Existential quantification and closed vs. open worlds

Dave Dubin

March 2019

Inference rule from last week

```
needstool(Recipe,Tool,Property) :-
   stepin(Step,Recipe),
   utool(Step,Tool),
   realizes(Tool, Property).
```

- If a step in a recipe uses a tool that realizes a property, then the recipe needs that tool for the property.

Inference rule from last week

```
needstool(Recipe,Tool,Property) :-
   stepin(Step,Recipe),
   utool(Step,Tool),
   realizes(Tool, Property).
```

- If a step in a recipe uses a tool that realizes a property, then the recipe needs that tool for the property.
- $\forall w \forall x \forall y \forall z (((Swx \wedge Uwy) \wedge Ryz) \rightarrow Nxyz)$

Inference rule from last week

```
needstool(Recipe,Tool,Property) :-
   stepin(Step,Recipe),
   utool(Step,Tool),
   realizes(Tool, Property).
```

- If a step in a recipe uses a tool that realizes a property, then the recipe needs that tool for the property.
- $\forall w \forall x \forall y \forall z (((Swx \wedge Uwy) \wedge Ryz) \rightarrow Nxyz)$
- However, our data usually doesn't specify both the tool and the property. One or the other is mentioned alone.

Using existential quantification

- With predicate logic we could deduce the existence of a needed tool from a property that it should realize.

Using existential quantification

- With predicate logic we could deduce the existence of a needed tool from a property that it should realize.
- $\forall x \forall y \forall z ((Rxy \wedge Syz) \rightarrow \exists w Nxwz)$

Using existential quantification

- With predicate logic we could deduce the existence of a needed tool from a property that it should realize.
- $\forall x \forall y \forall z((Rxy \wedge Syz) \rightarrow \exists w Nxwz)$
- This is a powerful inference, since it lets us reason about one or more individuals that aren't specified.

Using existential quantification

- With predicate logic we could deduce the existence of a needed tool from a property that it should realize.
- $\forall x \forall y \forall z ((Rxy \wedge Syz) \rightarrow \exists w Nxwz)$
- This is a powerful inference, since it lets us reason about one or more individuals that aren't specified.
- But Datalog doesn't give us full existential quantification for reasoning.

Our approximation using constants

- Datalog doesn't support $\forall x \forall y \forall z((Rxy \wedge Syz) \rightarrow \exists w Nxwz)$

Our approximation using constants

- Datalog doesn't support $\forall x \forall y \forall z((Rxy \wedge Syz) \rightarrow \exists w Nxwz)$
- Our work around is $\forall x \forall y \forall z((Rxy \wedge Syz) \rightarrow Nxaz)$

Our approximation using constants

- Datalog doesn't support $\forall x \forall y \forall z((Rxy \wedge Syz) \rightarrow \exists w Nxwz)$
- Our work around is $\forall x \forall y \forall z((Rxy \wedge Syz) \rightarrow Nxaz)$
- That is to say, if a recipe requires an ingredient and that ingredient satisfies a property, then the recipe needs a tool we call 'something.'

Our approximation using constants

- Datalog doesn't support $\forall x \forall y \forall z((Rxy \wedge Syz) \rightarrow \exists w Nxwz)$
- Our work around is $\forall x \forall y \forall z((Rxy \wedge Syz) \rightarrow Nxaz)$
- That is to say, if a recipe requires an ingredient and that ingredient satisfies a property, then the recipe needs a tool we call 'something.'
- But on this account, 'something' is some particular tool that we're calling 'something.'

Our approximation using constants

- Datalog doesn't support $\forall x \forall y \forall z((Rxy \wedge Syz) \rightarrow \exists w Nxwz)$
- Our work around is $\forall x \forall y \forall z((Rxy \wedge Syz) \rightarrow Nxaz)$
- That is to say, if a recipe requires an ingredient and that ingredient satisfies a property, then the recipe needs a tool we call 'something.'
- But on this account, 'something' is some particular tool that we're calling 'something.'
- And 'something' is always the same something, no matter how many different qualifying properties it's matched with.

Our approximation using constants

- Datalog doesn't support $\forall x \forall y \forall z((Rxy \wedge Syz) \rightarrow \exists w Nxwz)$
- Our work around is $\forall x \forall y \forall z((Rxy \wedge Syz) \rightarrow Nxaz)$
- That is to say, if a recipe requires an ingredient and that ingredient satisfies a property, then the recipe needs a tool we call 'something.'
- But on this account, 'something' is some particular tool that we're calling 'something.'
- And 'something' is always the same something, no matter how many different qualifying properties it's matched with.
- That's not consistent with the semantics of predicate logic.

Our approximation using constants

- Datalog doesn't support $\forall x \forall y \forall z((Rxy \wedge Syz) \rightarrow \exists w Nxwz)$
- Our work around is $\forall x \forall y \forall z((Rxy \wedge Syz) \rightarrow Nxaz)$
- That is to say, if a recipe requires an ingredient and that ingredient satisfies a property, then the recipe needs a tool we call 'something.'
- But on this account, 'something' is some particular tool that we're calling 'something.'
- And 'something' is always the same something, no matter how many different qualifying properties it's matched with.
- That's not consistent with the semantics of predicate logic.

Our approximation using constants

- Datalog doesn't support $\forall x \forall y \forall z((Rxy \wedge Syz) \rightarrow \exists w Nxwz)$
- Our work around is $\forall x \forall y \forall z((Rxy \wedge Syz) \rightarrow Nxaz)$
- That is to say, if a recipe requires an ingredient and that ingredient satisfies a property, then the recipe needs a tool we call 'something.'
- But on this account, 'something' is some particular tool that we're calling 'something.'
- And 'something' is always the same something, no matter how many different qualifying properties it's matched with.
- That's not consistent with the semantics of predicate logic.

```
needstool(Recipe,something,Property) :-
   requires(Recipe,Ingredient),
   satisfies(Ingredient, Property).
```

The closed world assumption

- Many deductive applications are based on a ‘closed world assumption.’

The closed world assumption

- Many deductive applications are based on a ‘closed world assumption.’
- This means that if a statement is not deduced to be true, then that statement should be treated as false.

The closed world assumption

- Many deductive applications are based on a ‘closed world assumption.’
- This means that if a statement is not deduced to be true, then that statement should be treated as false.
- For example, our assertions do not directly state that the chicken soup recipe needs a slow cooker.

The closed world assumption

- Many deductive applications are based on a 'closed world assumption.'
- This means that if a statement is not deduced to be true, then that statement should be treated as false.
- For example, our assertions do not directly state that the chicken soup recipe needs a slow cooker.
- But a substep of a step included in the recipe uses a slow cooker, so the need for the tool can be deduced.

The closed world assumption

- Many deductive applications are based on a ‘closed world assumption.’
- This means that if a statement is not deduced to be true, then that statement should be treated as false.
- For example, our assertions do not directly state that the chicken soup recipe needs a slow cooker.
- But a substep of a step included in the recipe uses a slow cooker, so the need for the tool can be deduced.
- We deduce that some tool is needed for dicing, skinning, and trimming.

The closed world assumption

- Many deductive applications are based on a 'closed world assumption.'
- This means that if a statement is not deduced to be true, then that statement should be treated as false.
- For example, our assertions do not directly state that the chicken soup recipe needs a slow cooker.
- But a substep of a step included in the recipe uses a slow cooker, so the need for the tool can be deduced.
- We deduce that some tool is needed for dicing, skinning, and trimming.
- But we can't deduce that we need a knife for those actions.

The closed world assumption

- Many deductive applications are based on a 'closed world assumption.'
- This means that if a statement is not deduced to be true, then that statement should be treated as false.
- For example, our assertions do not directly state that the chicken soup recipe needs a slow cooker.
- But a substep of a step included in the recipe uses a slow cooker, so the need for the tool can be deduced.
- We deduce that some tool is needed for dicing, skinning, and trimming.
- But we can't deduce that we need a knife for those actions.
- Under a closed world assumption we would take it as false that a knife is needed for this recipe.

The closed world assumption

- The closed world assumption is reasonable in applications where all relevant facts and rules of inference are available.

The closed world assumption

- The closed world assumption is reasonable in applications where all relevant facts and rules of inference are available.
- Systems under the stewardship of one organization with limited use cases and all information stored together are good candidates for the closed world assumption.

The closed world assumption

- The closed world assumption is reasonable in applications where all relevant facts and rules of inference are available.
- Systems under the stewardship of one organization with limited use cases and all information stored together are good candidates for the closed world assumption.
- But these days many information systems rely on loosely coupled, widely distributed data.

The closed world assumption

- The closed world assumption is reasonable in applications where all relevant facts and rules of inference are available.
- Systems under the stewardship of one organization with limited use cases and all information stored together are good candidates for the closed world assumption.
- But these days many information systems rely on loosely coupled, widely distributed data.
- Systems are designed for uses across institutional boundaries.

The closed world assumption

- The closed world assumption is reasonable in applications where all relevant facts and rules of inference are available.
- Systems under the stewardship of one organization with limited use cases and all information stored together are good candidates for the closed world assumption.
- But these days many information systems rely on loosely coupled, widely distributed data.
- Systems are designed for uses across institutional boundaries.
- Stakeholders cooperate on maintaining and integrating machine readable data.

The open world assumption

- With an open world assumption, we can't conclude that failure to deduce a proposition means that the proposition is false.

The open world assumption

- With an open world assumption, we can't conclude that failure to deduce a proposition means that the proposition is false.
- Reasoning must be based on what might be true, consistent with what we know.

The open world assumption

- With an open world assumption, we can't conclude that failure to deduce a proposition means that the proposition is false.
- Reasoning must be based on what might be true, consistent with what we know.
- Understanding how this works requires the concept of an *interpretation.*

Bach on Semantics

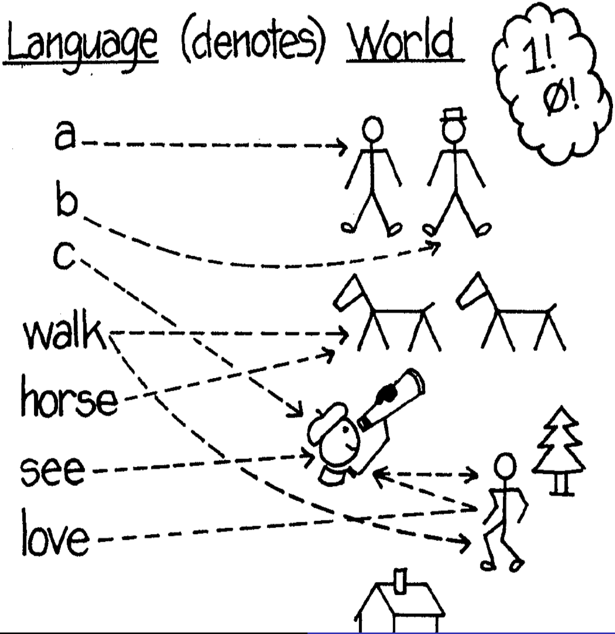
1. The structure of language can each be described using formal systems.

Bach on Semantics

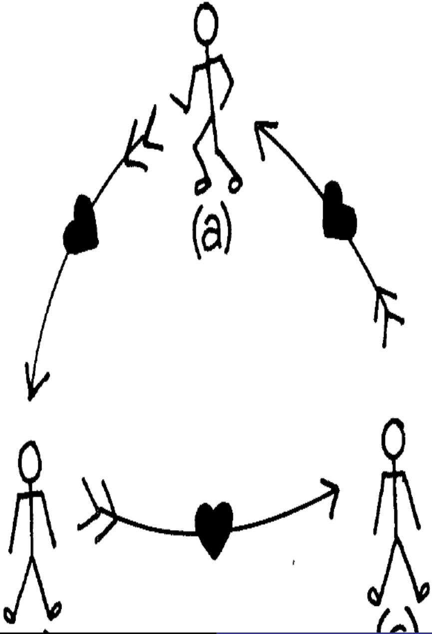
1. The structure of language can each be described using formal systems.
2. Language has meaning.

Bach on Semantics

1. The structure of language can each be described using formal systems.
2. Language has meaning.
3. Meanings are things that aren't language.


Language (denotes) World
1!
Ø!
a
b
c
walk
horse
see
love

Language and the world



Syntax of Bach's logic subset

$\langle var \rangle ::= w|x|y|z$

$\langle con \rangle ::= a|b|c|d$

$\langle ter \rangle ::= \langle var \rangle | \langle con \rangle$

$\langle 1pp \rangle ::= Run|Walk|Happy|Calm$

$\langle 2pp \rangle ::= Love|Kiss|Like|See$

$\langle wff \rangle ::= \langle 1pp \rangle(\langle ter \rangle)$

$\langle wff \rangle ::= \langle 2pp \rangle(\langle ter \rangle, \langle ter \rangle)$

$\langle wff \rangle ::= -\langle wff \rangle$

$\langle wff \rangle ::= (\langle wff \rangle \vee \langle wff \rangle)$

$\langle wff \rangle ::= (\langle wff \rangle \, \& \, \langle wff \rangle)$

$\langle wff \rangle ::= \forall \langle var \rangle \, \langle wff \rangle$

$\langle wff \rangle ::= \exists \langle var \rangle \, \langle wff \rangle$

Notational differences between readings

- The & and $\wedge$ symbols both mean conjunction ("and").

Notational differences between readings

- The & and $\wedge$ symbols both mean conjunction ("and").
- The $\neg$, $\sim$, and $-$ all mean negation ("not").

Notational differences between readings

- The & and $\wedge$ symbols both mean conjunction ("and").
- The $\neg$, $\sim$, and $-$ all mean negation ("not").
- Sometimes predicates are written with parentheses ($P(x, y)$), sometimes not (Pxy).

Notational differences between readings

- The & and $\wedge$ symbols both mean conjunction ("and").
- The $\neg$, $\sim$, and $-$ all mean negation ("not").
- Sometimes predicates are written with parentheses ($P(x, y)$), sometimes not (Pxy).
- In other words, $(Pab \wedge \neg Qbc)$ is the same as $(P(a, b) \ \& \ -Q(b, c))$.

Notational differences between readings

- The & and $\wedge$ symbols both mean conjunction ("and").
- The $\neg$, $\sim$, and $-$ all mean negation ("not").
- Sometimes predicates are written with parentheses ($P(x, y)$), sometimes not (Pxy).
- In other words, $(Pab \wedge \neg Qbc)$ is the same as $(P(a, b)\ \&\ -Q(b, c))$.
- Some authors use D for the domain set and I for the interpretation function.

Notational differences between readings

- The & and $\wedge$ symbols both mean conjunction ("and").
- The $\neg$, $\sim$, and $-$ all mean negation ("not").
- Sometimes predicates are written with parentheses ($P(x, y)$), sometimes not (Pxy).
- In other words, $(Pab \wedge \neg Qbc)$ is the same as $(P(a, b)\ \&\ -Q(b, c))$.
- Some authors use D for the domain set and I for the interpretation function.
- But Bach calls the domain set E (entities) and the interpretation function D (denotes).

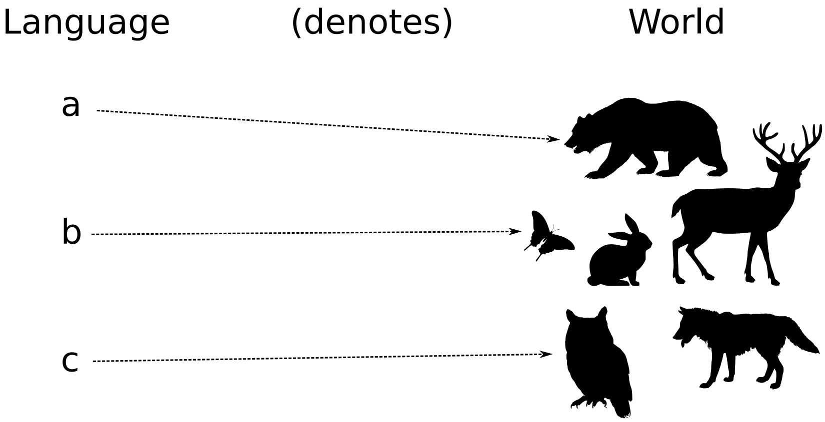
Notational differences between readings

- The & and $\wedge$ symbols both mean conjunction ("and").
- The $\neg$, $\sim$, and $-$ all mean negation ("not").
- Sometimes predicates are written with parentheses ($P(x, y)$), sometimes not (Pxy).
- In other words, $(Pab \wedge \neg Qbc)$ is the same as $(P(a, b)\ \&\ -Q(b, c))$.
- Some authors use D for the domain set and I for the interpretation function.
- But Bach calls the domain set E (entities) and the interpretation function D (denotes).
- I will bring in some details from other readings, but use Bach's notation in this presentation.

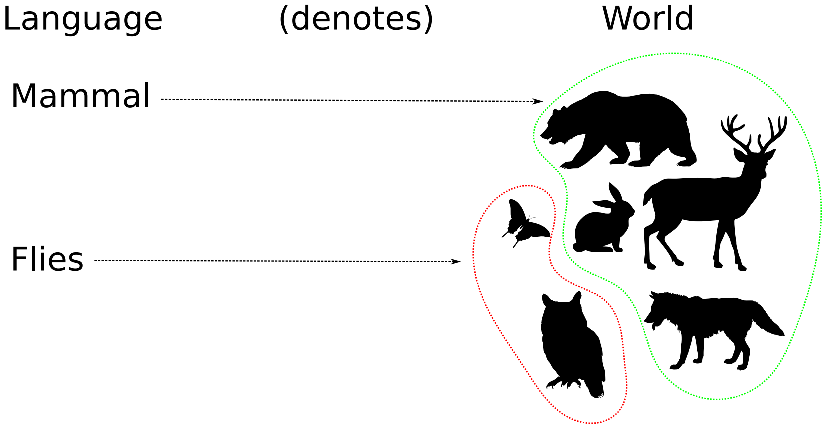
Notational differences between readings

- The & and $\wedge$ symbols both mean conjunction ("and").
- The $\neg$, $\sim$, and $-$ all mean negation ("not").
- Sometimes predicates are written with parentheses ($P(x, y)$), sometimes not (Pxy).
- In other words, $(Pab \wedge \neg Qbc)$ is the same as $(P(a, b)\ \&\ -Q(b, c))$.
- Some authors use D for the domain set and I for the interpretation function.
- But Bach calls the domain set E (entities) and the interpretation function D (denotes).
- I will bring in some details from other readings, but use Bach's notation in this presentation.
- Bach calls his structure an interpretation, so I'll use I for that.

Language and the world



Language and the world

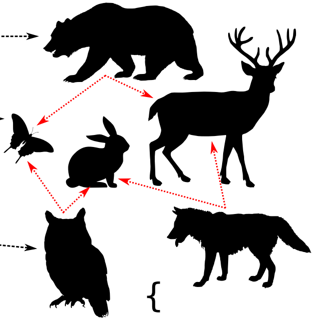


Language (denotes) World

a

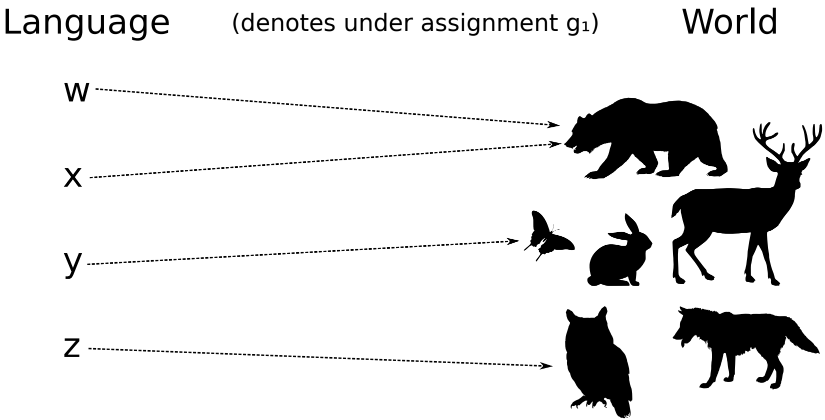
b

c



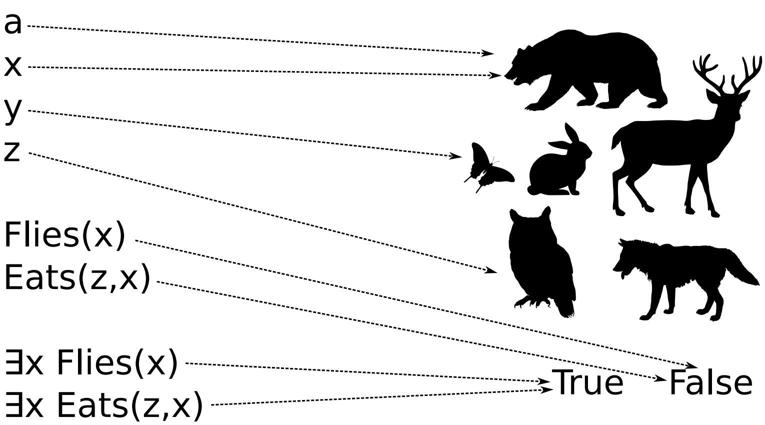
Eats

$\{\langle a_D, b_D\rangle, \langle c_D, b_D\rangle, \ldots\}$


Language
(denotes under assignment g_1)
World
w
x
y
z

Language and the world

Language (denotes under assignment g_1) World



Bach writes:

> *An interpretation is a way of assigning denotations in a certain model structure to expressions in a language.*

- Bach's formalization gives us: $I = \langle M1, D, G \rangle$ where:

Models and interpretations

Bach writes:

> *An interpretation is a way of assigning denotations in a certain model structure to expressions in a language.*

- Bach's formalization gives us: $I = \langle M1, D, G \rangle$ where:
  - $M1$ is the first model structure he proposes;

Models and interpretations

Bach writes:

> *An interpretation is a way of assigning denotations in a certain model structure to expressions in a language.*

- Bach's formalization gives us: $I = \langle M1, D, G \rangle$ where:
  - $M1$ is the first model structure he proposes;
  - G is a set of assignments of values to variables;

Models and interpretations

Bach writes:

An interpretation is a way of assigning denotations in a certain model structure to expressions in a language.

- Bach's formalization gives us: $I = \langle M1, D, G \rangle$ where:
  - $M1$ is the first model structure he proposes;
  - G is a set of assignments of values to variables;
  - D is the denoting evaluation function.

Models and interpretations

Bach writes:

> *An interpretation is a way of assigning denotations in a certain model structure to expressions in a language.*

- Bach's formalization gives us: $I = \langle M1, D, G \rangle$ where:
  - $M1$ is the first model structure he proposes;
  - G is a set of assignments of values to variables;
  - D is the denoting evaluation function.
- Bach's $M1$ is itself a two-place tuple consisting of:

Models and interpretations

Bach writes:

> *An interpretation is a way of assigning denotations in a certain model structure to expressions in a language.*

- Bach's formalization gives us: $I = \langle M1, D, G \rangle$ where:
  - $M1$ is the first model structure he proposes;
  - G is a set of assignments of values to variables;
  - D is the denoting evaluation function.
- Bach's $M1$ is itself a two-place tuple consisting of:
  - Set E of individuals;

Models and interpretations

Bach writes:

> *An interpretation is a way of assigning denotations in a certain model structure to expressions in a language.*

- Bach's formalization gives us: $I = \langle M1, D, G \rangle$ where:
  - $M1$ is the first model structure he proposes;
  - G is a set of assignments of values to variables;
  - D is the denoting evaluation function.
- Bach's $M1$ is itself a two-place tuple consisting of:
  - Set E of individuals;
  - The set of truth values $\{1, 0\}$.

Models and interpretations

Bach writes:

> *An interpretation is a way of assigning denotations in a certain model structure to expressions in a language.*

- Bach's formalization gives us: $I = \langle M1, D, G \rangle$ where:
  - $M1$ is the first model structure he proposes;
  - G is a set of assignments of values to variables;
  - D is the denoting evaluation function.
- Bach's $M1$ is itself a two-place tuple consisting of:
  - Set E of individuals;
  - The set of truth values $\{1, 0\}$.
- Toward the end of the reading he suggests adding a set of times and a set of possible worlds for more expressive languages.

Simple denotations

Individual constants (like a and b) will denote individuals in the domain, so they'll be elements of Bach's set E. The one-place predicates (classes) will denote their extensions (i.e., subsets of the domain). The two-place predicates (binary relations) will denote sets of ordered pairs.

$D(\langle con\rangle) \in E$

$D(\langle 1pp\rangle) \subseteq E$

$D(\langle 2pp\rangle) \subseteq E \times E$

Following a common convention, we adopt the notation φ_D for $D(\varphi)$. So, for example, a_D would be some individual in the Domain set E, and $Love_D$ would be a set of ordered pairs $\langle x, y\rangle$, in each of which x loves y.

Variable assignments

- Let V be the set of variables in the language. Bach's G is a set of variable assignments, and each $g \in G$ is a function from variables ($v \in V$) to individuals in E, i.e.: $g \subseteq V \times E$.

Variable assignments

- Let V be the set of variables in the language. Bach's G is a set of variable assignments, and each $g \in G$ is a function from variables ($v \in V$) to individuals in E, i.e.: $g \subseteq V \times E$.
- For example, suppose $g_1 = \{\langle w, a_D \rangle, \langle x, b_D \rangle, \langle y, c_D \rangle, \langle z, d_D \rangle\}$, but $g_2 = \{\langle w, b_D \rangle, \langle x, c_D \rangle, \langle y, d_D \rangle, \langle z, a_D \rangle\}$.

Variable assignments

- Let V be the set of variables in the language. Bach's G is a set of variable assignments, and each $g \in G$ is a function from variables ($v \in V$) to individuals in E, i.e.: $g \subseteq V \times E$.
- For example, suppose $g_1 = \{\langle w, a_D\rangle, \langle x, b_D\rangle, \langle y, c_D\rangle, \langle z, d_D\rangle\}$, but $g_2 = \{\langle w, b_D\rangle, \langle x, c_D\rangle, \langle y, d_D\rangle, \langle z, a_D\rangle\}$.
- Because our functions are cartesian product subsets, two or more different variables can be mapped to the same individual. For example, maybe $g_3 = \{\langle w, a_D\rangle, \langle x, a_D\rangle, \langle y, b_D\rangle, \langle z, a_D\rangle\}$ and $g_4 = \{\langle w, c_D\rangle, \langle x, c_D\rangle, \langle y, c_D\rangle, \langle z, c_D\rangle\}$. But they're functions, so every variable in the domain will be included.

Variable assignments

- Let V be the set of variables in the language. Bach's G is a set of variable assignments, and each $g \in G$ is a function from variables ($v \in V$) to individuals in E, i.e.: $g \subseteq V \times E$.
- For example, suppose $g_1 = \{\langle w, a_D \rangle, \langle x, b_D \rangle, \langle y, c_D \rangle, \langle z, d_D \rangle\}$, but $g_2 = \{\langle w, b_D \rangle, \langle x, c_D \rangle, \langle y, d_D \rangle, \langle z, a_D \rangle\}$.
- Because our functions are cartesian product subsets, two or more different variables can be mapped to the same individual. For example, maybe $g_3 = \{\langle w, a_D \rangle, \langle x, a_D \rangle, \langle y, b_D \rangle, \langle z, a_D \rangle\}$ and $g_4 = \{\langle w, c_D \rangle, \langle x, c_D \rangle, \langle y, c_D \rangle, \langle z, c_D \rangle\}$. But they're functions, so every variable in the domain will be included.
- The expression $g[v{:=}e]$ is understood to mean "the variable assignment that is completely like g except that v now gets the value e (where g might have assigned a different value)."

Variable assignments

- Let V be the set of variables in the language. Bach's G is a set of variable assignments, and each $g \in G$ is a function from variables ($v \in V$) to individuals in E, i.e.: $g \subseteq V \times E$.
- For example, suppose $g_1 = \{\langle w, a_D\rangle, \langle x, b_D\rangle, \langle y, c_D\rangle, \langle z, d_D\rangle\}$, but $g_2 = \{\langle w, b_D\rangle, \langle x, c_D\rangle, \langle y, d_D\rangle, \langle z, a_D\rangle\}$.
- Because our functions are cartesian product subsets, two or more different variables can be mapped to the same individual. For example, maybe $g_3 = \{\langle w, a_D\rangle, \langle x, a_D\rangle, \langle y, b_D\rangle, \langle z, a_D\rangle\}$ and $g_4 = \{\langle w, c_D\rangle, \langle x, c_D\rangle, \langle y, c_D\rangle, \langle z, c_D\rangle\}$. But they're functions, so every variable in the domain will be included.
- The expression $g[v{:=}e]$ is understood to mean "the variable assignment that is completely like g except that v now gets the value e (where g might have assigned a different value)."
- For example, $g_1[x{:=}d_D] = \{\langle w, a_D\rangle, \langle x, d_D\rangle, \langle y, c_D\rangle, \langle z, d_D\rangle\}$

Denotation, support, and truth

An expression in our language will be true or false in an interpretation, relative to a particular assignment of individuals to variables. So we write $I \models_g \varphi$ to mean that assignment g *satisfies* expression φ in interpretation I (or, in other words, φ is true in interpretation I under assignment g).

- $[\![v]\!]^g_D = g(v)$

Denotation, support, and truth

An expression in our language will be true or false in an interpretation, relative to a particular assignment of individuals to variables. So we write $I \models_g \varphi$ to mean that assignment g *satisfies* expression φ in interpretation I (or, in other words, φ is true in interpretation I under assignment g).

- $[\![v]\!]_D^g = g(v)$
- $[\![c]\!]_D^g = c_D$

Denotation, support, and truth

An expression in our language will be true or false in an interpretation, relative to a particular assignment of individuals to variables. So we write $I \models_g \varphi$ to mean that assignment g *satisfies* expression φ in interpretation I (or, in other words, φ is true in interpretation I under assignment g).

- $[\![v]\!]_D^g = g(v)$
- $[\![c]\!]_D^g = c_D$
- $I \models_g P(t)$ iff $[\![t]\!]_D^g \in P_D$

Denotation, support, and truth

An expression in our language will be true or false in an interpretation, relative to a particular assignment of individuals to variables. So we write $I \models_g \varphi$ to mean that assignment g *satisfies* expression φ in interpretation I (or, in other words, φ is true in interpretation I under assignment g).

- $[\![v]\!]_D^g = g(v)$
- $[\![c]\!]_D^g = c_D$
- $I \models_g P(t)$ iff $[\![t]\!]_D^g \in P_D$
- $I \models_g R(s,t)$ iff $\langle [\![s]\!]_D^g, [\![t]\!]_D^g \rangle \in R_D$

Denotation, support, and truth

An expression in our language will be true or false in an interpretation, relative to a particular assignment of individuals to variables. So we write $I \models_g \varphi$ to mean that assignment g *satisfies* expression φ in interpretation I (or, in other words, φ is true in interpretation I under assignment g).

- $[\![v]\!]^g_D = g(v)$
- $[\![c]\!]^g_D = c_D$
- $I \models_g P(t)$ iff $[\![t]\!]^g_D \in P_D$
- $I \models_g R(s,t)$ iff $\langle [\![s]\!]^g_D, [\![t]\!]^g_D \rangle \in R_D$
- $I \models_g -\varphi$ iff it is not the case that $I \models_g \varphi$

Denotation, support, and truth

An expression in our language will be true or false in an interpretation, relative to a particular assignment of individuals to variables. So we write $I \models_g \varphi$ to mean that assignment g *satisfies* expression φ in interpretation I (or, in other words, φ is true in interpretation I under assignment g).

- $[\![v]\!]^g_D = g(v)$
- $[\![c]\!]^g_D = c_D$
- $I \models_g P(t)$ iff $[\![t]\!]^g_D \in P_D$
- $I \models_g R(s,t)$ iff $\langle [\![s]\!]^g_D, [\![t]\!]^g_D \rangle \in R_D$
- $I \models_g -\varphi$ iff it is not the case that $I \models_g \varphi$
- $I \models_g (\varphi_1 \ \&\ \varphi_2)$ iff $I \models_g \varphi_1$ and $I \models_g \varphi_2$

Denotation, support, and truth

An expression in our language will be true or false in an interpretation, relative to a particular assignment of individuals to variables. So we write $I \models_g \varphi$ to mean that assignment g *satisfies* expression φ in interpretation I (or, in other words, φ is true in interpretation I under assignment g).

- $[\![v]\!]^g_D = g(v)$
- $[\![c]\!]^g_D = c_D$
- $I \models_g P(t)$ iff $[\![t]\!]^g_D \in P_D$
- $I \models_g R(s,t)$ iff $\langle [\![s]\!]^g_D, [\![t]\!]^g_D \rangle \in R_D$
- $I \models_g -\varphi$ iff it is not the case that $I \models_g \varphi$
- $I \models_g (\varphi_1 \;\&\; \varphi_2)$ iff $I \models_g \varphi_1$ and $I \models_g \varphi_2$
- $I \models_g (\varphi_1 \vee \varphi_2)$ iff $I \models_g \varphi_1$ or $I \models_g \varphi_2$

Denotation, support, and truth

An expression in our language will be true or false in an interpretation, relative to a particular assignment of individuals to variables. So we write $I \models_g \varphi$ to mean that assignment g *satisfies* expression φ in interpretation I (or, in other words, φ is true in interpretation I under assignment g).

- $[\![v]\!]^g_D = g(v)$
- $[\![c]\!]^g_D = c_D$
- $I \models_g P(t)$ iff $[\![t]\!]^g_D \in P_D$
- $I \models_g R(s,t)$ iff $\langle [\![s]\!]^g_D, [\![t]\!]^g_D \rangle \in R_D$
- $I \models_g -\varphi$ iff it is not the case that $I \models_g \varphi$
- $I \models_g (\varphi_1 \,\&\, \varphi_2)$ iff $I \models_g \varphi_1$ and $I \models_g \varphi_2$
- $I \models_g (\varphi_1 \vee \varphi_2)$ iff $I \models_g \varphi_1$ or $I \models_g \varphi_2$
- $I \models_g \forall v \varphi$ iff for all $e \in E$ it holds that $I \models_{g[v:=e]} \varphi$

Denotation, support, and truth

An expression in our language will be true or false in an interpretation, relative to a particular assignment of individuals to variables. So we write $I \models_g \varphi$ to mean that assignment g *satisfies* expression φ in interpretation I (or, in other words, φ is true in interpretation I under assignment g).

- $[\![v]\!]^g_D = g(v)$
- $[\![c]\!]^g_D = c_D$
- $I \models_g P(t)$ iff $[\![t]\!]^g_D \in P_D$
- $I \models_g R(s,t)$ iff $\langle [\![s]\!]^g_D, [\![t]\!]^g_D \rangle \in R_D$
- $I \models_g -\varphi$ iff it is not the case that $I \models_g \varphi$
- $I \models_g (\varphi_1 \; \& \; \varphi_2)$ iff $I \models_g \varphi_1$ and $I \models_g \varphi_2$
- $I \models_g (\varphi_1 \vee \varphi_2)$ iff $I \models_g \varphi_1$ or $I \models_g \varphi_2$
- $I \models_g \forall v \varphi$ iff for all $e \in E$ it holds that $I \models_{g[v:=e]} \varphi$
- $I \models_g \exists v \varphi$ iff for at least one $e \in E$ it holds that $I \models_{g[v:=e]} \varphi$