quantified query

  $\textsc{Ask}(KB, \exists x\ Bird(x))$

Substitution

Suppose another KB has the following predicates:

Bird(Swan), *Bird*(Crane), *Bird*(Parrot),

- Quantified query

  ASK(*KB*, $\exists x$ *Bird*(x))   returns *true*

Substitution

Suppose another KB has the following predicates:

Bird(Swan), *Bird*(Crane), *Bird*(Parrot),

- Quantified query

  ASK(*KB*, $\exists x$ *Bird*(x))   returns *true*

- To know what values of x make the sentence true

  ASKVARS(*KB*, *Bird*(x))

Substitution

Suppose another KB has the following predicates:

Bird(Swan), *Bird*(Crane), *Bird*(Parrot),

- Quantified query

  ASK(*KB*, $\exists x$ *Bird*(x)) returns *true*

- To know what values of x make the sentence true

  ASKVARS(*KB*, *Bird*(x))

  The query returns

  $\{x/$ Swan$\}$, $\{x/$ Crane$\}$, and $\{x/$ Parrot$\}$

Substitution

Suppose another KB has the following predicates:

Bird(Swan), *Bird*(Crane), *Bird*(Parrot),

- Quantified query

  ASK(*KB*, $\exists x$ *Bird*(x)) returns *true*

- To know what values of x make the sentence true

  ASKVARS(*KB*, *Bird*(x))

  The query returns

  $\underbrace{\{x/ \text{Swan}\}, \{x/ \text{Crane}\}, \text{and } \{x/ \text{Parrot}\}}_{\textit{substitution} \text{ or } \textit{binding list}}$

The Kinship Domain

Kinship relations are represented by binary predicates.

// One's mother is the person's female parent.

$\forall m, c \; \mathit{Mother}(c) = m \Leftrightarrow \mathit{Female}(m) \wedge \mathit{Parent}(m, c)$

The Kinship Domain

Kinship relations are represented by binary predicates.

// One's mother is the person's female parent.

$\forall m, c \; \mathit{Mother}(c) = m \Leftrightarrow \mathit{Female}(m) \wedge \mathit{Parent}(m, c)$

// One's husband is the person's male spouse.

$\forall w, h \; \mathit{Husband}(h, w) \Leftrightarrow \mathit{Male}(h) \wedge \mathit{Spouse}(h, w)$

The Kinship Domain

Kinship relations are represented by binary predicates.

// One's mother is the person's female parent.

$\forall m, c \; \textit{Mother}(c) = m \Leftrightarrow \textit{Female}(m) \land \textit{Parent}(m, c)$

// One's husband is the person's male spouse.

$\forall w, h \; \textit{Husband}(h, w) \Leftrightarrow \textit{Male}(h) \land \textit{Spouse}(h, w)$

// Parent and child are inverse relations.

$\forall p, c \; \textit{Parent}(p, c) \Leftrightarrow \textit{Child}(c, p)$

The Kinship Domain

Kinship relations are represented by binary predicates.

// One's mother is the person's female parent.

$\forall m, c\ \textit{Mother}(c) = m \Leftrightarrow \textit{Female}(m) \wedge \textit{Parent}(m, c)$

// One's husband is the person's male spouse.

$\forall w, h\ \textit{Husband}(h, w) \Leftrightarrow \textit{Male}(h) \wedge \textit{Spouse}(h, w)$

// Parent and child are inverse relations.

$\forall p, c\ \textit{Parent}(p, c) \Leftrightarrow \textit{Child}(c, p)$

// A grand parent is a parent of one's parent

$\forall g, c\ \textit{GrandParent}(g, c) \Leftrightarrow \exists p\ (\textit{Parent}(g, p) \wedge \textit{Parent}(p, c))$

The Kinship Domain

Kinship relations are represented by binary predicates.

// One's mother is the person's female parent.

$\forall m, c\ \textit{Mother}(c) = m \Leftrightarrow \textit{Female}(m) \land \textit{Parent}(m, c)$

// One's husband is the person's male spouse.

$\forall w, h\ \textit{Husband}(h, w) \Leftrightarrow \textit{Male}(h) \land \textit{Spouse}(h, w)$

// Parent and child are inverse relations.

$\forall p, c\ \textit{Parent}(p, c) \Leftrightarrow \textit{Child}(c, p)$

// A grand parent is a parent of one's parent

$\forall g, c\ \textit{GrandParent}(g, c) \Leftrightarrow \exists p\ (\textit{Parent}(g, p) \land \textit{Parent}(p, c))$

// A sibling is another child of one's parent

$\forall x, s\ \textit{Sibling}(x, s) \Leftrightarrow x \neq s \land \exists p\ (\textit{Parent}(p, x) \land \textit{Parent}(p, s))$

The Kinship Domain

Kinship relations are represented by binary predicates.

Axioms

// One's mother is the person's female parent.

$\forall m, c \; \textit{Mother}(c) = m \Leftrightarrow \textit{Female}(m) \wedge \textit{Parent}(m, c)$

// One's husband is the person's male spouse.

$\forall w, h \; \textit{Husband}(h, w) \Leftrightarrow \textit{Male}(h) \wedge \textit{Spouse}(h, w)$

// Parent and child are inverse relations.

$\forall p, c \; \textit{Parent}(p, c) \Leftrightarrow \textit{Child}(c, p)$

// A grand parent is a parent of one's parent

$\forall g, c \; \textit{GrandParent}(g, c) \Leftrightarrow \exists p \; (\textit{Parent}(g, p) \wedge \textit{Parent}(p, c))$

// A sibling is another child of one's parent

$\forall x, s \; \textit{Sibling}(x, s) \Leftrightarrow x \neq s \wedge \exists p \; (\textit{Parent}(p, x) \wedge \textit{Parent}(p, s))$

The Kinship Domain

Kinship relations are represented by binary predicates.

Axioms

// One's mother is the person's female parent.

$\forall m, c \; \textit{Mother}(c) = m \Leftrightarrow \textit{Female}(m) \wedge \textit{Parent}(m, c)$

// One's husband is the person's male spouse.

$\forall w, h \; \textit{Husband}(h, w) \Leftrightarrow \textit{Male}(h) \wedge \textit{Spouse}(h, w)$

// Parent and child are inverse relations.

$\forall p, c \; \textit{Parent}(p, c) \Leftrightarrow \textit{Child}(c, p)$

// A grand parent is a parent of one's parent

$\forall g, c \; \textit{GrandParent}(g, c) \Leftrightarrow \exists p \; (\textit{Parent}(g, p) \wedge \textit{Parent}(p, c))$

// A sibling is another child of one's parent

$\forall x, s \; \textit{Sibling}(x, s) \Leftrightarrow x \neq s \wedge \exists p \; (\textit{Parent}(p, x) \wedge \textit{Parent}(p, s))$

These are *definitions* in the form of $\forall x, y \; P(x, y) \Leftrightarrow \ldots$ and built upon a basic set of predicates *Child*, *Male*, *Female*, etc.

Axioms and Theorems

- *Axioms* in a domain are logical sentences that are taken to be true without being derived.

- *Theorems* are logical sentences entailed by axioms.

Axioms and Theorems

- *Axioms* in a domain are logical sentences that are taken to be true without being derived.

- *Theorems* are logical sentences entailed by axioms.

$$\forall x, y\ \mathit{Sibling}(x, y) \Leftrightarrow \mathit{Sibling}(y, x)$$

// entailed by

// $\forall x, s\ \mathit{Sibling}(x, s) \Leftrightarrow x \neq s \land \exists p\ (\mathit{Parent}(p, x) \land \mathit{Parent}(p, s))$

Axioms and Theorems

- *Axioms* in a domain are logical sentences that are taken to be true without being derived.

- *Theorems* are logical sentences entailed by axioms.

$\forall x, y$ *Sibling*$(x, y) \Leftrightarrow$ *Sibling*(y, x)

// entailed by
// $\forall x, s$ *Sibling*$(x, s) \Leftrightarrow x \neq s \wedge \exists p$ (*Parent*$(p, x) \wedge$ *Parent*$(p, s))$

ASK(*KB*, $\forall x, y$ *Sibling*$(x, y) \Leftrightarrow$ *Sibling*(y, x)) should return *true*.

Axioms and Theorems

- *Axioms* in a domain are logical sentences that are taken to be true without being derived.

- *Theorems* are logical sentences entailed by axioms.

$\forall x, y$ *Sibling*$(x, y) \Leftrightarrow$ *Sibling*(y, x)

// entailed by
// $\forall x, s$ *Sibling*$(x, s) \Leftrightarrow x \neq s \wedge \exists p$ (*Parent*$(p, x) \wedge$ *Parent*$(p, s))$

ASK$($*KB*, $\forall x, y$ *Sibling*$(x, y) \Leftrightarrow$ *Sibling*$(y, x))$ should return *true*.

- Some axioms are not definitions.

$\forall x$ *Person*$(x) \Leftrightarrow \cdots$ // no clear way to define

Axioms and Theorems

- *Axioms* in a domain are logical sentences that are taken to be true without being derived.

- *Theorems* are logical sentences entailed by axioms.

$$\forall x, y \; \textit{Sibling}(x, y) \Leftrightarrow \textit{Sibling}(y, x)$$

// entailed by

// $\forall x, s \; \textit{Sibling}(x, s) \Leftrightarrow x \neq s \land \exists p \; (\textit{Parent}(p, x) \land \textit{Parent}(p, s))$

ASK(KB, $\forall x, y \; \textit{Sibling}(x, y) \Leftrightarrow \textit{Sibling}(y, x)$) should return *true*.

- Some axioms are not definitions.

$\forall x \; \textit{Person}(x) \Leftrightarrow \cdots$ // no clear way to define

- Some predicates have no complete definitions.

Axioms and Theorems

♠ *Axioms* in a domain are logical sentences that are taken to be true without being derived.

♠ *Theorems* are logical sentences entailed by axioms.

$$\forall x, y \; Sibling(x, y) \Leftrightarrow Sibling(y, x)$$

// entailed by
// $\forall x, s \; Sibling(x, s) \Leftrightarrow x \neq s \wedge \exists p \; (Parent(p, x) \wedge Parent(p, s))$

$\textsc{Ask}(KB, \forall x, y \; Sibling(x, y) \Leftrightarrow Sibling(y, x))$ should return *true*.

♠ Some axioms are not definitions.

$\forall x \; Person(x) \Leftrightarrow \cdots$ // no clear way to define

♠ Some predicates have no complete definitions.

$\forall x \; Person(x) \Longrightarrow \cdots$ // partial specification of
$\forall x \; \cdots \Longrightarrow Person(x)$ // properties

II. Domain of Natural Numbers

Describe the theory of natural numbers using merely:

- one constant symbol, 0
- one function symbol, S (successor)
- one predicate, *NatNum*

II. Domain of Natural Numbers

Describe the theory of natural numbers using merely:

- one constant symbol, 0
- one function symbol, S (successor)
- one predicate, *NatNum*

♦ Recursive definition:

$$\mathit{NatNum}(0)$$

$$\forall n\ \mathit{NatNum}(n) \Rightarrow \mathit{NatNum}(S(n))$$

II. Domain of Natural Numbers

Describe the theory of natural numbers using merely:

- one constant symbol, 0
- one function symbol, S (successor)
- one predicate, *NatNum*

♦ Recursive definition:

$$\mathit{NatNum}(0)$$

$$\forall n\ \mathit{NatNum}(n) \Rightarrow \mathit{NatNum}(S(n))$$

♦ Axioms to constrain the successor function:

$$\forall n\ \ 0 \neq S(n)$$

$$\forall m, n\ \ \ m \neq n \Rightarrow S(m)\ \neq S(n)$$

Defining Addition

$\forall m \; NatNum(m) \Rightarrow +(0, m) = m$

Defining Addition

$$\forall m \; \mathit{NatNum}(m) \Rightarrow \underbrace{+(0, m)}_{\textit{prefix notation}} = m$$

Defining Addition

$$\forall m\ \textit{NatNum}(m) \Rightarrow \underbrace{+(0, m)}_{\textit{prefix notation}} = m$$

Use infix notation for readability.

$$\forall m\ \textit{NatNum}(m) \Rightarrow 0 + m = m$$

Defining Addition

$$\forall m \; \textit{NatNum}(m) \Rightarrow \underbrace{+(0, m)}_{\textit{prefix notation}} = m$$

Use infix notation for readability.

$$\forall m \; \textit{NatNum}(m) \Rightarrow \underbrace{0 + m}_{\textit{infix notation}} = m$$

Defining Addition

$$\forall m\ NatNum(m) \Rightarrow \underbrace{+(0,m)}_{\textit{prefix notation}} = m$$

Use infix notation for readability.

$$\forall m\ NatNum(m) \Rightarrow \underbrace{0+m}_{\textit{infix notation}} = m$$

$$\forall m,n\ NatNum(m) \land NatNum(n) \Rightarrow +(S(m),n) = S(+(m,n))$$

Defining Addition

$$\forall m\ \mathit{NatNum}(m) \Rightarrow \underbrace{+(0, m)}_{\textit{prefix notation}} = m$$

Use infix notation
for readability.

$$\forall m\ \mathit{NatNum}(m) \Rightarrow \underbrace{0 + m}_{\textit{infix notation}} = m$$

$$\forall m, n\ \mathit{NatNum}(m) \wedge \mathit{NatNum}(n) \Rightarrow +(S(m), n) = S(+(m, n))$$

Write $S(n)$
as $n + 1$.

$$\forall m, n\ \mathit{NatNum}(m) \wedge \mathit{NatNum}(n) \Rightarrow (m + 1) + n = (m + n) + 1$$

Defining Addition

$\forall m\ NatNum(m) \Rightarrow \underbrace{+(0,m)}_{\textit{prefix notation}} = m$

Use infix notation for readability.

Syntactic sugar: an extension to the standard syntax that does not change the semantics but improves readability.

$\forall m\ NatNum(m) \Rightarrow \underbrace{0+m}_{\textit{infix notation}} = m$

$\forall m,n\ NatNum(m) \land NatNum(n) \Rightarrow +(S(m),n) = S(+(m,n))$

Write $S(n)$ as $n+1$.

$\forall m,n\ NatNum(m) \land NatNum(n) \Rightarrow (m+1)+n = (m+n)+1$

Defining Addition

$\forall m\ \mathit{NatNum}(m) \Rightarrow \underbrace{+(0,m)}_{\text{prefix notation}} = m$

Use infix notation for readability.

Syntactic sugar: an extension to the standard syntax that does not change the semantics but improves readability.

$\forall m\ \mathit{NatNum}(m) \Rightarrow \underbrace{0+m}_{\text{infix notation}} = m$

$\forall m,n\ \mathit{NatNum}(m) \wedge \mathit{NatNum}(n) \Rightarrow +(S(m),n) = S(+(m,n))$

Write $S(n)$ as $n+1$.

$\boxed{\forall m,n}\ \mathit{NatNum}(m) \wedge \mathit{NatNum}(n) \Rightarrow (m+1)+n = (m+n)+1$

Syntactic sugar: $\forall m \forall n$ according to syntax

Defining Addition

$\forall m\ NatNum(m) \Rightarrow \underbrace{+(0, m)}_{\textit{prefix notation}} = m$

Use infix notation for readability.

Syntactic sugar: an extension to the standard syntax that does not change the semantics but improves readability.

$\forall m\ NatNum(m) \Rightarrow \underbrace{0 + m}_{\textit{infix notation}} = m$

$\forall m, n\ NatNum(m) \wedge NatNum(n) \Rightarrow +(S(m), n) = S(+(m, n))$

Write $S(n)$ as $n + 1$.

$\boxed{\forall m, n}\ NatNum(m) \wedge NatNum(n) \Rightarrow (m + 1) + n = (m + n) + 1$

Syntactic sugar: $\forall m \forall n$ according to syntax

Domain of Sets

Syntatic sugar:

- {} for the empty set
- One unary predicate, *Set*

Domain of Sets

Syntatic sugar:

- $\{\}$ for the empty set
- One unary predicate, *Set*
- Binary predicate, $\in$

  e.g., $x \in s$ (x is a member of set s)

Domain of Sets

Syntatic sugar:

- $\{\}$ for the empty set
- One unary predicate, *Set*
- Binary predicate, ∈

  e.g., $x \in s$ (x is a member of set s)
- Binary predicate, ⊆

  e.g., $s_1 \subseteq s_2$ (set s_1 is a subset of set s_2)

Domain of Sets

Syntatic sugar:

- {} for the empty set
- One unary predicate, *Set*
- Binary predicate, ∈

  e.g., $x \in s$ (x is a member of set s)

- Binary predicate, ⊆

  e.g., $s_1 \subseteq s_2$ (set s_1 is a subset of set s_2)

- Binary functions, ∩ (intersection), ∪ (union), *Add*

  $s_1 \cap s_2$

Domain of Sets

Syntatic sugar:

- $\{\}$ for the empty set
- One unary predicate, *Set*
- Binary predicate, $\in$

  e.g., $x \in s$ (x is a member of set s)
- Binary predicate, $\subseteq$

  e.g., $s_1 \subseteq s_2$ (set s_1 is a subset of set s_2)
- Binary functions, $\cap$ (intersection), $\cup$ (union), *Add*

  $s_1 \cap s_2$ $s_1 \cup s_2$

Domain of Sets

Syntatic sugar:

- $\{\}$ for the empty set
- One unary predicate, *Set*
- Binary predicate, $\in$

  e.g., $x \in s$ (x is a member of set s)
- Binary predicate, $\subseteq$

  e.g., $s_1 \subseteq s_2$ (set s_1 is a subset of set s_2)
- Binary functions, $\cap$ (intersection), $\cup$ (union), *Add*

  $s_1 \cap s_2$ $\quad$ $s_1 \cup s_2$ $\quad$ $Add(x, s)$

  the set resulting from add element x to set s

Eight Axioms for Sets

- A set is either an empty set or made by adding something to a set.

$$\forall s\ Set(s) \Leftrightarrow (s = \{\}) \vee (\exists x, s_0\ \ Set(s_0) \wedge s = Add(x, s_0))$$

Eight Axioms for Sets

- A set is either an empty set or made by adding something to a set.

$$\forall s\ \mathit{Set}(s) \Leftrightarrow (s = \{\}) \vee (\exists x, s_0\ \ \mathit{Set}(s_0) \wedge s = \mathit{Add}(x, s_0))$$

- The empty set has no elements added to it.

$$\neg \exists x, s\ \ \mathit{Add}(x, s) = \{\}$$

// equivalently, no way to decompose {}
// into a smaller set and an element

Eight Axioms for Sets

- A set is either an empty set or made by adding something to a set.

$$\forall s \; Set(s) \Leftrightarrow (s = \{\}) \vee (\exists x, s_0 \;\; Set(s_0) \wedge s = Add(x, s_0))$$

- The empty set has no elements added to it.

$$\neg \exists x, s \;\; Add(x, s) = \{\}$$

// equivalently, no way to decompose {}
// into a smaller set and an element

- Adding an element already in the set has no effect.

$$\forall x, s \quad x \in s \Leftrightarrow s = Add(x, s)$$

Eight Axioms for Sets

- A set is either an empty set or made by adding something to a set.

$$\forall s\ Set(s) \Leftrightarrow (s = \{\}) \vee (\exists x, s_0\ \ Set(s_0) \wedge s = Add(x, s_0))$$

- The empty set has no elements added to it.

$$\neg\exists x, s\ \ Add(x, s) = \{\}$$

// equivalently, no way to decompose {}
// into a smaller set and an element

- Adding an element already in the set has no effect.

$$\forall x, s\ \ \ x \in s \Leftrightarrow s = Add(x, s)$$

- The only members of set are the added elements.

Eight Axioms for Sets

- A set is either an empty set or made by adding something to a set.

$$\forall s \; Set(s) \Leftrightarrow (s = \{\}) \vee (\exists x, s_0 \;\; Set(s_0) \wedge s = Add(x, s_0))$$

- The empty set has no elements added to it.

$$\neg \exists x, s \;\; Add(x, s) = \{\}$$

// equivalently, no way to decompose {}
// into a smaller set and an element

- Adding an element already in the set has no effect.

$$\forall x, s \quad x \in s \Leftrightarrow s = Add(x, s)$$

- The only members of set are the added elements.

$$\forall x, s \quad x \in s \Leftrightarrow \exists y, s_2 \quad (s = Add(y, s_2) \wedge (x = y \vee x \in s_2))$$

// expressed recursively

Eight Axioms for Sets

- A set is either an empty set or made by adding something to a set.

$$\forall s \; Set(s) \Leftrightarrow (s = \{\}) \vee (\exists x, s_0 \;\; Set(s_0) \wedge s = Add(x, s_0))$$

- The empty set has no elements added to it.

$$\neg \exists x, s \;\; Add(x, s) = \{\}$$

// equivalently, no way to decompose {}
// into a smaller set and an element

- Adding an element already in the set has no effect.

$$\forall x, s \quad x \in s \Leftrightarrow s = Add(x, s)$$

- The only members of set are the added elements.

$$\forall x, s \quad x \in s \Leftrightarrow \exists y, s_2 \quad (s = Add(y, s_2) \wedge (x = y \vee x \in s_2))$$

// expressed recursively

must be true at some recursion level

Set Axioms (cont'd)

- Subset relationship

$$\forall s_1, s_2 \quad s_1 \subseteq s_2 \Leftrightarrow (\forall x \;\; x \in s_1 \Rightarrow x \in s_2)$$

Set Axioms (cont'd)

- Subset relationship

$$\forall s_1, s_2 \quad s_1 \subseteq s_2 \Leftrightarrow (\forall x \;\; x \in s_1 \Rightarrow x \in s_2)$$

- Set equality

$$\forall s_1, s_2 \quad s_1 = s_2 \Leftrightarrow (s_1 \subseteq s_2 \wedge s_2 \subseteq s_1)$$

Set Axioms (cont'd)

- Subset relationship

$$\forall s_1, s_2 \quad s_1 \subseteq s_2 \Leftrightarrow (\forall x \;\; x \in s_1 \Rightarrow x \in s_2)$$

- Set equality

$$\forall s_1, s_2 \quad s_1 = s_2 \Leftrightarrow (s_1 \subseteq s_2 \wedge s_2 \subseteq s_1)$$

- Intersection

$$\forall x, s_1, s_2 \quad x \in s_1 \cap s_2 \Leftrightarrow (x \in s_1 \wedge x \in s_2)$$

Set Axioms (cont'd)

- Subset relationship

$$\forall s_1, s_2 \quad s_1 \subseteq s_2 \Leftrightarrow (\forall x \;\; x \in s_1 \Rightarrow x \in s_2)$$

- Set equality

$$\forall s_1, s_2 \quad s_1 = s_2 \Leftrightarrow (s_1 \subseteq s_2 \land s_2 \subseteq s_1)$$

- Intersection

$$\forall x, s_1, s_2 \quad x \in s_1 \cap s_2 \Leftrightarrow (x \in s_1 \land x \in s_2)$$

- Union

$$\forall x, s_1, s_2 \quad x \in s_1 \cup s_2 \Leftrightarrow (x \in s_1 \lor x \in s_2)$$

III. The Wumpus World in FOL

First-order logic axioms are much more concise than propositional axioms.

III. The Wumpus World in FOL

First-order logic axioms are much more concise than propositional axioms.

- Predicate for percept

$$Percept([Stench, Breeze, Glitter, None, None], 5)$$

III. The Wumpus World in FOL

First-order logic axioms are much more concise than propositional axioms.

- Predicate for percept

$$\underbrace{\mathit{Percept}([\mathit{Stench}, \mathit{Breeze}, \mathit{Glitter}, \mathit{None}, \mathit{None}]}_{\text{sensor reading}}, 5)$$

III. The Wumpus World in FOL

First-order logic axioms are much more concise than propositional axioms.

- Predicate for percept

$$\mathit{Percept}(\underbrace{[\mathit{Stench}, \mathit{Breeze}, \mathit{Glitter}, \mathit{None}, \mathit{None}]}_{\text{sensor reading}}, \underset{\text{time}}{5})$$

III. The Wumpus World in FOL

First-order logic axioms are much more concise than propositional axioms.

- Predicate for percept

$$\underbrace{\mathit{Percept}([\underbrace{\mathit{Stench}, \mathit{Breeze}, \mathit{Glitter}, \mathit{None}, \mathit{None}}_{\text{sensor reading}}], \underbrace{5}_{\text{time}})}$$

- Actions

Turn(*Right*), *Turn*(Left), *Forward*, *Shoot*, *Grab*, *Climb*

III. The Wumpus World in FOL

First-order logic axioms are much more concise than propositional axioms.

- Predicate for percept

$$\textit{Percept}(\underbrace{[\textit{Stench}, \textit{Breeze}, \textit{Glitter}, \textit{None}, \textit{None}]}_{\text{sensor reading}}, \underbrace{5}_{\text{time}})$$

- Actions

Turn(*Right*), *Turn*(Left), *Forward*, *Shoot*, *Grab*, *Climb*

- Querying the KB for best action

ASKVARS(*KB*, *BestAction*(a, 5))

A binding list, e.g., $\{a/\textit{Grab}\}$, is returned.

Input/Output Rules

No need for the use of fluents (e.g., *Forward*^{t}, *Bump*$^{t+1}$)!

$\forall t, s, g, w, c \;\; Percept([s, Breeze, g, w, c], t) \Rightarrow Breeze(t)$

$\forall t, s, g, w, c \;\; Percept([s, None, g, w, c], t) \Rightarrow \neg Breeze(t)$

$\forall t, s, b, w, c \;\; Percept([s, b, Glitter, w, c], t) \Rightarrow Glitter(t)$

$\forall t, s, b, w, c \;\; Percept([s, b, None, w, c], t) \Rightarrow \neg Glitter(t)$

$\vdots$

Input/Output Rules

No need for the use of fluents (e.g., $\mathit{Forward}^{t}$, Bump^{t+1})!

$$\forall t, s, g, w, c \;\; \mathit{Percept}([s, \mathit{Breeze}, g, w, c], t) \Rightarrow \mathit{Breeze}(t)$$

$$\forall t, s, g, w, c \;\; \mathit{Percept}([s, \mathit{None}, g, w, c], t) \Rightarrow \neg \mathit{Breeze}(t)$$

$$\forall t, s, b, w, c \;\; \mathit{Percept}([s, b, \mathit{Glitter}, w, c], t) \Rightarrow \mathit{Glitter}(t)$$

$$\forall t, s, b, w, c \;\; \mathit{Percept}([s, b, \mathit{None}, w, c], t) \Rightarrow \neg \mathit{Glitter}(t)$$

$$\vdots$$

$$\forall t \;\; \mathit{Glitter}(t) \Rightarrow \mathit{BestAction}(\mathit{Grab}, t)$$

// simple “reflex” behavior

Rules for the Environment

- Adjacency of two squares

$$\forall x, y, a, b \;\; \textit{Adjacent}([x, y], [a, b]) \Leftrightarrow$$
$$(x = a \wedge (y = b - 1 \vee y = b + 1)) \vee (y = b \wedge (x = a - 1 \vee x = a + 1))$$

// if using proportional logic, we would have to name every square, say, Square$_{i,j}$,
// for $1 \leq i, j \leq 4$, and would need one such fact for 120 different pairs of squares!

Rules for the Environment

- Adjacency of two squares

square at row x and column y

$$\forall x, y, a, b \;\; \textit{Adjacent}([x, y], [a, b]) \Leftrightarrow$$
$$(x = a \land (y = b - 1 \lor y = b + 1)) \lor (y = b \land (x = a - 1 \lor x = a + 1))$$

// if using proportional logic, we would have to name every square, say, Square$_{i,j}$,
// for $1 \le i, j \le 4$, and would need one such fact for 120 different pairs of squares!

Rules for the Environment

- Adjacency of two squares

square at row x and column y

$$\forall x, y, a, b \;\; \textit{Adjacent}([x, y], [a, b]) \Leftrightarrow$$
$$(x = a \wedge (y = b - 1 \vee y = b + 1)) \vee (y = b \wedge (x = a - 1 \vee x = a + 1))$$

// if using proportional logic, we would have to name every square, say, Square$_{i,j}$,
// for $1 \leq i, j \leq 4$, and would need one such fact for 120 different pairs of squares!

- Unary predicate *Pit* (no reason to distinguish among pits).

Rules for the Environment

- Adjacency of two squares

square at row x and column y

$$\forall x, y, a, b \;\; \textit{Adjacent}([x, y], [a, b]) \Leftrightarrow$$
$$(x = a \wedge (y = b - 1 \vee y = b + 1)) \vee (y = b \wedge (x = a - 1 \vee x = a + 1))$$

// if using proportional logic, we would have to name every square, say, Square$_{i,j}$,
// for $1 \leq i, j \leq 4$, and would need one such fact for 120 different pairs of squares!

- Unary predicate *Pit* (no reason to distinguish among pits).

$\textit{Pit}([x, y]) \equiv \textit{true}$ if and only if the square $[x, y]$ contains a pit.

Rules for the Environment

- Adjacency of two squares

square at row x and column y

$$\forall x, y, a, b \;\; \textit{Adjacent}([x, y], [a, b]) \Leftrightarrow$$
$$(x = a \wedge (y = b - 1 \vee y = b + 1)) \vee (y = b \wedge (x = a - 1 \vee x = a + 1))$$

// if using proportional logic, we would have to name every square, say, Square$_{i,j}$,
// for $1 \leq i, j \leq 4$, and would need one such fact for 120 different pairs of squares!

- Unary predicate *Pit* (no reason to distinguish among pits).

$\textit{Pit}([x, y]) \equiv$ *true* if and only if the square $[x, y]$ contains a pit.

- Constants *Agent*, *Wumpus*

Rules for the Environment

- Adjacency of two squares

square at row x and column y

$$\forall x, y, a, b \;\; \mathit{Adjacent}([x,y],[a,b]) \Leftrightarrow$$
$$(x = a \wedge (y = b-1 \vee y = b+1)) \vee (y = b \wedge (x = a-1 \vee x = a+1))$$

// if using proportional logic, we would have to name every square, say, Square$_{i,j}$,
// for $1 \leq i,j \leq 4$, and would need one such fact for 120 different pairs of squares!

- Unary predicate *Pit* (no reason to distinguish among pits).

$\mathit{Pit}([x,y]) \equiv \mathit{true}$ if and only if the square $[x,y]$ contains a pit.

- Constants *Agent*, *Wumpus*

- Ternary predicate *At* to represent changing or non-changing locations

$\forall t \; \mathit{At}(\mathit{Wumpus}, [1,3], t)$ // fixed location for the wumpus

Rules for the Environment

- Adjacency of two squares

square at row x and column y

$$\forall x, y, a, b \;\; \textit{Adjacent}([x, y], [a, b]) \Leftrightarrow$$
$$(x = a \wedge (y = b - 1 \vee y = b + 1)) \vee (y = b \wedge (x = a - 1 \vee x = a + 1))$$

// if using proportional logic, we would have to name every square, say, Square$_{i,j}$,
// for $1 \leq i, j \leq 4$, and would need one such fact for 120 different pairs of squares!

- Unary predicate *Pit* (no reason to distinguish among pits).

$\textit{Pit}([x, y]) \equiv \textit{true}$ if and only if the square $[x, y]$ contains a pit.

- Constants *Agent*, *Wumpus*

- Ternary predicate *At* to represent changing or non-changing locations

$\forall t \; \textit{At}(\textit{Wumpus}, [1, 3], t)$ // fixed location for the wumpus

$\forall x, s_1, s_2, t \quad \textit{At}(x, s_1, t) \wedge \textit{At}(x, s_2, t) \Rightarrow s_1 = s_2$ // only one location at a time

Rules for Percepts, Actions and Inferences

- Percepts of breeze, stench, etc.

$$\forall s,t\ At(Agent,s,t) \land Breeze(t) \Rightarrow Breezy(s)$$

Rules for Percepts, Actions and Inferences

- Percepts of breeze, stench, etc.

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

- *Diagnostic* (inferring cause from effect)

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

// if using proportional logic, we would need a separate axiom for every square.

Rules for Percepts, Actions and Inferences

- Percepts of breeze, stench, etc.

$$\forall s,t\ \mathit{At}(\mathit{Agent},s,t) \land \mathit{Breeze}(t) \Rightarrow \mathit{Breezy}(s)$$

- *Diagnostic* (inferring cause from effect)

$$\forall s\ \mathit{Breezy}(s) \ \Leftrightarrow \exists r\ \ \mathit{Adjacent}(r,s) \land \mathit{Pit}(r)$$

// if using proportional logic, we would need a separate axiom for every square.

- *Causal* (inferring effect from cause)

$$\forall s\ (\forall r\ \ \mathit{Adjacent}(r,s) \ \Rightarrow \neg \mathit{Pit}(r)) \Rightarrow \neg \mathit{Breezy}(s)$$

Rules for Percepts, Actions and Inferences

- Percepts of breeze, stench, etc.

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

- *Diagnostic* (inferring cause from effect)

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

// if using proportional logic, we would need a separate axiom for every square.

- *Causal* (inferring effect from cause)

$$\forall s \ (\forall r \ Adjacent(r, s) \Rightarrow \neg Pit(r)) \Rightarrow \neg Breezy(s)$$

- Quantification over time

$$\forall t \ HaveArrow(t+1) \Leftrightarrow HaveArrow(t) \wedge \neg Action(Shoot, t)$$

Rules for Percepts, Actions and Inferences

- Percepts of breeze, stench, etc.

$$\forall s, t \; At(Agent, s, t) \wedge Breeze(t) \Rightarrow Breezy(s)$$

- *Diagnostic* (inferring cause from effect)

$$\forall s \; Breezy(s) \; \Leftrightarrow \exists r \;\; Adjacent(r, s) \wedge Pit(r)$$

// if using proportional logic, we would need a separate axiom for every square.

- *Causal* (inferring effect from cause)

$$\forall s \; (\forall r \;\; Adjacent(r, s) \; \Rightarrow \neg Pit(r)) \Rightarrow \neg Breezy(s)$$

- Quantification over time

$$\forall t \; HaveArrow(t+1) \; \Leftrightarrow HaveArrow(t) \wedge \neg Action(Shoot, t)$$

The first-order logic formulation is no less concise than the English description.

Rules for Percepts, Actions and Inferences

- Percepts of breeze, stench, etc.

$$\forall s,t\ \textit{At}(\textit{Agent},s,t) \land \textit{Breeze}(t) \Rightarrow \textit{Breezy}(s)$$

- *Diagnostic* (inferring cause from effect)

$$\forall s\ \textit{Breezy}(s) \Leftrightarrow \exists r\ \ \textit{Adjacent}(r,s) \land \textit{Pit}(r)$$

// if using proportional logic, we would need a separate axiom for every square.

- *Causal* (inferring effect from cause)

$$\forall s\ (\forall r\ \ \textit{Adjacent}(r,s) \Rightarrow \neg \textit{Pit}(r)) \Rightarrow \neg \textit{Breezy}(s)$$

- Quantification over time

$$\forall t\ \textit{HaveArrow}(t+1) \Leftrightarrow \textit{HaveArrow}(t) \land \neg \textit{Action}(\textit{Shoot},t)$$

The first-order logic formulation is no less concise than the English description.