Using First-Order Logic (FOL)

Outline

I. The wumpus world in FOL

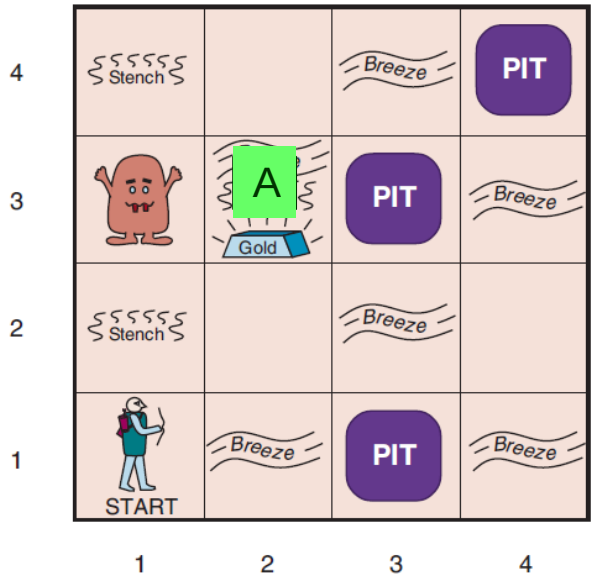
II. Knowledge engineering process

III. Instantiation and Skolemization

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

I. The Wumpus World in FOL

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

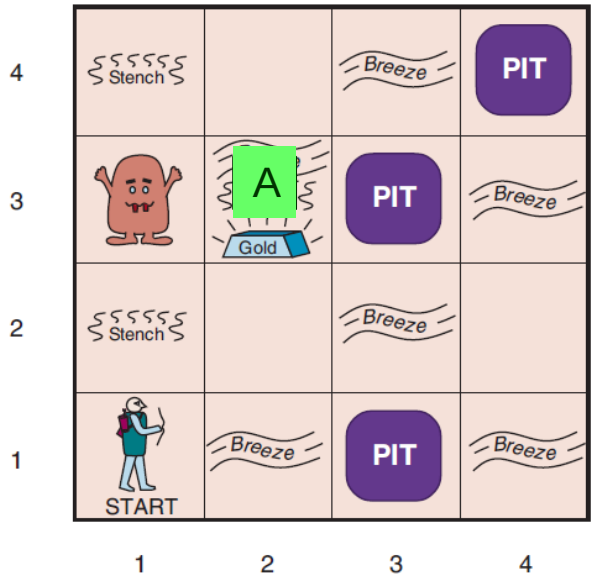


I. The Wumpus World in FOL

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

- Predicate for percept

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



I. The Wumpus World in FOL

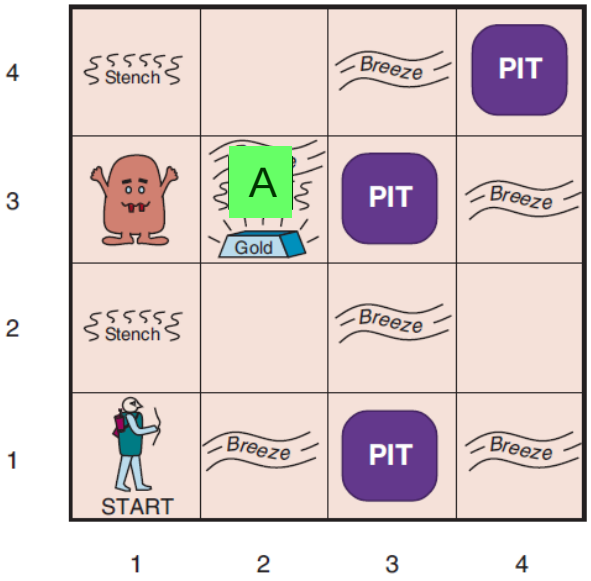
First-order logic axioms are much more concise than propositional axioms.

- Predicate for percept

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

No Bump *No Scream*

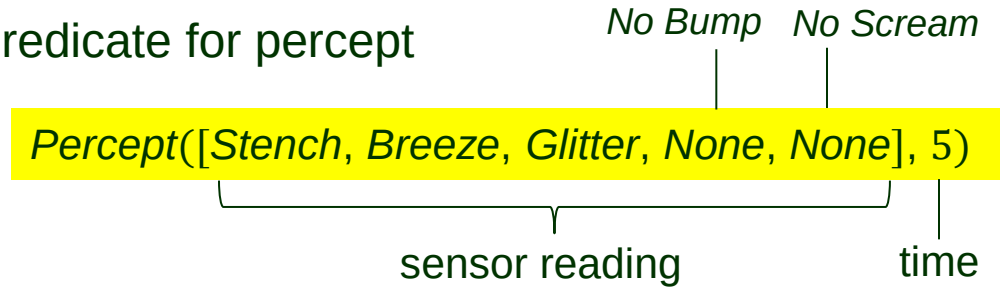
sensor reading

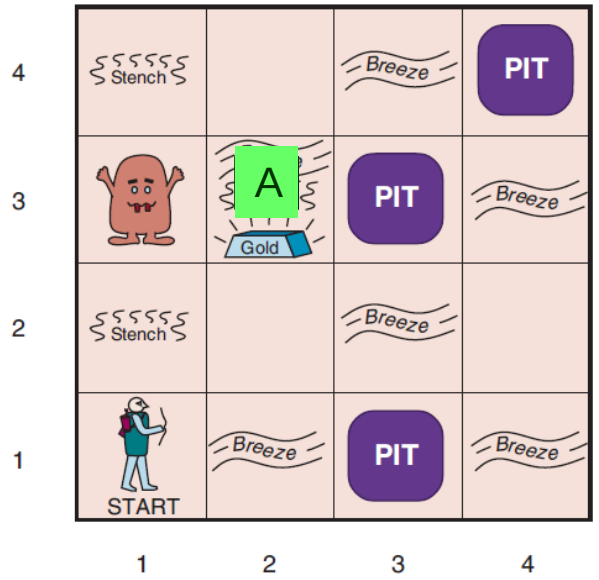


I. The Wumpus World in FOL

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

- Predicate for percept





I. 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)

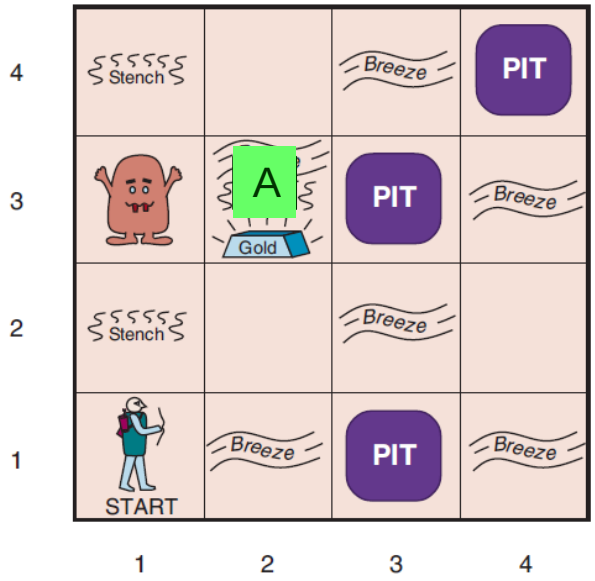
No Bump *No Scream*

sensor reading

time

- Actions

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



I. 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)

No Bump *No Scream*

sensor reading time

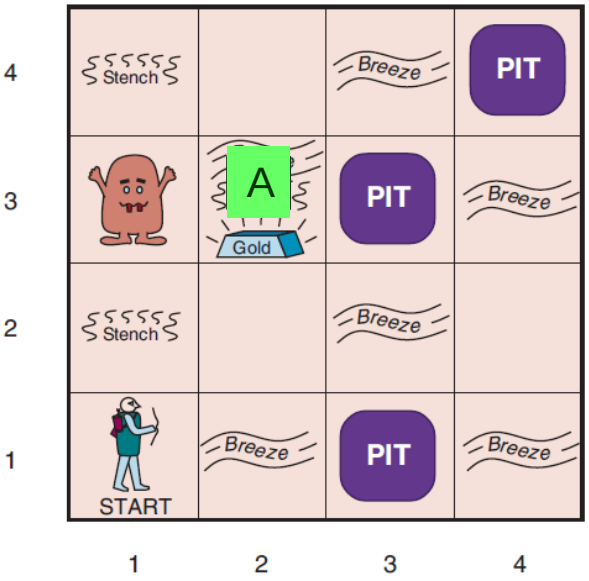
- Actions

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

- Querying the KB for best action

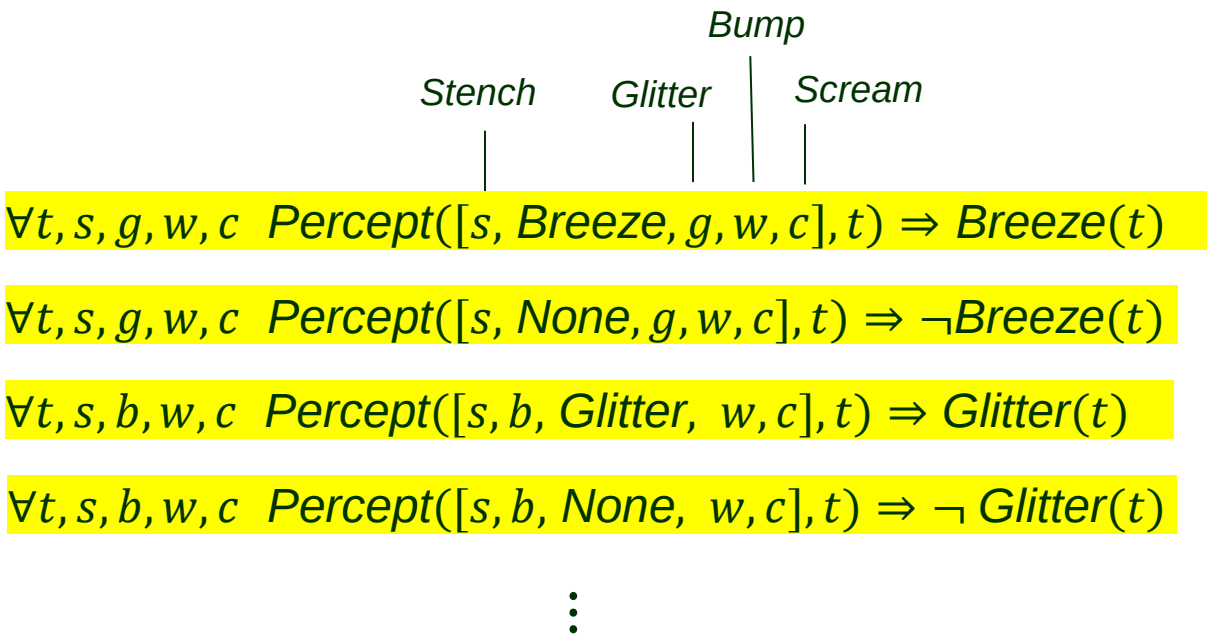
ASKVARS(*KB*, *BestAction*(a, 5))

A binding list, e.g., {a/*Grab*}, is returned.



Input/Output Rules

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



Input/Output Rules

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

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

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

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

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

⋮

$\forall t \;\; Glitter(t) \Rightarrow BestAction(Grab, t)$ // simple “reflex” behavior

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, $\text{Square}_{i,j}$,
// 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

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

// if using proportional logic, we would have to name every square, say, $\text{Square}_{i,j}$,
// 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, $\text{Square}_{i,j}$,
// 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).

$\textit{Pit}([x, y]) \equiv \textit{true}$ 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

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

// if using proportional logic, we would have to name every square, say, $\text{Square}_{i,j}$,
// 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

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, $\text{Square}_{i,j}$,
// 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.

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

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.

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

- *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)

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

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.

II. Knowledge Engineering Process

- Identification of questions and facts.

  What will the KB support and what facts are available?

II. Knowledge Engineering Process

- Identification of questions and facts.

  What will the KB support and what facts are available?

- Knowledge assembly or acquisition.

  Work with real experts to extract knowledge (not yet formally represented).

II. Knowledge Engineering Process

- Identification of questions and facts.

  What will the KB support and what facts are available?

- Knowledge assembly or acquisition.

  Work with real experts to extract knowledge (not yet formally represented).

- Decision on a vocabulary (predicates, functions, and constants).

  In the wumpus world, should pits be represented by objects or by a unary predicate on squares?

II. Knowledge Engineering Process

- Identification of questions and facts.

  What will the KB support and what facts are available?

- Knowledge assembly or acquisition.

  Work with real experts to extract knowledge (not yet formally represented).

- Decision on a vocabulary (predicates, functions, and constants).

  In the wumpus world, should pits be represented by objects or by a unary predicate on squares?

- Encoding of general knowledge (axioms).

- Formal description of the problem instance.

II. Knowledge Engineering Process

- Identification of questions and facts.

  What will the KB support and what facts are available?

- Knowledge assembly or acquisition.

  Work with real experts to extract knowledge (not yet formally represented).

- Decision on a vocabulary (predicates, functions, and constants).

  In the wumpus world, should pits be represented by objects or by a unary predicate on squares?

- Encoding of general knowledge (axioms).

- Formal description of the problem instance.

- Queries to and answers from the inference procedure.

  Derive the facts we are interested in knowing.

II. Knowledge Engineering Process

- Identification of questions and facts.

  What will the KB support and what facts are available?

- Knowledge assembly or acquisition.

  Work with real experts to extract knowledge (not yet formally represented).

- Decision on a vocabulary (predicates, functions, and constants).

  In the wumpus world, should pits be represented by objects or by a unary predicate on squares?

- Encoding of general knowledge (axioms).

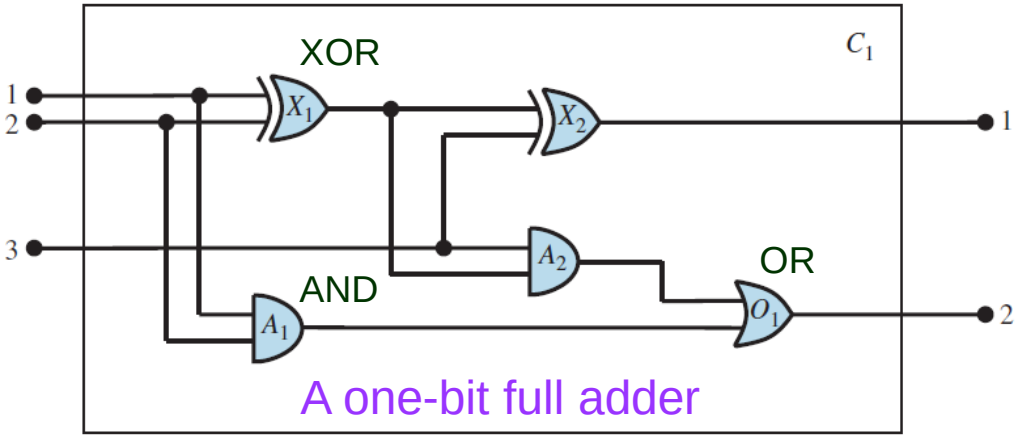
- Formal description of the problem instance.

- Queries to and answers from the inference procedure.

  Derive the facts we are interested in knowing.

- Debugging and evaluation of the KB.

The Electronic Circuits Domain

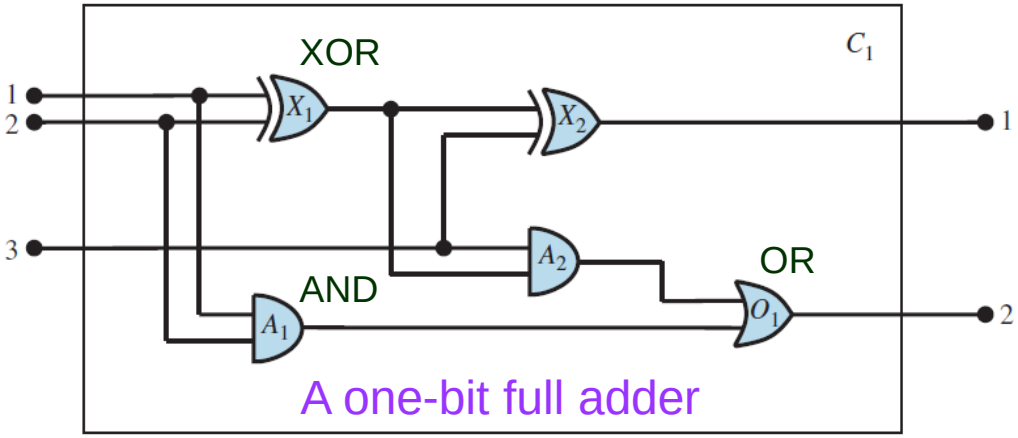


- Four types of gates: *AND*, *OR*, *XOR*, *NOT*.

  1 or 2 inputs, and 1 output

- Represent connections between terminals.

The Electronic Circuits Domain

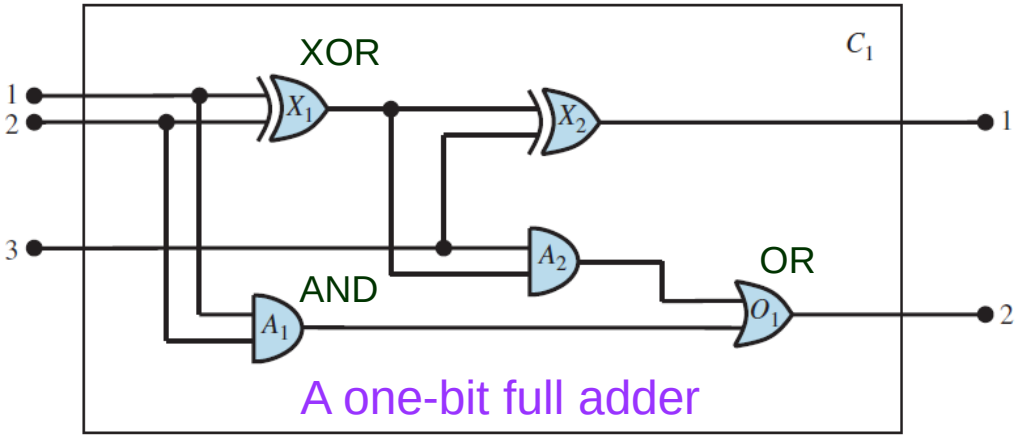


- Four types of gates: *AND*, *OR*, *XOR*, *NOT*.

  1 or 2 inputs, and 1 output

- Represent connections between terminals.
- No need to represent wires or their paths.
- Component sizes, shapes, colors, or costs are irrelevant.

The Electronic Circuits Domain



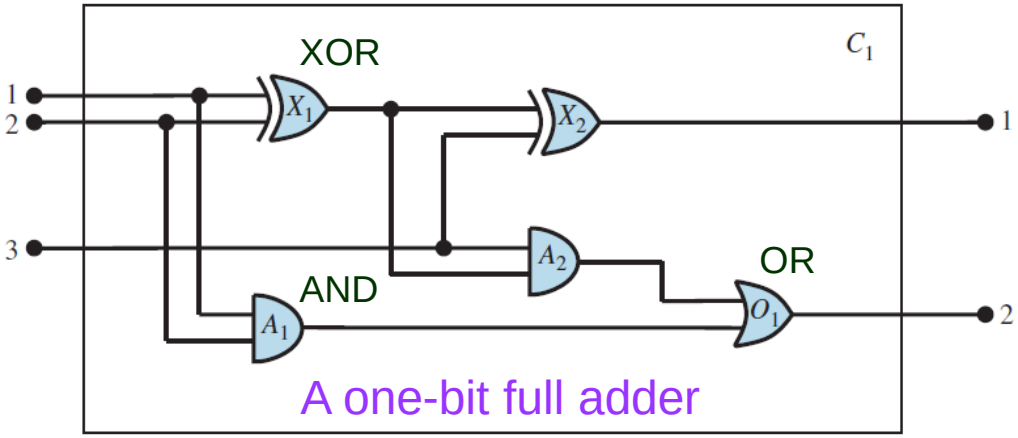
- Four types of gates: *AND*, *OR*, *XOR*, *NOT*.

  1 or 2 inputs, and 1 output

- Represent connections between terminals.
- No need to represent wires or their paths.
- Component sizes, shapes, colors, or costs are irrelevant.

♦ Objects: actual gates and circuits. E.g., X_1, A_2, C_1.

The Electronic Circuits Domain



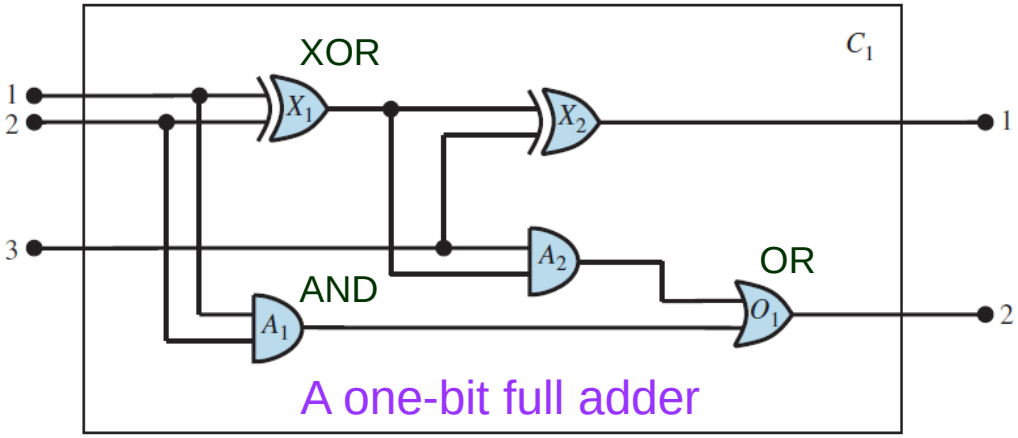
- Four types of gates: *AND*, *OR*, *XOR*, *NOT*.

  1 or 2 inputs, and 1 output

- Represent connections between terminals.
- No need to represent wires or their paths.
- Component sizes, shapes, colors, or costs are irrelevant.

- Objects: actual gates and circuits. E.g., X_1, A_2, C_1.
- Predicates: *Gate*, *Terminal*, *Circuit*

The Electronic Circuits Domain



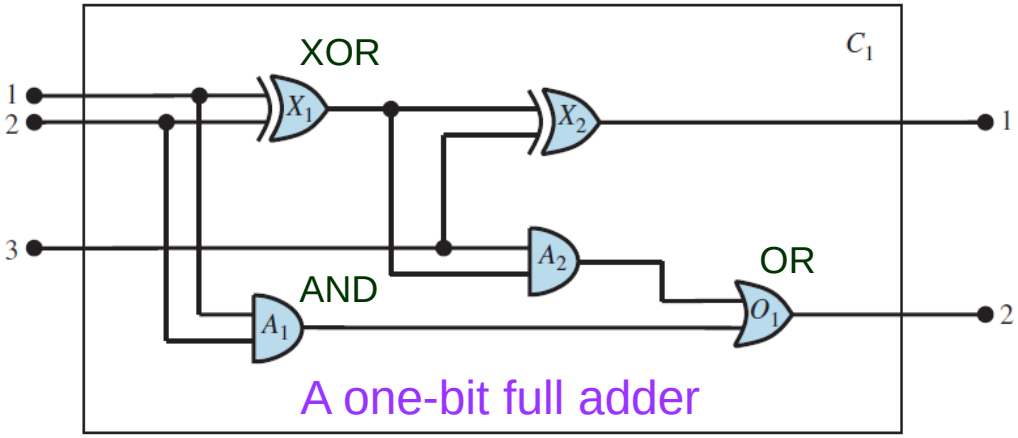
- Four types of gates: *AND*, *OR*, *XOR*, *NOT*.

  1 or 2 inputs, and 1 output

- Represent connections between terminals.
- No need to represent wires or their paths.
- Component sizes, shapes, colors, or costs are irrelevant.

- Objects: actual gates and circuits. E.g., X_1, A_2, C_1.
- Predicates: *Gate*, *Terminal*, *Circuit*
- Functions
  - *Type*(X_1): type of gate X_1 (which is *XOR*)
  - *In*$(1, X_2)$: 1st input terminal for gate X_2
  - *Out*$(2, C_1)$: 2nd output terminal for circuit C_1
  - *Arity*$(A_1, 2,1)$: two input terminals and one output terminal for the gate A_1

The Electronic Circuits Domain



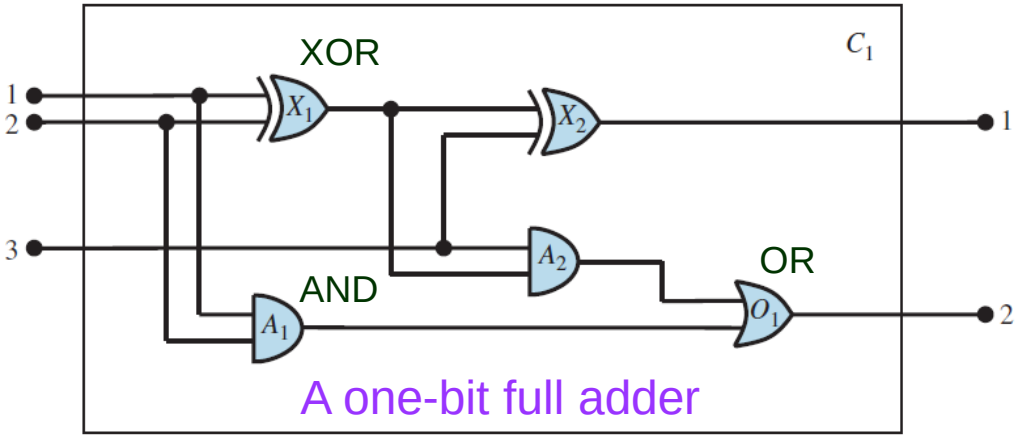
- Four types of gates: *AND*, *OR*, *XOR*, *NOT*.

  1 or 2 inputs, and 1 output

- Represent connections between terminals.
- No need to represent wires or their paths.
- Component sizes, shapes, colors, or costs are irrelevant.

- Objects: actual gates and circuits. E.g., X_1, A_2, C_1.
- Predicates: *Gate*, *Terminal*, *Circuit*
- Functions
  - *Type*(X_1): type of gate X_1 (which is *XOR*)
  - *In*$(1, X_2)$: 1st input terminal for gate X_2
  - *Out*$(2, C_1)$: 2nd output terminal for circuit C_1
  - *Arity*$(A_1, 2,1)$: two input terminals and one output terminal for the gate A_1
- Connectivity predicate: *Connected*$\big($*Out*$(1, X_1)$, *In*$(1, X_2)\big)$

The Electronic Circuits Domain



- Four types of gates: *AND*, *OR*, *XOR*, *NOT*.

  1 or 2 inputs, and 1 output

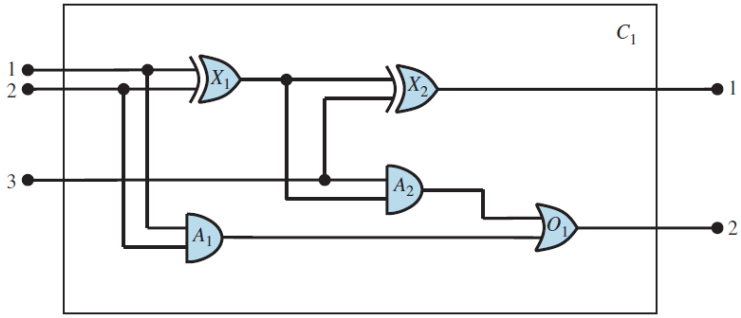
- Represent connections between terminals.
- No need to represent wires or their paths.
- Component sizes, shapes, colors, or costs are irrelevant.

- Objects: actual gates and circuits. E.g., X_1, A_2, C_1.
- Predicates: *Gate*, *Terminal*, *Circuit*
- Functions
  - *Type*(X_1): type of gate X_1 (which is *XOR*)
  - *In*$(1, X_2)$: 1st input terminal for gate X_2
  - *Out*$(2, C_1)$: 2nd output terminal for circuit C_1
  - *Arity*$(A_1, 2,1)$: two input terminals and one output terminal for the gate A_1
- Connectivity predicate: *Connected*$\big($*Out*$(1, X_1)$, *In*$(1, X_2)\big)$
- Signal function: *Signal*(t) has value 1 or 0 at time t.

Encoding General Domain Knowledge

// Two connected terminals have the same signal.

$\forall t_1, t_2 \quad Terminal(t_1) \wedge Terminal(t_2) \wedge Connected(t_1, t_2) \Rightarrow Signal(t_1) = Signal(t_2)$



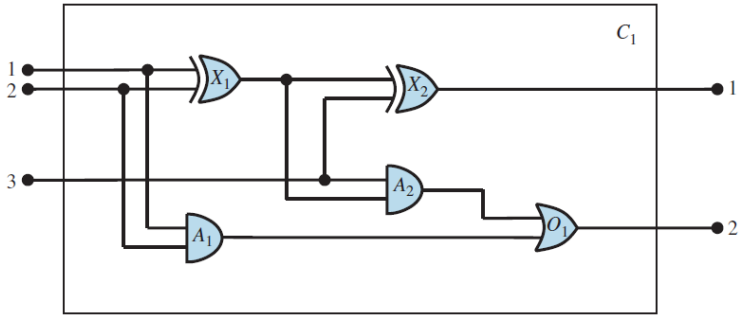
Encoding General Domain Knowledge

// Two connected terminals have the same signal.

$\forall t_1, t_2 \quad Terminal(t_1) \wedge Terminal(t_2) \wedge Connected(t_1, t_2) \Rightarrow Signal(t_1) = Signal(t_2)$

// Every terminal has signal that is either 1 or 0.

$\forall t \quad Terminal(t) \Rightarrow Signal(t) = 1 \vee Signal(t) = 0$



Encoding General Domain Knowledge

// Two connected terminals have the same signal.

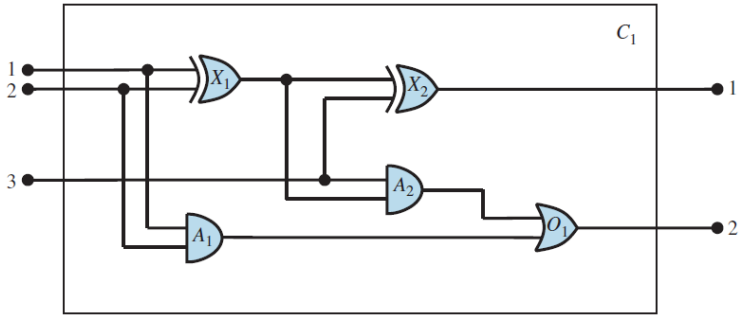
$\forall t_1, t_2 \quad Terminal(t_1) \wedge Terminal(t_2) \wedge Connected(t_1, t_2) \Rightarrow Signal(t_1) = Signal(t_2)$

// Every terminal has signal that is either 1 or 0.

$\forall t \quad Terminal(t) \Rightarrow Signal(t) = 1 \vee Signal(t) = 0$

// Connectivity is commutative.

$\forall t_1, t_2 \quad Connected(t_1, t_2) \Leftrightarrow Connected(t_2, t_1)$



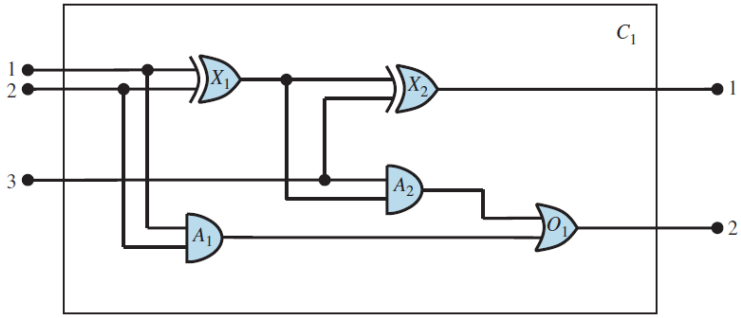
Encoding General Domain Knowledge

// Two connected terminals have the same signal.

$\forall t_1, t_2 \quad Terminal(t_1) \wedge Terminal(t_2) \wedge Connected(t_1, t_2) \Rightarrow Signal(t_1) = Signal(t_2)$

// Every terminal has signal that is either 1 or 0.

$\forall t \quad Terminal(t) \Rightarrow Signal(t) = 1 \vee Signal(t) = 0$



// Connectivity is commutative.

$\forall t_1, t_2 \quad Connected(t_1, t_2) \Leftrightarrow Connected(t_2, t_1)$

// Four types of gates.

$\forall g, k \quad Gate(g) \wedge k = Type(g) \Rightarrow k = AND \vee k = OR \vee k = XOR \vee k = NOT$

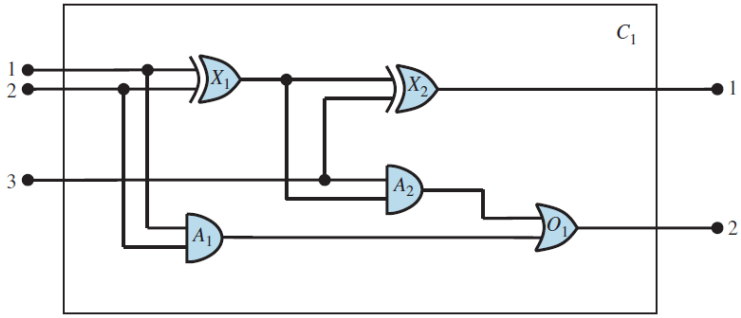
Encoding General Domain Knowledge

// Two connected terminals have the same signal.

$\forall t_1, t_2 \quad Terminal(t_1) \wedge Terminal(t_2) \wedge Connected(t_1, t_2) \Rightarrow Signal(t_1) = Signal(t_2)$

// Every terminal has signal that is either 1 or 0.

$\forall t \quad Terminal(t) \Rightarrow Signal(t) = 1 \vee Signal(t) = 0$



// Connectivity is commutative.

$\forall t_1, t_2 \quad Connected(t_1, t_2) \Leftrightarrow Connected(t_2, t_1)$

// Four types of gates.

$\forall g, k \quad Gate(g) \wedge k = Type(g) \Rightarrow k = AND \vee k = OR \vee k = XOR \vee k = NOT$

// An AND gate outputs 0 if and only if any of its input is 0.

$\forall g \quad Gate(g) \wedge Type(g) = AND \Rightarrow (Signal(Out(1, g)) = 0 \Leftrightarrow \exists n \; Signal(In(n, g)) = 0)$

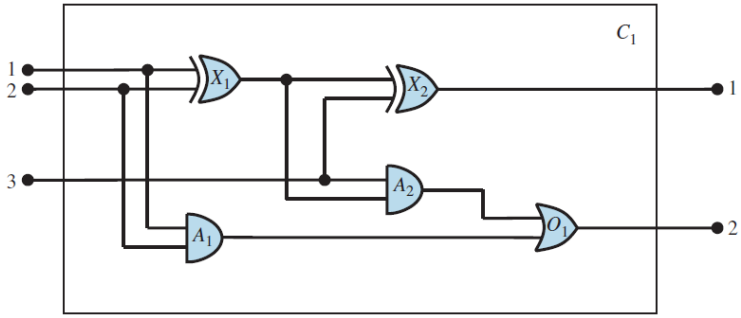
Encoding General Domain Knowledge

// Two connected terminals have the same signal.

$\forall t_1, t_2 \quad Terminal(t_1) \wedge Terminal(t_2) \wedge Connected(t_1, t_2) \Rightarrow Signal(t_1) = Signal(t_2)$

// Every terminal has signal that is either 1 or 0.

$\forall t \quad Terminal(t) \Rightarrow Signal(t) = 1 \vee Signal(t) = 0$



// Connectivity is commutative.

$\forall t_1, t_2 \quad Connected(t_1, t_2) \Leftrightarrow Connected(t_2, t_1)$

// Four types of gates.

$\forall g, k \quad Gate(g) \wedge k = Type(g) \Rightarrow k = AND \vee k = OR \vee k = XOR \vee k = NOT$

// An AND gate outputs 0 if and only if any of its input is 0.

$\forall g \quad Gate(g) \wedge Type(g) = AND \Rightarrow (Signal(Out(1, g)) = 0 \Leftrightarrow \exists n \; Signal(In(n, g)) = 0)$

// An OR gate outputs 1 if and only if any of its input is 1.

$\forall g \quad Gate(g) \wedge Type(g) = OR \Rightarrow (Signal(Out(1, g)) = 1 \Leftrightarrow \exists n \; Signal(In(n, g)) = 1)$

cont'd

// An XOR gate outputs 1 if and only if its inputs are different.

$$\forall g \;\; Gate(g) \wedge Type(g) = XOR \Rightarrow (Signal(Out(1,g)) = 1 \Leftrightarrow Signal(In(1,g)) \neq Signal(In(2,g)))$$

// An NOT gate's output is different from its input.

$$\forall g \;\; Gate(g) \wedge Type(g) = NOT \Rightarrow (Signal(Out(1,g)) = 1 \Leftrightarrow Signal(Out(1,g)) \neq Signal(In(1,g)))$$

cont'd

// An XOR gate outputs 1 if and only if its inputs are different.

$\forall g \;\; Gate(g) \wedge Type(g) = XOR \Rightarrow (Signal(Out(1,g)) = 1 \Leftrightarrow Signal(In(1,g)) \neq Signal(In(2,g)))$

// An NOT gate's output is different from its input.

$\forall g \;\; Gate(g) \wedge Type(g) = NOT \Rightarrow (Signal(Out(1,g)) = 1 \Leftrightarrow Signal(Out(1,g)) \neq Signal(In(1,g)))$

// All the gates (except for NOT) have two inputs and one output.

$\forall g \;\; Gate(g) \wedge Type(g) = NOT \Rightarrow Arity(g,1,1)$

$\forall g,k \;\; Gate(g) \wedge k{=}Type(g) \wedge (k = AND \vee k = OR \vee k = XOR) \Rightarrow Arity(g,2,1)$

cont'd

// An XOR gate outputs 1 if and only if its inputs are different.

$\forall g \;\; Gate(g) \wedge Type(g) = XOR \Rightarrow (Signal(Out(1,g)) = 1 \Leftrightarrow Signal(In(1,g)) \neq Signal(In(2,g)))$

// An NOT gate's output is different from its input.

$\forall g \;\; Gate(g) \wedge Type(g) = NOT \Rightarrow (Signal(Out(1,g)) = 1 \Leftrightarrow Signal(Out(1,g)) \neq Signal(In(1,g)))$

// All the gates (except for NOT) have two inputs and one output.

$\forall g \;\; Gate(g) \wedge Type(g) = NOT \Rightarrow Arity(g,1,1)$

$\forall g,k \;\; Gate(g) \wedge k{=}Type(g) \wedge (k = AND \vee k = OR \vee k = XOR) \Rightarrow Arity(g,2,1)$

// A circuit has terminals exactly up to its input and output arity.

$$\forall c,i,j \;\; Circuit(c) \wedge Arity(c,i,j) \Rightarrow$$
$$\forall n \;\; (n \leq i \Rightarrow Terminal(In(n,c)) \wedge (n > i \Rightarrow In(n,c) = Nothing)) \wedge$$
$$\forall n \;\; (n \leq j \Rightarrow Terminal(Out(n,c)) \wedge (n > j \Rightarrow Out(n,c) = Nothing))$$

cont'd

// An XOR gate outputs 1 if and only if its inputs are different.

$\forall g \;\; Gate(g) \wedge Type(g) = XOR \Rightarrow (Signal(Out(1,g)) = 1 \Leftrightarrow Signal(In(1,g)) \neq Signal(In(2,g)))$

// An NOT gate's output is different from its input.

$\forall g \;\; Gate(g) \wedge Type(g) = NOT \Rightarrow (Signal(Out(1,g)) = 1 \Leftrightarrow Signal(Out(1,g)) \neq Signal(In(1,g)))$

// All the gates (except for NOT) have two inputs and one output.

$\forall g \;\; Gate(g) \wedge Type(g) = NOT \Rightarrow Arity(g,1,1)$

$\forall g,k \;\; Gate(g) \wedge k{=}Type(g) \wedge (k = AND \vee k = OR \vee k = XOR) \Rightarrow Arity(g,2,1)$

// A circuit has terminals exactly up to its input and output arity.

$$\forall c,i,j \;\; Circuit(c) \wedge Arity(c,i,j) \Rightarrow$$
$$\forall n \;\; (n \leq i \Rightarrow Terminal(In(n,c)) \wedge (n > i \Rightarrow In(n,c) = Nothing)) \wedge$$
$$\forall n \;\; (n \leq j \Rightarrow Terminal(Out(n,c)) \wedge (n > j \Rightarrow Out(n,c) = Nothing))$$

// Gates and terminals are all distinct.

$\forall g,t,s \;\; Gate(g) \wedge Terminal(t) \wedge Signal(s) \Rightarrow g \neq t \wedge g \neq s \wedge t \neq s$

function not predicate

cont'd

// An XOR gate outputs 1 if and only if its inputs are different.

$\forall g \;\; Gate(g) \wedge Type(g) = XOR \Rightarrow (Signal(Out(1,g)) = 1 \Leftrightarrow Signal(In(1,g)) \neq Signal(In(2,g)))$

// An NOT gate's output is different from its input.

$\forall g \;\; Gate(g) \wedge Type(g) = NOT \Rightarrow (Signal(Out(1,g)) = 1 \Leftrightarrow Signal(Out(1,g)) \neq Signal(In(1,g)))$

// All the gates (except for NOT) have two inputs and one output.

$\forall g \;\; Gate(g) \wedge Type(g) = NOT \Rightarrow Arity(g,1,1)$

$\forall g,k \;\; Gate(g) \wedge k{=}Type(g) \wedge (k = AND \vee k = OR \vee k = XOR) \Rightarrow Arity(g,2,1)$

// A circuit has terminals exactly up to its input and output arity.

$$\forall c,i,j \;\; Circuit(c) \wedge Arity(c,i,j) \Rightarrow$$
$$\forall n \;\; (n \leq i \Rightarrow Terminal(In(n,c)) \wedge (n > i \Rightarrow In(n,c) = Nothing)) \wedge$$
$$\forall n \;\; (n \leq j \Rightarrow Terminal(Out(n,c)) \wedge (n > j \Rightarrow Out(n,c) = Nothing))$$

// Gates and terminals are all distinct.

$\forall g,t,s \;\; Gate(g) \wedge Terminal(t) \wedge Signal(s) \Rightarrow g \neq t \wedge g \neq s \wedge t \neq s$

function not predicate

(errors/typos in the text)

cont'd

// An XOR gate outputs 1 if and only if its inputs are different.

$\forall g \;\; Gate(g) \wedge Type(g) = XOR \Rightarrow (Signal(Out(1,g)) = 1 \Leftrightarrow Signal(In(1,g)) \neq Signal(In(2,g)))$

// An NOT gate's output is different from its input.

$\forall g \;\; Gate(g) \wedge Type(g) = NOT \Rightarrow (Signal(Out(1,g)) = 1 \Leftrightarrow Signal(Out(1,g)) \neq Signal(In(1,g)))$

// All the gates (except for NOT) have two inputs and one output.

$\forall g \;\; Gate(g) \wedge Type(g) = NOT \Rightarrow Arity(g,1,1)$

$\forall g,k \;\; Gate(g) \wedge k{=}Type(g) \wedge (k = AND \vee k = OR \vee k = XOR) \Rightarrow Arity(g,2,1)$

// A circuit has terminals exactly up to its input and output arity.

$$\forall c,i,j \;\; Circuit(c) \wedge Arity(c,i,j) \Rightarrow$$
$$\forall n \;\; (n \leq i \Rightarrow Terminal(In(n,c)) \wedge (n > i \Rightarrow In(n,c) = Nothing)) \wedge$$
$$\forall n \;\; (n \leq j \Rightarrow Terminal(Out(n,c)) \wedge (n > j \Rightarrow Out(n,c) = Nothing))$$

// Gates and terminals are all distinct.

$\forall g,t,s \;\; Gate(g) \wedge Terminal(t) \wedge Signal(s) \Rightarrow g \neq t \wedge g \neq s \wedge t \neq s$

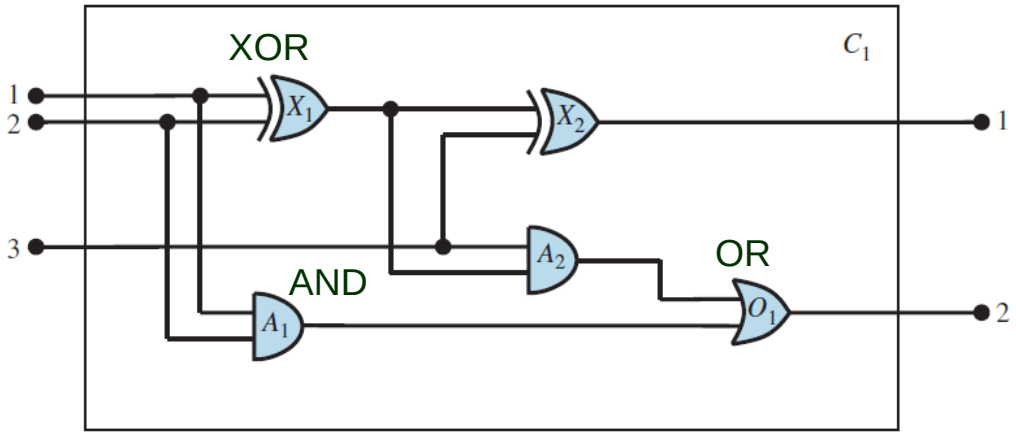
function not predicate

// Gates are circuits.

$\forall g \;\; Gate(g) \Rightarrow Circuit(g)$

(errors/typos in the text)

Encoding a Problem Instance



- Circuit and component gates:

$Circuit(C_1) \wedge Arity(C_1, 3, 2)$

$Gate(X_1) \wedge Type(X_1) = XOR$

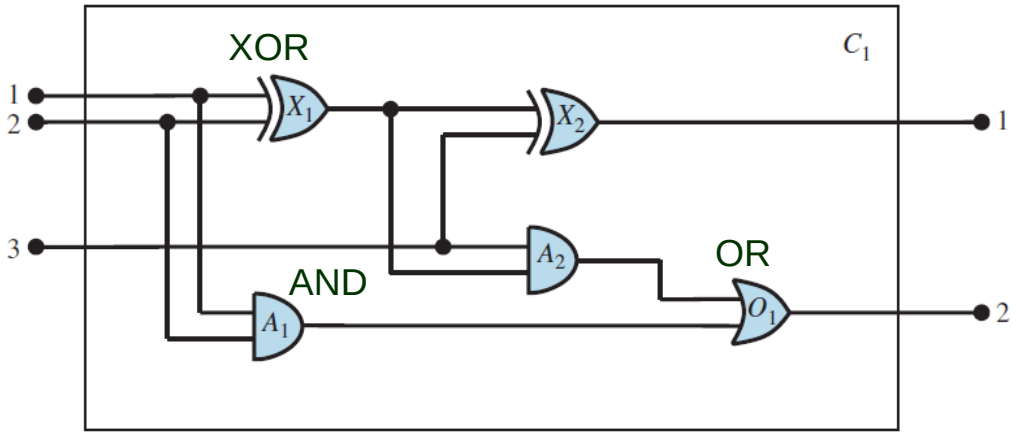
$Gate(X_2) \wedge Type(X_2) = XOR$

$Gate(A_1) \wedge Type(A_1) = AND$

$Gate(A_2) \wedge Type(A_2) = AND$

$Gate(O_1) \wedge Type(O_1) = OR$

Encoding a Problem Instance



- Circuit and component gates:

$Circuit(C_1) \wedge Arity(C_1, 3, 2)$

$Gate(X_1) \wedge Type(X_1) = XOR$

$Gate(X_2) \wedge Type(X_2) = XOR$

$Gate(A_1) \wedge Type(A_1) = AND$

$Gate(A_2) \wedge Type(A_2) = AND$

$Gate(O_1) \wedge Type(O_1) = OR$

- Connections between the circuit and component gates:

$Connected(Out(1, X_1), In(1, X_2))$

$Connected(Out(1, X_1), In(2, A_2))$

$Connected(Out(1, A_2), In(1, O_1))$

$Connected(Out(1, A_1), In(2, O_1))$

$Connected(Out(1, X_2), Out(1, C_1))$

$Connected(Out(1, O_1), Out(2, C_1))$

$Connected(In(1, C_1), In(1, X_1))$

$Connected(In(1, C_1), In(1, A_1))$

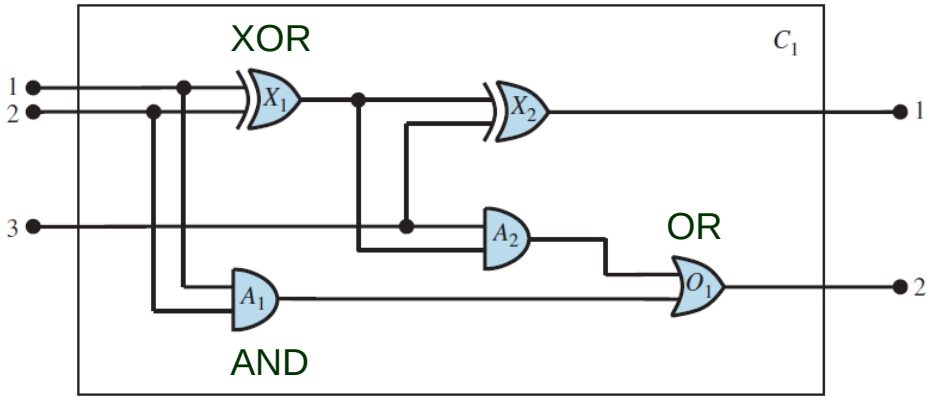
$Connected(In(2, C_1), In(2, X_1))$

$Connected(In(2, C_1), In(2, A_1))$

$Connected(In(3, C_1), In(2, X_2))$

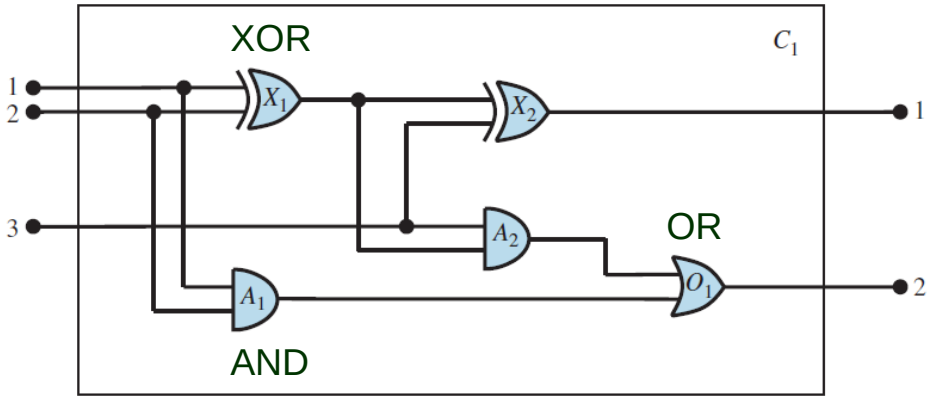
$Connected(In(3, C_1), In(1, A_2))$

Queries



Q. What combinations of inputs would cause the first output of C_1 to be 0 and its second output to be 1?

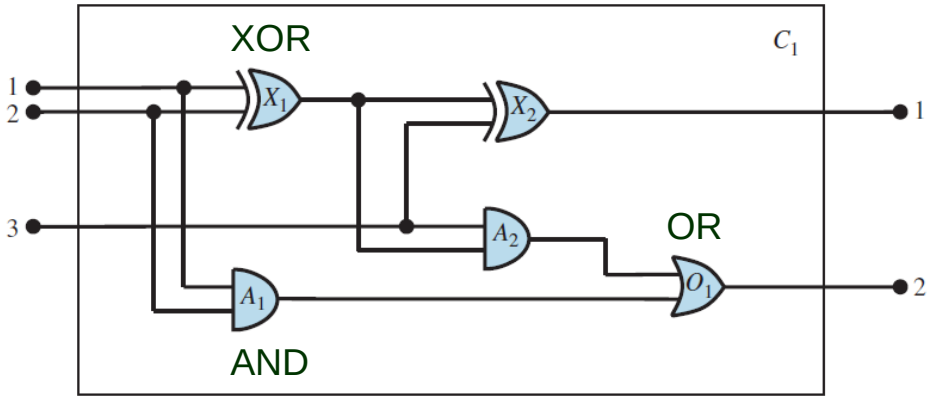
Queries



Q. What combinations of inputs would cause the first output of C_1 to be 0 and its second output to be 1?

$$\exists i_1, i_2, i_3 \ \ \mathit{Signal}\big(\mathit{In}(1, C_1)\big) = i_1 \wedge \mathit{Signal}\big(\mathit{In}(2, C_1)\big) = i_2 \wedge \mathit{Signal}\big(\mathit{In}(3, C_1)\big) = i_3$$
$$\wedge\ \mathit{Signal}\big(\mathit{Out}(1, C_1)\big) = 0 \wedge \mathit{Signal}\big(\mathit{Out}(2, C_1)\big) = 1$$

Queries



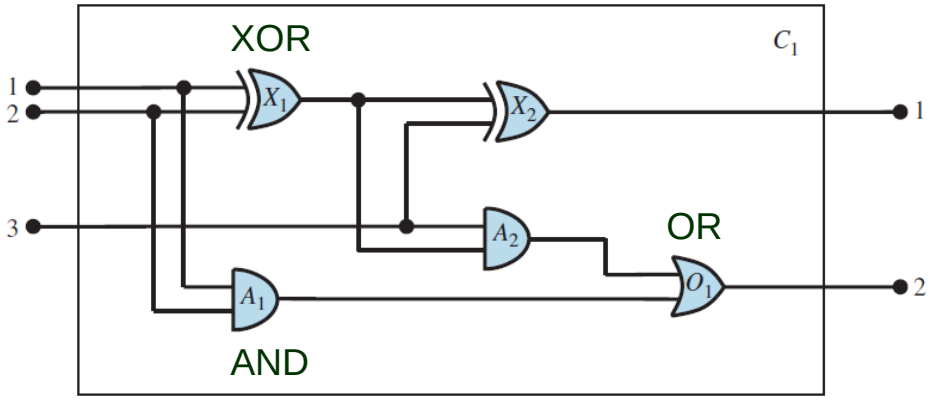
Q. What combinations of inputs would cause the first output of C_1 to be 0 and its second output to be 1?

$$\exists i_1, i_2, i_3 \ \ Signal(In(1, C_1)) = i_1 \wedge Signal(In(2, C_1)) = i_2 \wedge Signal(In(3, C_1)) = i_3$$
$$\wedge\ Signal(Out(1, C_1)) = 0 \wedge Signal(Out(2, C_1)) = 1$$

ASKVARS will give three substitutions as answers.

$$\{i_1/1, i_2/1, i_3/0\} \qquad \{i_1/1, i_2/0, i_3/1\} \qquad \{i_1/0, i_2/1, i_3/1\}$$

Queries



Q. What combinations of inputs would cause the first output of C_1 to be 0 and its second output to be 1?

$$\exists i_1, i_2, i_3 \; \mathit{Signal}(\mathit{In}(1, C_1)) = i_1 \wedge \mathit{Signal}(\mathit{In}(2, C_1)) = i_2 \wedge \mathit{Signal}(\mathit{In}(3, C_1)) = i_3$$
$$\wedge \mathit{Signal}(\mathit{Out}(1, C_1)) = 0 \wedge \mathit{Signal}(\mathit{Out}(2, C_1)) = 1$$

ASKVARS will give three substitutions as answers.

$$\{i_1/1, i_2/1, i_3/0\} \qquad \{i_1/1, i_2/0, i_3/1\} \qquad \{i_1/0, i_2/1, i_3/1\}$$

Debugging: We can also perturb the KB to see what erroneous behaviors would emerge, and then identify missing rules for instance.

III. Propositional vs. First-Order Inference

One way of inference is to convert the first-order KB to propositional logic.

III. Propositional vs. First-Order Inference

One way of inference is to convert the first-order KB to propositional logic.

- Eliminate universal quantifiers.

$\forall x \ \mathit{Human}(x) \Rightarrow \mathit{Fallible}(x)$ // All humans are fallible.

III. Propositional vs. First-Order Inference

One way of inference is to convert the first-order KB to propositional logic.

- Eliminate universal quantifiers.

$\forall x \ \ Human(x) \Rightarrow Fallible(x)$ // All humans are fallible.

We can infer sentences like

$Human(Socrates) \Rightarrow Fallible(Socrates)$

$Human(Einstein) \Rightarrow Fallible(Einstein)$

$Human(Messi) \Rightarrow Fallible(Messi)$

$\vdots$

III. Propositional vs. First-Order Inference

One way of inference is to convert the first-order KB to propositional logic.

- Eliminate universal quantifiers.

$\forall x \ \ Human(x) \Rightarrow Fallible(x)$ // All humans are fallible.

We can infer sentences like

$Human(Socrates) \Rightarrow Fallible(Socrates)$

$Human(Einstein) \Rightarrow Fallible(Einstein)$

$Human(Messi) \Rightarrow Fallible(Messi)$

$\vdots$

From

// All birds are warm-blooded and have wings.

$\forall x \ \ Bird(x) \Rightarrow WarmBlooded(x) \wedge HaveWings(x)$

III. Propositional vs. First-Order Inference

One way of inference is to convert the first-order KB to propositional logic.

- Eliminate universal quantifiers.

$\forall x \ \mathit{Human}(x) \Rightarrow \mathit{Fallible}(x)$ // All humans are fallible.

We can infer sentences like

$\mathit{Human}(\mathit{Socrates}) \Rightarrow \mathit{Fallible}(\mathit{Socrates})$

$\mathit{Human}(\mathit{Einstein}) \Rightarrow \mathit{Fallible}(\mathit{Einstein})$

$\mathit{Human}(\mathit{Messi}) \Rightarrow \mathit{Fallible}(\mathit{Messi})$

$\vdots$

From

// All birds are warm-blooded and have wings.

$\forall x \ \mathit{Bird}(x) \Rightarrow \mathit{WarmBlooded}(x) \wedge \mathit{HaveWings}(x)$

We can infer (if $\mathit{Bird}(\mathit{Ostrich})$ and $\mathit{Bird}(\mathit{Peacock})$ are in the KB):

$\mathit{WarmBlooded}(\mathit{Ostrich}) \wedge \mathit{HaveWings}(\mathit{Ostrich})$

$\mathit{WarmBlooded}(\mathit{Peacock}) \wedge \mathit{HaveWings}(\mathit{Peacock})$

Universal and Existential Instantiations

A *ground term* in FOL is a term without variables.

Universal and Existential Instantiations

A *ground term* in FOL is a term without variables.

Substitute a ground term for a universally quantified variable.

$$\frac{\forall v\ \alpha}{\text{SUBST}(\{v/g\}, \alpha)}$$

Universal and Existential Instantiations

A *ground term* in FOL is a term without variables.

Substitute a ground term for a universally quantified variable.

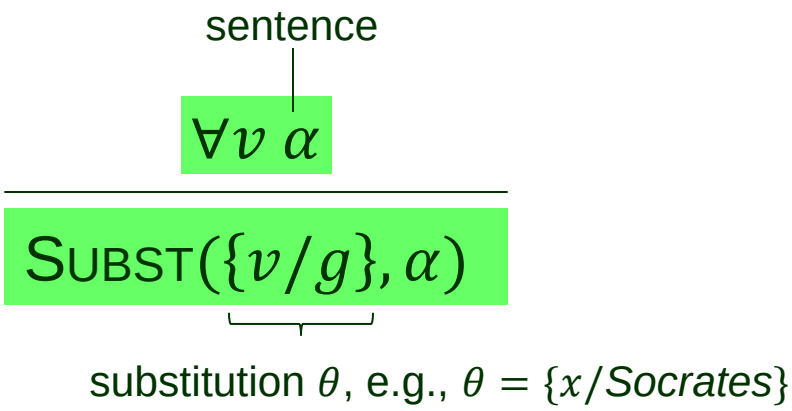
sentence

$$\frac{\forall v\ \alpha}{\text{SUBST}(\{v/g\}, \alpha)}$$

Universal and Existential Instantiations

A *ground term* in FOL is a term without variables.

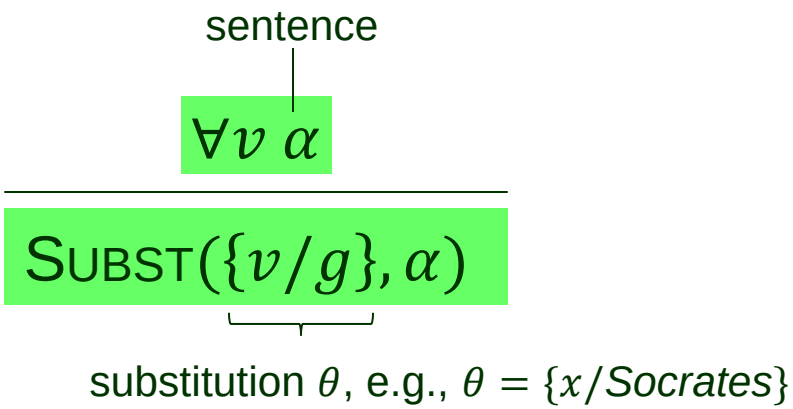
Substitute a ground term for a universally quantified variable.



Universal and Existential Instantiations

A *ground term* in FOL is a term without variables.

Substitute a ground term for a universally quantified variable.



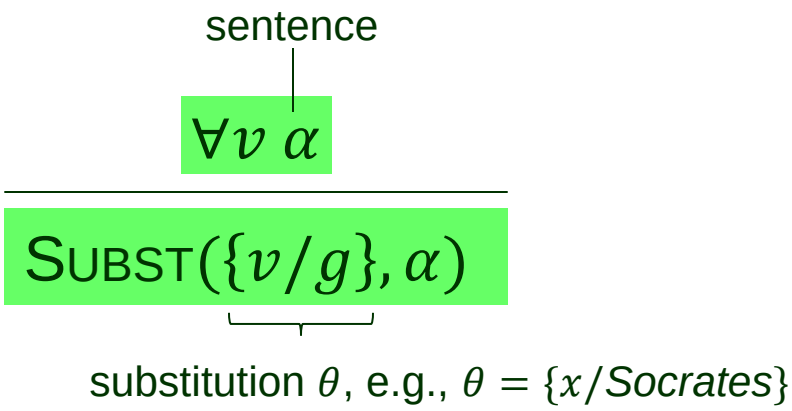
E.g., $\theta = \{x/Ostrich\}$

$\forall x\ \ Bird(x) \Rightarrow WarmBlooded(x) \wedge HaveWings(x)$

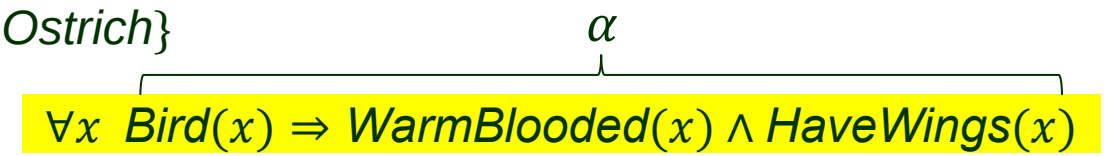
Universal and Existential Instantiations

A *ground term* in FOL is a term without variables.

Substitute a ground term for a universally quantified variable.



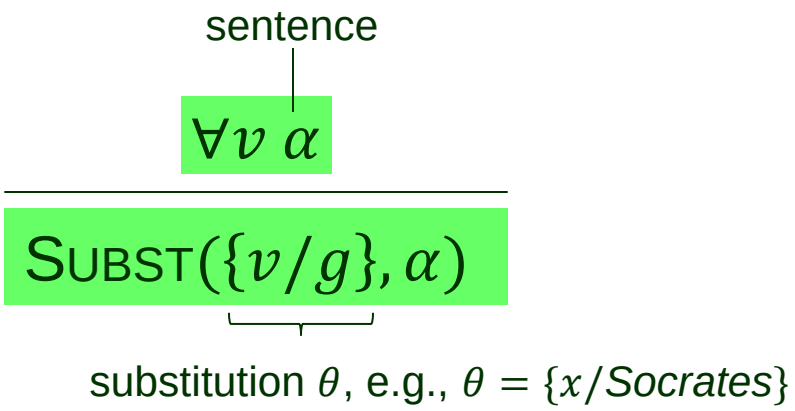
E.g., $\theta = \{x/Ostrich\}$



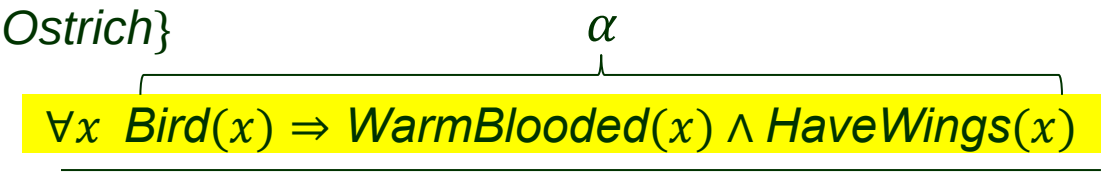
Universal and Existential Instantiations

A *ground term* in FOL is a term without variables.

Substitute a ground term for a universally quantified variable.



E.g., $\theta = \{x/Ostrich\}$



SUBST(θ, α) $\equiv$ $Bird(Ostrich) \Rightarrow WarmBlooded(Ostrich) \wedge HaveWings(Ostrich)$

Existential Instantiation

Substitute a single new constant symbol for an existentially quantified variable.

$$\frac{\exists v\ \alpha}{\textsc{Subst}(\{v/g\}, \alpha)}$$

Existential Instantiation

Substitute a single new constant symbol for an existentially quantified variable.

$$\frac{\exists v\ \alpha}{\text{SUBST}(\{v/g\}, \alpha)}$$

From

$\exists y$ *Mother*$(y, \textit{Liam})$

Existential Instantiation

Substitute a single new constant symbol for an existentially quantified variable.

$$\frac{\exists v\ \alpha}{\text{SUBST}(\{v/g\}, \alpha)}$$

From

$$\exists y\ \ \mathit{Mother}(y, \mathit{Liam})$$

we can infer

$$\mathit{Mother}(\mathit{LiamsMom}, \mathit{Liam})$$

as long as *LiamsMom* does **not** appear elsewhere in the KB.

Existential Instantiation

Substitute a single new constant symbol for an existentially quantified variable.

$$\frac{\exists v\ \alpha}{\text{SUBST}(\{v/g\}, \alpha)}$$

From

$$\exists y\ \ \mathit{Mother}(y, \mathit{Liam})$$

we can infer

$$\mathit{Mother}(\mathit{LiamsMom}, \mathit{Liam})$$

as long as *LiamsMom* does **not** appear elsewhere in the KB.

Skolem constant

Propositionalization (Non-Standard Way)

$\forall x \forall y \quad Ancestor(x, y) \Rightarrow Parent(x, y) \vee \exists z (Ancestor(x, z) \wedge Ancestor(z, y))$

KB:

$Ancestor(John, David)$

$Parent(John, David)$

$Parent(David, Lisa)$

Propositionalization (Non-Standard Way)

$\forall x \forall y \quad Ancestor(x, y) \Rightarrow Parent(x, y) \vee \exists z (Ancestor(x, z) \wedge Ancestor(z, y))$

KB:

$Ancestor(John, David)$

$Parent(John, David)$

$Parent(David, Lisa)$

$Ancestor(John, David)$
$\Rightarrow Parent(John, David) \vee (Ancestor(John, JohnDavidAnc) \wedge Ancestor(JohnDavidAnc, David))$

Propositionalization (Non-Standard Way)

$\forall x \forall y \quad Ancestor(x, y) \Rightarrow Parent(x, y) \vee \exists z (Ancestor(x, z) \wedge Ancestor(z, y))$

KB:

$Ancestor(John, David)$

$Parent(John, David)$

$Parent(David, Lisa)$

$Ancestor(John, David)$
$\Rightarrow Parent(John, David) \vee (Ancestor(John, JohnDavidAnc) \wedge Ancestor(JohnDavidAnc, David))$

$Ancestor(David, John)$
$\Rightarrow Parent(David, John) \vee (Ancestor(David, DavidJohnAnc) \wedge Ancestor(DavidJohnAnc, John))$

$Ancestor(John, Lisa)$
$\Rightarrow Parent(John, Lisa) \vee (Ancestor(John, JohnLisaAnc) \wedge Ancestor(JohnLisaAnc, Lisa))$

$\vdots$

Skolemization (Standard Way)

A more standard way to eliminate an existential quantifier is to introduce a new function symbol, which is, however, not applicable in generating a PL sentence.

$$\exists y \;\; P(y, x_1, \ldots\ldots, x_n)$$

Skolemization (Standard Way)

A more standard way to eliminate an existential quantifier is to introduce a new function symbol, which is, however, not applicable in generating a PL sentence.

$\exists y \;\; P(y, x_1, \ldots\ldots, x_n)$ // y depends on $x_1, \ldots\ldots, x_n$

Skolemization (Standard Way)

A more standard way to eliminate an existential quantifier is to introduce a new function symbol, which is, however, not applicable in generating a PL sentence.

$\exists y \;\; P(y, x_1, \ldots, x_n)$ // y depends on $x_1, \ldots, x_n$

⇩ eliminate y by introducing function f

$P(f(x_1, \ldots, x_n), x_1, \ldots, x_n)$

Skolemization (Standard Way)

A more standard way to eliminate an existential quantifier is to introduce a new function symbol, which is, however, not applicable in generating a PL sentence.

$\exists y \;\; P(y, x_1, \ldots\ldots, x_n)$ // y depends on $x_1, \ldots\ldots, x_n$

⇩ eliminate y by introducing function f

$$P(f(x_1, \ldots\ldots, x_n), x_1, \ldots\ldots, x_n)$$

- That $P(y, x_1, \ldots\ldots, x_n) = \textit{true}$ implicitly defines y as a function of $x_1, \ldots\ldots, x_n$ (analogous to the implicit function theorem in multivariate calculus).

Skolemization (Standard Way)

A more standard way to eliminate an existential quantifier is to introduce a new function symbol, which is, however, not applicable in generating a PL sentence.

$\exists y \;\; P(y, x_1, \ldots, x_n)$ // y depends on $x_1, \ldots, x_n$

⇩ eliminate y by introducing function f

$P(f(x_1, \ldots, x_n), x_1, \ldots, x_n)$

- That $P(y, x_1, \ldots, x_n) = \textit{true}$ implicitly defines y as a function of $x_1, \ldots, x_n$ (analogous to the implicit function theorem in multivariate calculus).

$\exists y \;\; \textit{Mother}(y, \textit{Liam})$

$\exists y \;\; \textit{Mother}(y, \textit{Sophia})$

Skolemization (Standard Way)

A more standard way to eliminate an existential quantifier is to introduce a new function symbol, which is, however, not applicable in generating a PL sentence.

$\exists y \;\; P(y, x_1, \ldots, x_n)$ // y depends on $x_1, \ldots, x_n$

⇓ eliminate y by introducing function f

$P(f(x_1, \ldots, x_n), x_1, \ldots, x_n)$

- That $P(y, x_1, \ldots, x_n) = \textit{true}$ implicitly defines y as a function of $x_1, \ldots, x_n$ (analogous to the implicit function theorem in multivariate calculus).

$\exists y \;\; \textit{Mother}(y, \textit{Liam})$

$\exists y \;\; \textit{Mother}(y, \textit{Sophia})$

⟹ new unary function *mom*()

$\textit{Mother}(\textit{mom}(\textit{Liam}), \textit{Liam})$

$\textit{Mother}(\textit{mom}(\textit{Sophia}), \textit{Sophia})$

Skolemization (Standard Way)

A more standard way to eliminate an existential quantifier is to introduce a new function symbol, which is, however, not applicable in generating a PL sentence.

$\exists y \;\; P(y, x_1, \ldots, x_n)$ // y depends on $x_1, \ldots, x_n$

⇩ eliminate y by introducing function f

$P(f(x_1, \ldots, x_n), x_1, \ldots, x_n)$

- That $P(y, x_1, \ldots, x_n) = \textit{true}$ implicitly defines y as a function of $x_1, \ldots, x_n$ (analogous to the implicit function theorem in multivariate calculus).

$\exists y \;\; \textit{Mother}(y, \textit{Liam})$

$\exists y \;\; \textit{Mother}(y, \textit{Sophia})$

new unary function *mom*() ⟹

$\textit{Mother}(\textit{mom}(\textit{Liam}), \textit{Liam})$

$\textit{Mother}(\textit{mom}(\textit{Sophia}), \textit{Sophia})$

- Advantage: one function instead of two new constants to denote the moms of Liam and Sophia.