Formalization of the Resolution Calculus for First-Order Logic

Anders Schlichtkrull

DTU Compute
Department of Applied Mathematics and Computer Science

The resolution calculus for first-order logic

The resolution calculus for first-order logic

- is a proof calculus for FO CNF formulas. $\mathrm{p}(x) \wedge (\mathrm{q}(y) \vee \mathrm{r}(x))$

The resolution calculus for first-order logic

- is a proof calculus for FO CNF formulas. $\mathrm{p}(x) \wedge (\mathrm{q}(y) \vee \mathrm{r}(x))$
  - plain logic without types, sorts, equality

The resolution calculus for first-order logic

- is a proof calculus for FO CNF formulas. $\mathrm{p}(x) \wedge (\mathrm{q}(y) \vee \mathrm{r}(x))$
  - plain logic without types, sorts, equality

- is a refutation proof calculus. $P \vdash \bot$

The resolution calculus for first-order logic

- is a proof calculus for FO CNF formulas.
  - plain logic without types, sorts, equality

$$\mathrm{p}(x) \wedge (\mathrm{q}(y) \vee \mathrm{r}(x))$$

- is a refutation proof calculus.

$$P \vdash \bot$$

- was introduced by
  J. A. Robinson, J. ACM, 1965.

The resolution calculus for first-order logic

- is a proof calculus for FO CNF formulas.
  - plain logic without types, sorts, equality

$$\mathrm{p}(x) \wedge (\mathrm{q}(y) \vee \mathrm{r}(x))$$

- is a refutation proof calculus.

$$P \vdash \bot$$

- was introduced by
  J. A. Robinson, J. ACM, 1965.

1930-2016

The resolution calculus for first-order logic

- is a proof calculus for FO CNF formulas.
  - plain logic without types, sorts, equality

$$\mathrm{p}(x) \wedge (\mathrm{q}(y) \vee \mathrm{r}(x))$$

- is a refutation proof calculus.

$$P \vdash \bot$$

- was introduced by
  J. A. Robinson, J. ACM, 1965.

1930-2016

- is used in automatic theorem provers (e.g. E, SPASS, Vampire).

The resolution calculus for propositional logic

$$\frac{A \qquad A \to C_2}{C_2}$$

The resolution calculus for propositional logic

$$\frac{\neg C_1 \rightarrow A \qquad A \rightarrow C_2}{\neg C_1 \rightarrow C_2}$$

The resolution calculus for propositional logic

$$\frac{\neg C_1 \rightarrow A \qquad A \rightarrow C_2}{\neg C_1 \rightarrow C_2} \qquad \frac{C_1 \vee A \qquad \neg A \vee C_2}{C_1 \vee C_2}$$

The resolution calculus for propositional logic

$$\frac{\neg C_1 \rightarrow A \qquad A \rightarrow C_2}{\neg C_1 \rightarrow C_2} \qquad \frac{C_1 \vee A \qquad \neg A \vee C_2}{C_1 \vee C_2}$$

Clashing literals

Motivation

IsaFoL project
Isabelle Formalization of Logic

The formalization is part of IsaFoL.

IsaFoL = library of basic results in automated reasoning.

New calculi or calculus variants can be easily developed directly in Isabelle.

IsaFoL

- Completeness of FOL

  Blanchette, Popescu, Traytel (IJCAR 2014)

- CDCL with extensions

  Blanchette, Fleury, Weidenbach (IJCAR 2016)

- FO resolution

  Schlichtkrull (ITP 2016)

IsaFoL

- Completeness of FOL

  Blanchette, Popescu, Traytel (IJCAR 2014)

- CDCL with extensions

  Blanchette, Fleury, Weidenbach (IJCAR 2016)

- FO resolution

  Schlichtkrull (ITP 2016)

Related work

- FO model theory
  Harrison in HOL Light (TPHOL 1998)
- FO (but no terms) sequent calculus
  Margetson, Ridge in Isabelle/HOL (AFP 2004)
- FO (but no terms) verified prover
  Margetson, Ridge in Isabelle/HOL (TPHOL 2005)
- FO sequent calculus
  Brasenmann, Koepke in Mizar (Formalized Mathematics 2005)
- Soundness of HOL Light
  Harrison in HOL Light (IJCAR 2006)
- FO natural deduction
  Berghofer in Isabelle/HOL (AFP 2007)

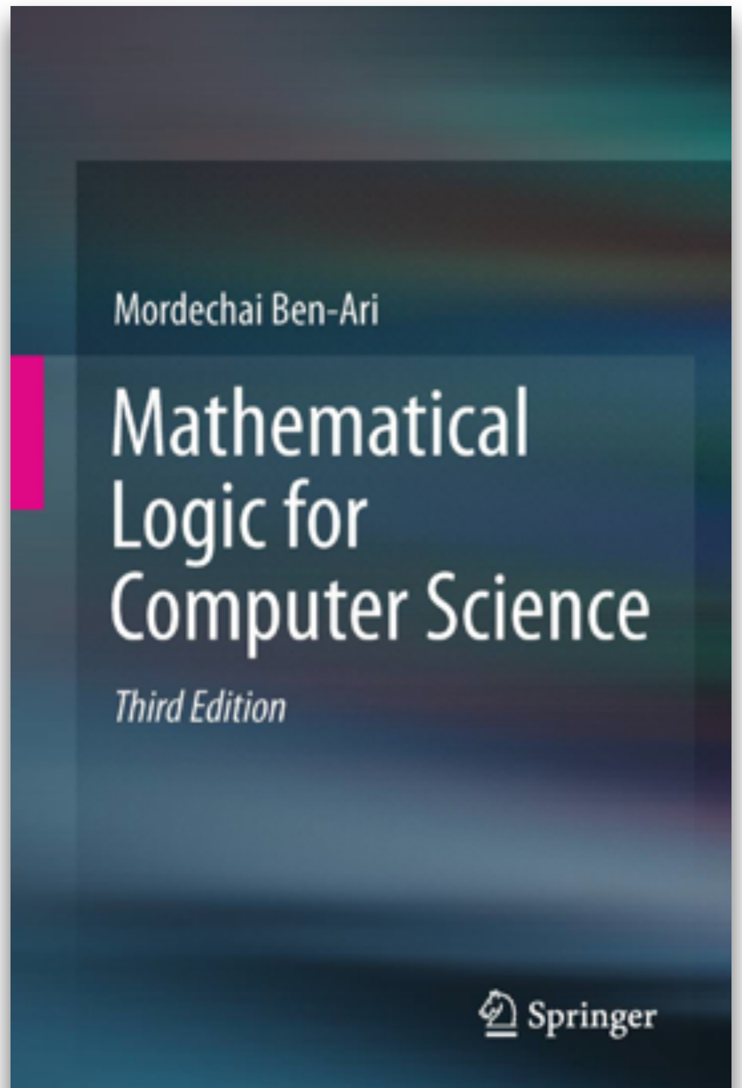
...

Related work

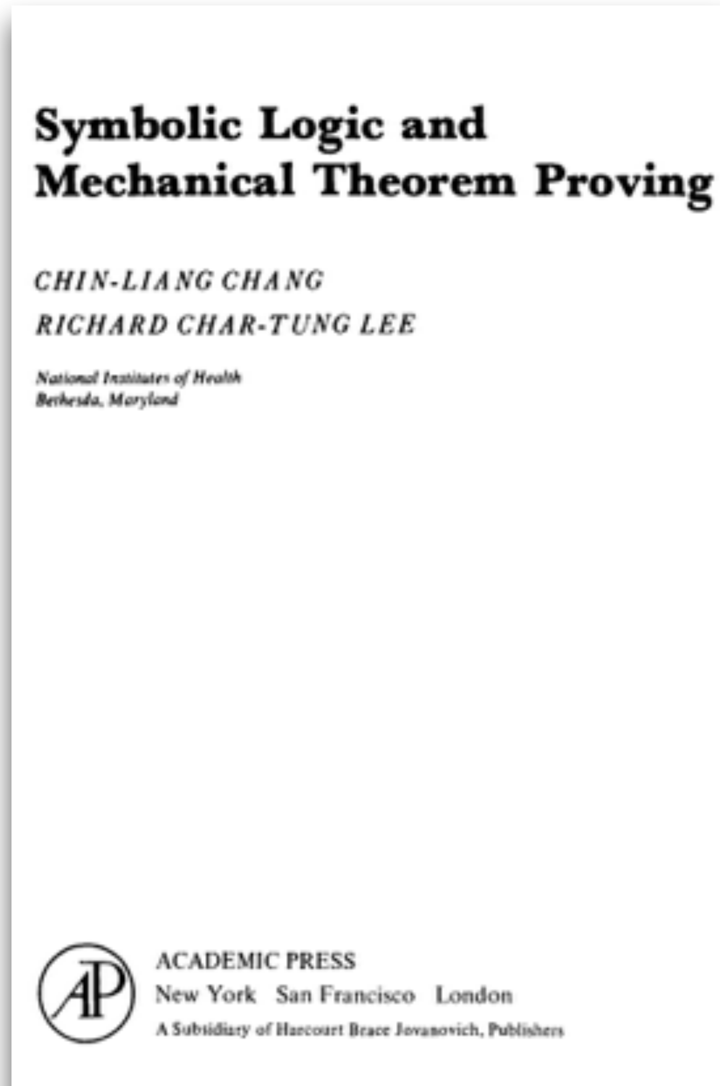
…

- Constructive completeness proofs
  Illik in Coq (PhD thesis 2010)
- FO sequent calculus and uncountable languages
  Schlöder, Koepke in Mizar (Formalized Mathematics 2012)
- Gödel's incompleteness
  Paulson in Isabelle/HOL (JAR 2015)
- Soundness of HOL Light with definitions
  Kumar, Arthan, Myreen, Owens (JAR 2016)
- The Incredible Proof Machine
  Breitner, Lohner in Isabelle/HOL (ITP 2016)
- FO axiomatic system (soundness only)
  Jensen, Schlichtkrull, Villadsen in Isabelle/HOL (Isabelle Workshop 2016)

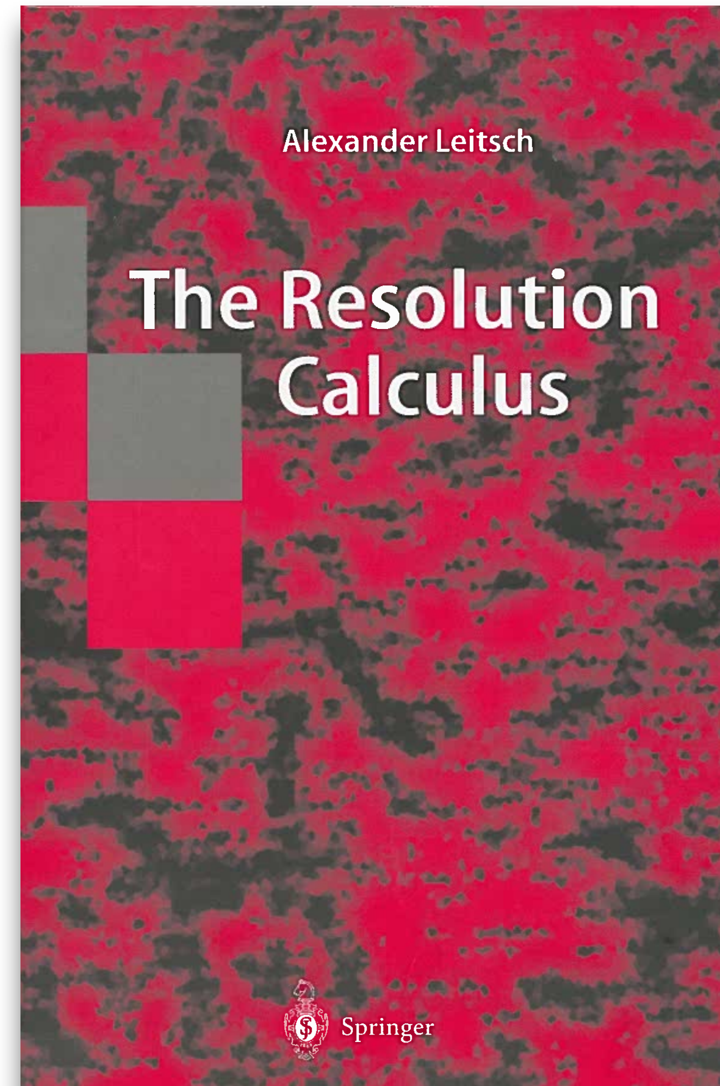
Books I followed



Ben-Ari



Chang and Lee



Leitsch

Tools I used

- Isabelle/jEdit
- Isar
- Proof methods of Isabelle: auto, blast, metis
- Sledgehammer

Clausal first-order logic

Terms: x; y; f(c, x); f(y, f(x, c))

```
datatype fterm =
  Var var-sym
| Fun fun-sym (fterm list)
```

Herbrand (ground) terms: c; d; f(c, d); f(d, f(c, c))

```
datatype hterm =
  HFun fun-sym (hterm list)
```

Clausal first-order logic

Clausal first-order logic

Atoms: $\mathrm{p}(\mathrm{c}, x); \mathrm{q}(\mathrm{d})$

```
type-synonym 't atom = pred-sym * 't list
```

Clausal first-order logic

Atoms: $\mathrm{p}(\mathrm{c}, x); \mathrm{q}(\mathrm{d})$

```
type-synonym 't atom = pred-sym * 't list
```

Literals: $\mathrm{p}(\mathrm{c}, x); \neg\mathrm{q}(\mathrm{d})$

```
datatype 't literal =
  Pos pred-sym ('t list)
| Neg pred-sym ('t list)
```

Clausal first-order logic

Atoms: $\mathrm{p}(\mathrm{c}, x)$; $\mathrm{q}(\mathrm{d})$

```
type-synonym 't atom = pred-sym * 't list
```

Literals: $\mathrm{p}(\mathrm{c}, x)$; $\neg\mathrm{q}(\mathrm{d})$

```
datatype 't literal =
  Pos pred-sym ('t list)
| Neg pred-sym ('t list)
```

Clauses: $\forall x\ y\ z.\ \mathrm{p}(x, y) \lor \mathrm{q}(z) \lor \mathrm{q}(\mathrm{a})$

```
type-synonym 't clause = 't literal set
```

From propositional resolution to FO resolution

$$\frac{r \lor p \qquad \neg r \lor q}{p \lor q}$$

$$\frac{\{r, p\} \quad \{\neg r, q\}}{\{p, q\}}$$

From propositional resolution to FO resolution

$$\frac{r \vee p \qquad \neg r \vee q}{p \vee q}$$

$$\frac{\{r, p\} \quad \{\neg r, q\}}{\{p, q\}}$$

$$\frac{\{r(x), r(y), p(y)\} \quad \{\neg r(c), q\}}{???}$$

Machinery

Machinery

Complement of a literal:

$$\mathrm{p}(x, y)^{\mathrm{C}} = \neg\mathrm{p}(x, y); \ \ \neg\mathrm{q}(\mathrm{f}(x))^{\mathrm{C}} = \mathrm{q}(\mathrm{f}(x))$$

```
fun complement :: 't literal ⇒ 't literal where
  (Pos P ts)ᶜ = Neg P ts
| (Neg P ts)ᶜ = Pos P ts
```

Machinery

Complement of a literal:

$$p(x, y)^C = \neg p(x, y); \;\; \neg q(f(x))^C = q(f(x))$$

```
fun complement :: 't literal ⇒ 't literal where
  (Pos P ts)ᶜ = Neg P ts
| (Neg P ts)ᶜ = Pos P ts
```

Complement of a set of literals:

$$\{p(x, y), \neg q(f(x))\}^C = \{\neg p(x, y), q(f(x))\}$$

```
abbreviation complements :: 't literal set ⇒ 't literal set where
  Lᶜ ≡ complement ` L
```

Machinery

Machinery

Substitutions:

$$\{x \mapsto \mathrm{c}, y \mapsto \mathrm{d}\}; \{x \mapsto \mathrm{f}(x, y), z \mapsto y\}$$

```
type_synonym substitution = var-sym ⇒ fterm
```

Machinery

Substitutions:

$$\{x \mapsto \mathrm{c}, y \mapsto \mathrm{d}\}; \{x \mapsto \mathrm{f}(x, y), z \mapsto y\}$$

```
type_synonym substitution = var-sym ⇒ fterm
```

Application:

$$\mathrm{f}(x, \mathrm{g}(y)) \cdot \{x \mapsto \mathrm{c}, y \mapsto \mathrm{d}\} = \mathrm{f}(\mathrm{c}, \mathrm{g}(d))$$

```
fun sub  :: fterm ⇒ substitution ⇒ fterm where
  (Var x) · σ = σ x
| (Fun f ts) · σ = Fun f (map (λt. t · σ) ts)
```

Machinery

Machinery

Unifier:

$\{\mathrm{p}(x, y), \mathrm{p}(z, \mathrm{c})\}$ has unifier $\{x \mapsto \mathrm{c}, y \mapsto \mathrm{c}, z \mapsto \mathrm{c}\}$

```
definition unifier :: substitution ⇒ fterm literal set ⇒ bool
where
  unifier σ L ⟷ (∃l'. ∀l ∈ L. l · σ = l')
```

Machinery

Unifier:

$\{\mathrm{p}(x, y), \mathrm{p}(z, \mathrm{c})\}$ has unifier $\{x \mapsto \mathrm{c}, y \mapsto \mathrm{c}, z \mapsto \mathrm{c}\}$

```
definition unifier :: substitution ⇒ fterm literal set ⇒ bool
where
  unifier σ L ⟷ (∃l'. ∀l ∈ L. l · σ = l')
```

Most general unifier:

$\{\mathrm{p}(x, y), \mathrm{p}(z, \mathrm{c})\}$ has MGU $\{x \mapsto x, y \mapsto \mathrm{c}, z \mapsto x\}$

```
definition mgu :: substitution ⇒ fterm literal set ⇒ bool where
  mgu σ L ⟷ unifier σ L ∧ (∀u. unifier u L ⟶ (∃i. u = σ · i))
```

FO resolution

$$\frac{C_1 \qquad\qquad C_2}{((C_1 - L_1) \cup (C_2 - L_2)) \cdot \sigma}$$

C_1 and C_2 share no variables,
$L_1 \subseteq C_1,\ L_2 \subseteq C_2,$
σ MGU for $L_1 \cup L_2{}^{\mathrm{c}}$

FO resolution

$$\frac{C_1 \qquad\qquad C_2}{((C_1 - L_1) \cup (C_2 - L_2)) \cdot \sigma}$$

C_1 and C_2 share no variables,
$L_1 \subseteq C_1,\ L_2 \subseteq C_2$,
σ MGU for $L_1 \cup L_2^{\mathrm{c}}$

E.g. we can resolve

$$\frac{\{\mathrm{r}(x), \mathrm{r}(y), \mathrm{p}(y)\} \quad \{\neg\mathrm{r}(\mathrm{c}), \mathrm{q}\}}{\{\mathrm{p}(\mathrm{c}), \mathrm{q}\}}$$

because $\{\mathrm{r}(x), \mathrm{r}(y)\} \cup \{\mathrm{r}(\mathrm{c})\}$ has MGU $\{x \mapsto \mathrm{c}, y \mapsto \mathrm{c}\}$

Formalization of FO resolution

Formalization of FO resolution

```
definition applicable C₁ C₂ L₁ L₂ σ ⟷
      C₁ ≠ {} ∧ C₂ ≠ {} ∧ L₁ ≠ {} ∧ L₂ ≠ {}
    ∧ vars C₁ ∩ vars C₂ = {}
    ∧ L₁ ⊆ C₁ ∧ L₂ ⊆ C₂
    ∧ mgu σ (L₁ ∪ L₂ ᶜ)"
```

Formalization of FO resolution

```
definition applicable C₁ C₂ L₁ L₂ σ ⟷
       C₁ ≠ {} ∧ C₂ ≠ {} ∧ L₁ ≠ {} ∧ L₂ ≠ {}
     ∧ vars C₁ ∩ vars C₂ = {}
     ∧ L₁ ⊆ C₁ ∧ L₂ ⊆ C₂
     ∧ mgu σ (L₁ ∪ L₂ ᶜ)"

definition resolution C₁ C₂ L₁ L₂ σ = ((C₁ - L₁) ∪ (C₂ - L₂)) · σ
```

Formalization of FO resolution

```
definition applicable C₁ C₂ L₁ L₂ σ ⟷
       C₁ ≠ {} ∧ C₂ ≠ {} ∧ L₁ ≠ {} ∧ L₂ ≠ {}
     ∧ vars C₁ ∩ vars C₂ = {}
     ∧ L₁ ⊆ C₁ ∧ L₂ ⊆ C₂
     ∧ mgu σ (L₁ ∪ L₂ ᶜ)"

definition resolution C₁ C₂ L₁ L₂ σ = ((C₁ - L₁) ∪ (C₂ - L₂)) · σ

inductive resolution_step
  :: fterm clause set ⇒ fterm clause set ⇒ bool where
  resolution_rule:
   C₁ ∈ Cs ⟹ C₂ ∈ Cs ⟹ applicable C₁ C₂ L₁ L₂ σ ⟹
      resolution_step Cs (Cs ∪ {resolution C₁ C₂ L₁ L₂ σ})
| standardize_apart:
   C ∈ Cs ⟹ var_renaming_of C C' ⟹ resolution_step Cs (Cs ∪ {C'})
```

Formalization of FO resolution

```
definition applicable C₁ C₂ L₁ L₂ σ ⟷
        C₁ ≠ {} ∧ C₂ ≠ {} ∧ L₁ ≠ {} ∧ L₂ ≠ {}
      ∧ vars C₁ ∩ vars C₂ = {}
      ∧ L₁ ⊆ C₁ ∧ L₂ ⊆ C₂
      ∧ mgu σ (L₁ ∪ L₂ ᶜ)"

definition resolution C₁ C₂ L₁ L₂ σ = ((C₁ - L₁) ∪ (C₂ - L₂)) · σ

inductive resolution_step
  :: fterm clause set ⇒ fterm clause set ⇒ bool where
  resolution_rule:
    C₁ ∈ Cs ⟹ C₂ ∈ Cs ⟹ applicable C₁ C₂ L₁ L₂ σ ⟹
       resolution_step Cs (Cs ∪ {resolution C₁ C₂ L₁ L₂ σ})
| standardize_apart:
    C ∈ Cs ⟹ var_renaming_of C C' ⟹ resolution_step Cs (Cs ∪ {C'})

definition resolution_deriv = rtranclp resolution_step
```

Refutational completeness

Refutational completeness

Refutational completeness:

If C is unsatisfiable then the calculus can derive a contradiction

Refutational completeness

Refutational completeness:

If C is unsatisfiable then the calculus can derive a contradiction

```
unsatisfiable C ⟹ (C ⊢ {})
```

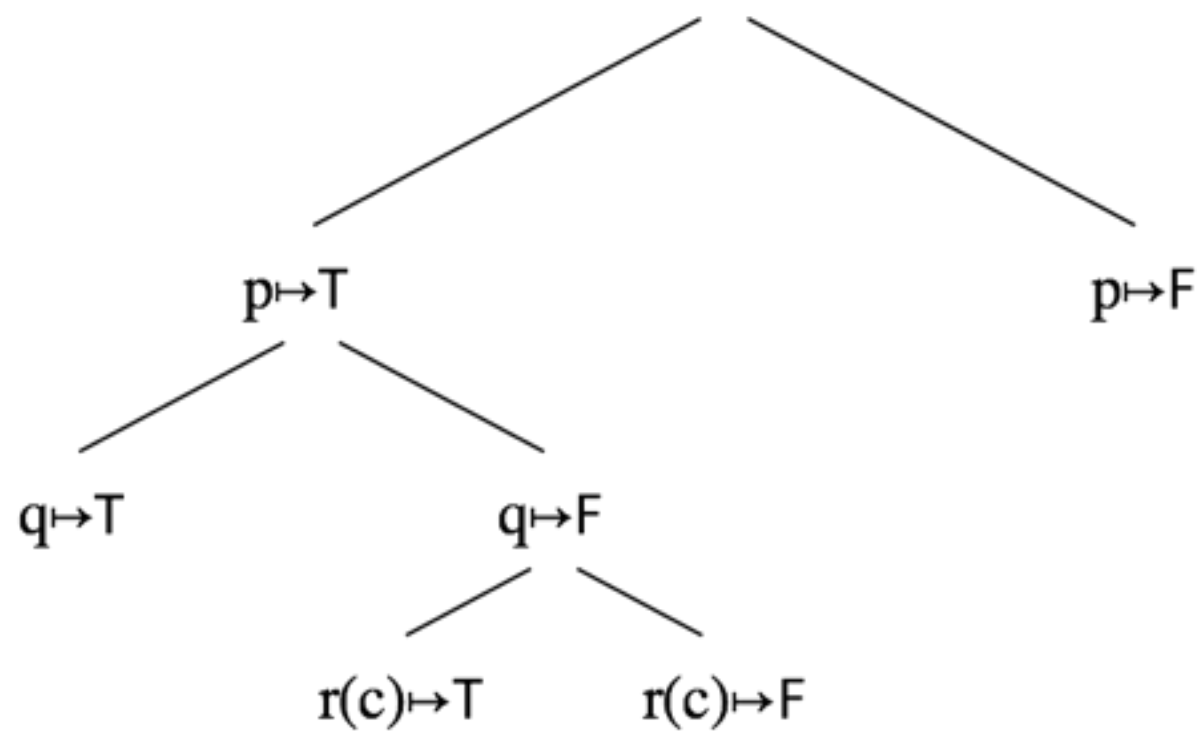
Semantic tree

Semantic tree

Enumeration of ground terms: $p, q, r(c), \ldots$

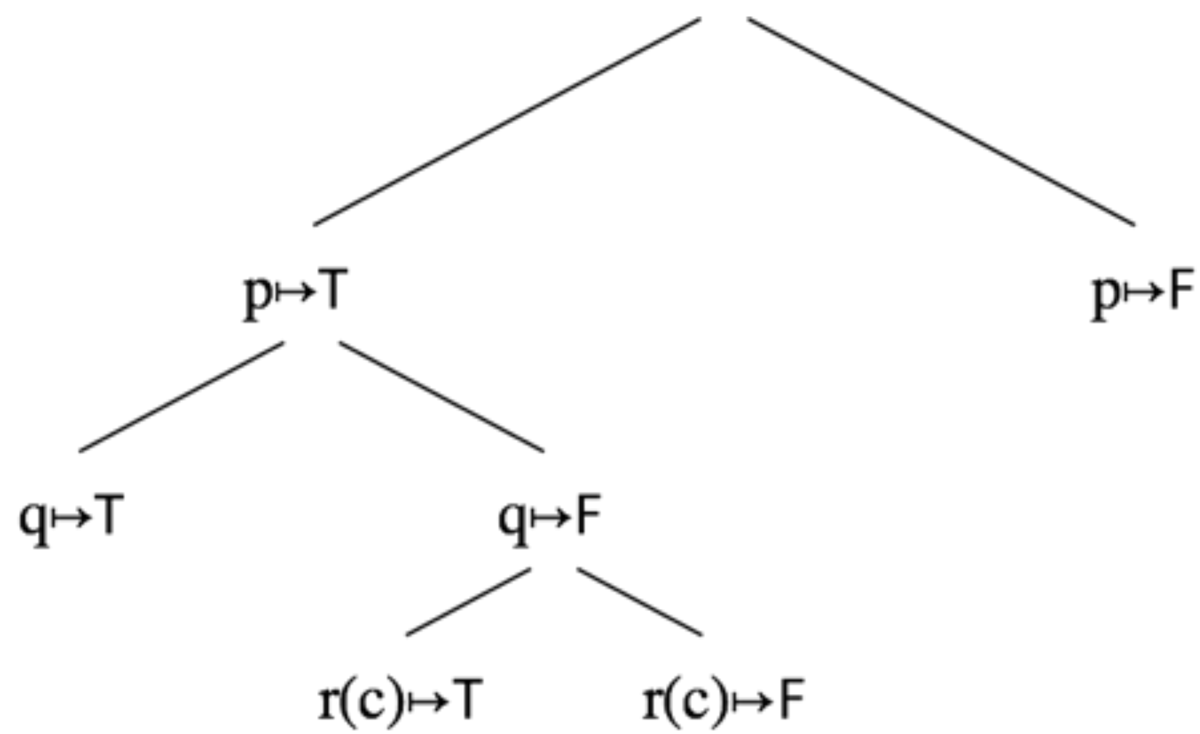
Semantic tree

Enumeration of ground terms: $p, q, r(c), \ldots$



Semantic tree

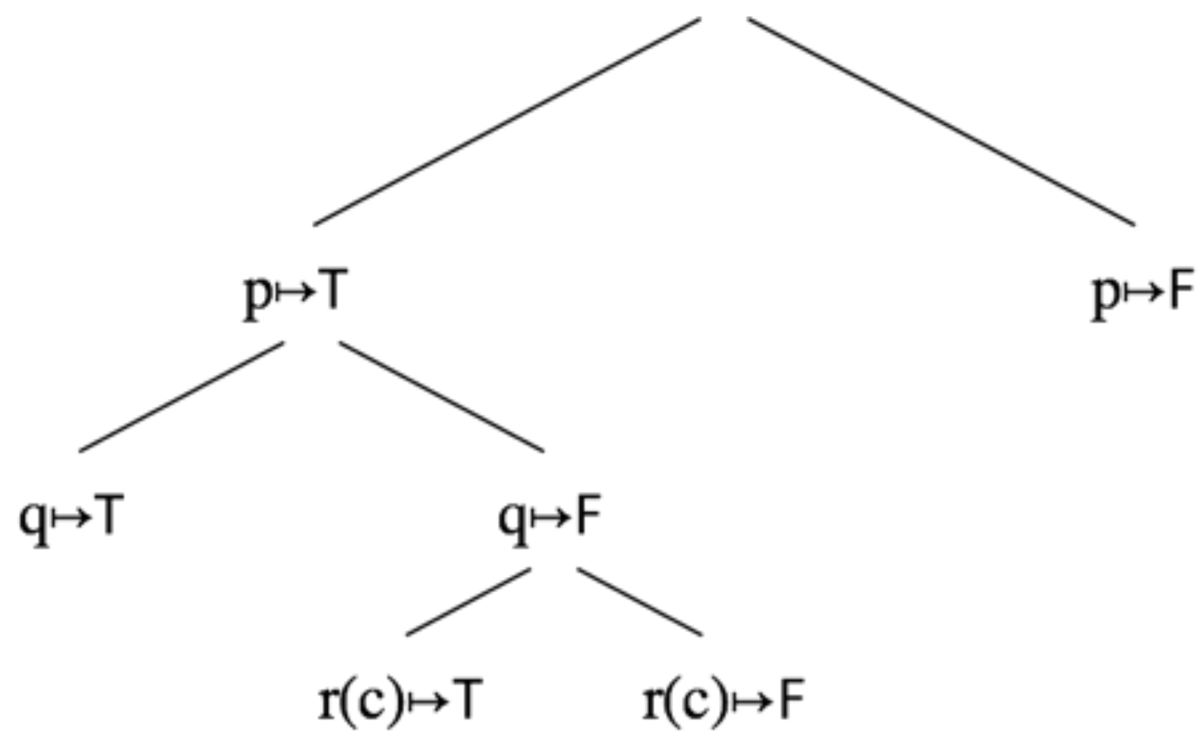
Enumeration of ground terms: $p, q, r(c), \ldots$



Semantic trees are decision trees assigning `True` and `False` to the ground atoms.

Semantic tree

Enumeration of ground terms: $p, q, r(c), \ldots$

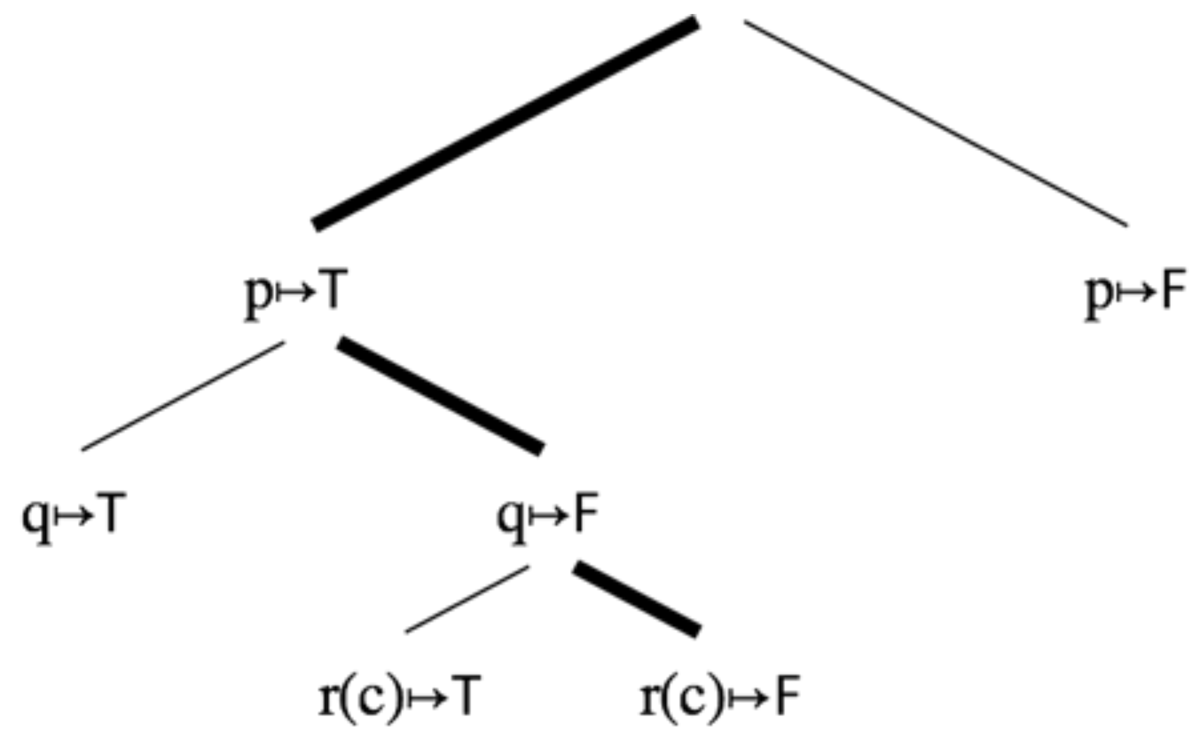


Semantic trees are decision trees assigning `True` and `False` to the ground atoms.

Node on depth `i` makes decision for atom `i`.

Semantic tree

A path represents a partial (Herbrand) interpretation.



E.g. $\{p \mapsto T, q \mapsto F, r(c) \mapsto F\}$

Formalized enumeration

Formalized enumeration

```
definition nat_from_hatom :: hterm atom ⇒ nat where
  nat_from_hatom ≡ (SOME f. bij f)
```

Formalized enumeration

```
definition nat_from_hatom :: hterm atom ⇒ nat where
  nat_from_hatom ≡ (SOME f. bij f)

instantiation hterm :: countable begin
instance by countable_datatype
end
```

Formalized enumeration

```
definition nat_from_hatom :: hterm atom ⇒ nat where
  nat_from_hatom ≡ (SOME f. bij f)

instantiation hterm :: countable begin
instance by countable_datatype
end

lemma infinite_hatoms: infinite (UNIV :: 't atom set)
<proof>
```

Formalized enumeration

```
definition nat_from_hatom :: hterm atom ⇒ nat where
  nat_from_hatom ≡ (SOME f. bij f)

instantiation hterm :: countable begin
instance by countable_datatype
end

lemma infinite_hatoms: infinite (UNIV :: 't atom set)
<proof>

lemma nat_from_hatom_bij: bij nat_from_hatom
proof -
  have countable (UNIV :: hterm atom set) by simp
  moreover
  have infinite (UNIV :: hterm atom set) using infinite_hatoms by auto
  ultimately
  obtain x where bij (x :: hterm atom ⇒ nat) using countableE_infinite by blast
  then show ?thesis using ... someI by metis
qed
```

Formalized enumeration

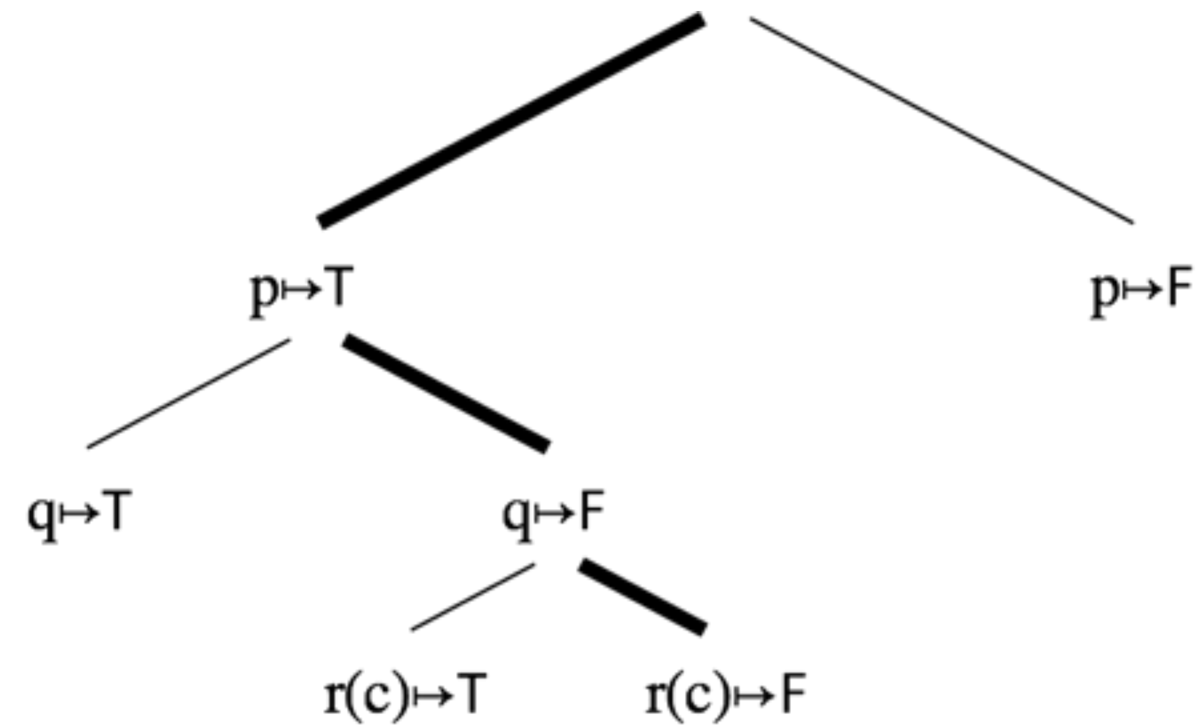
```
definition nat_from_hatom :: hterm atom ⇒ nat where
  nat_from_hatom ≡ (SOME f. bij f)

instantiation hterm :: countable begin
instance by countable_datatype
end

lemma infinite_hatoms: infinite (UNIV :: 't atom set)
<proof>

lemma nat_from_hatom_bij: bij nat_from_hatom
proof -
  have countable (UNIV :: hterm atom set) by simp
  moreover
  have infinite (UNIV :: hterm atom set) using infinite_hatoms by auto
  ultimately
  obtain x where bij (x :: hterm atom ⇒ nat) using countableE_infinite by blast
  then show ?thesis using ... someI by metis
qed
```

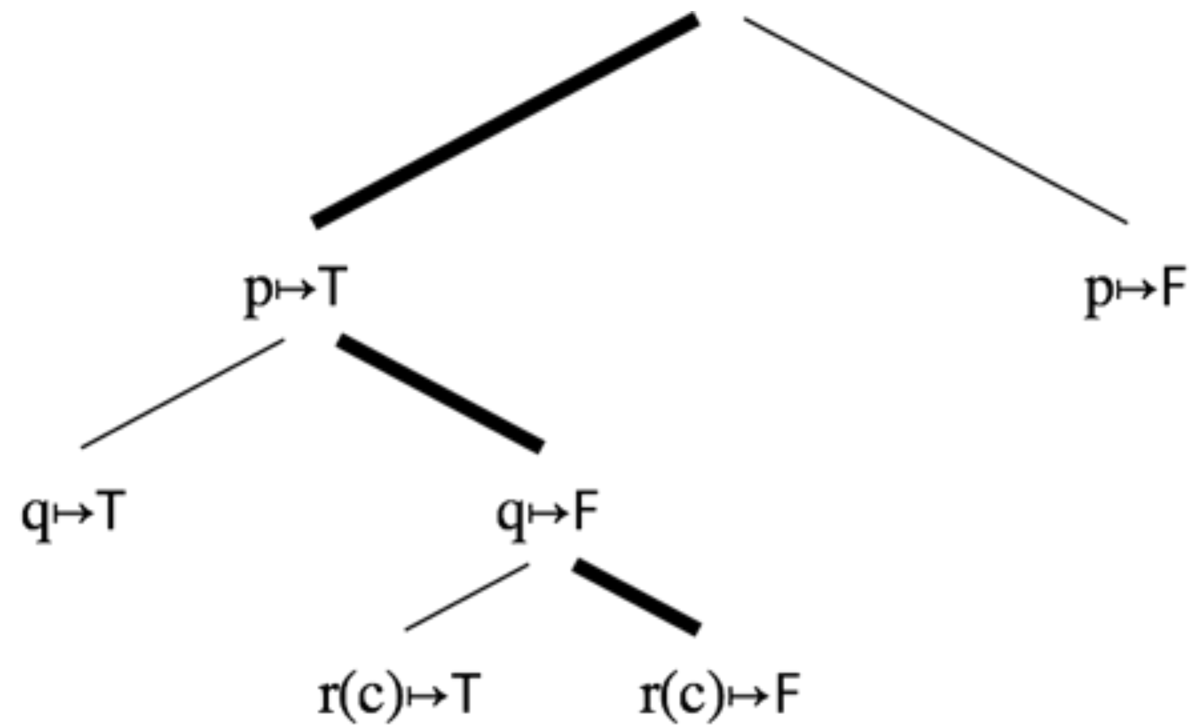
Formalized semantic trees



Formalized semantic trees

Finite trees:

```
datatype tree =
  Leaf
| Branching tree tree
```



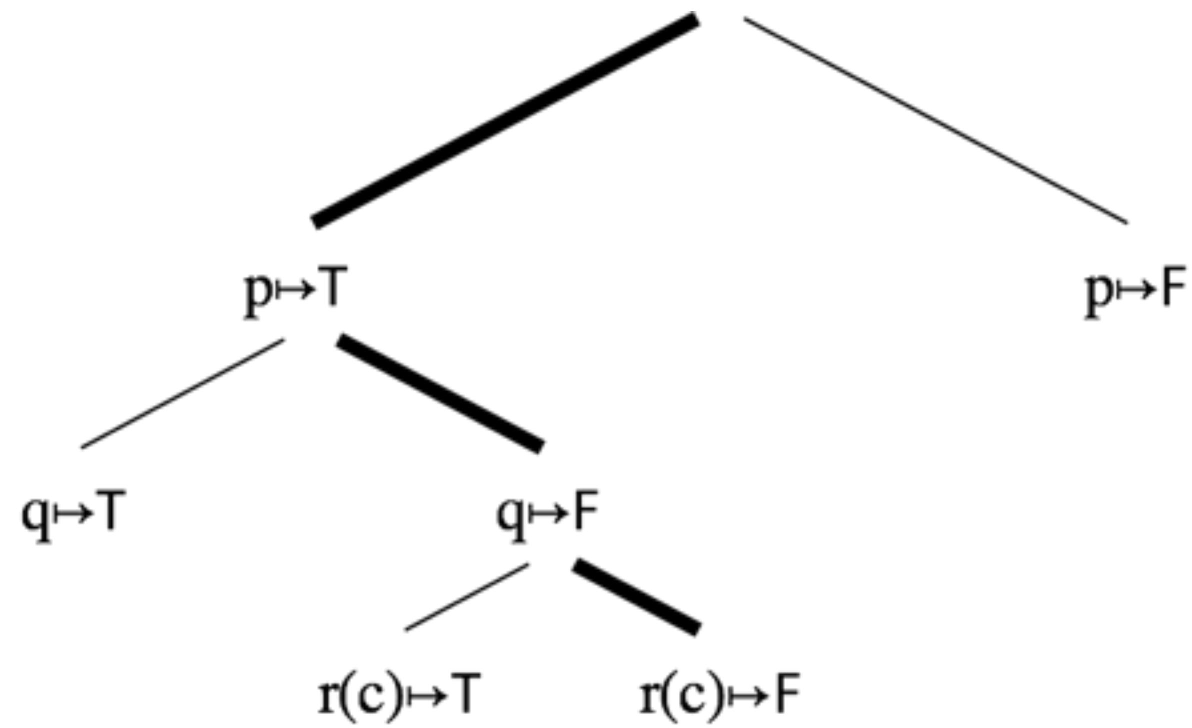
Formalized semantic trees

Finite trees:

```
datatype tree =
  Leaf
| Branching tree tree
```

Paths:

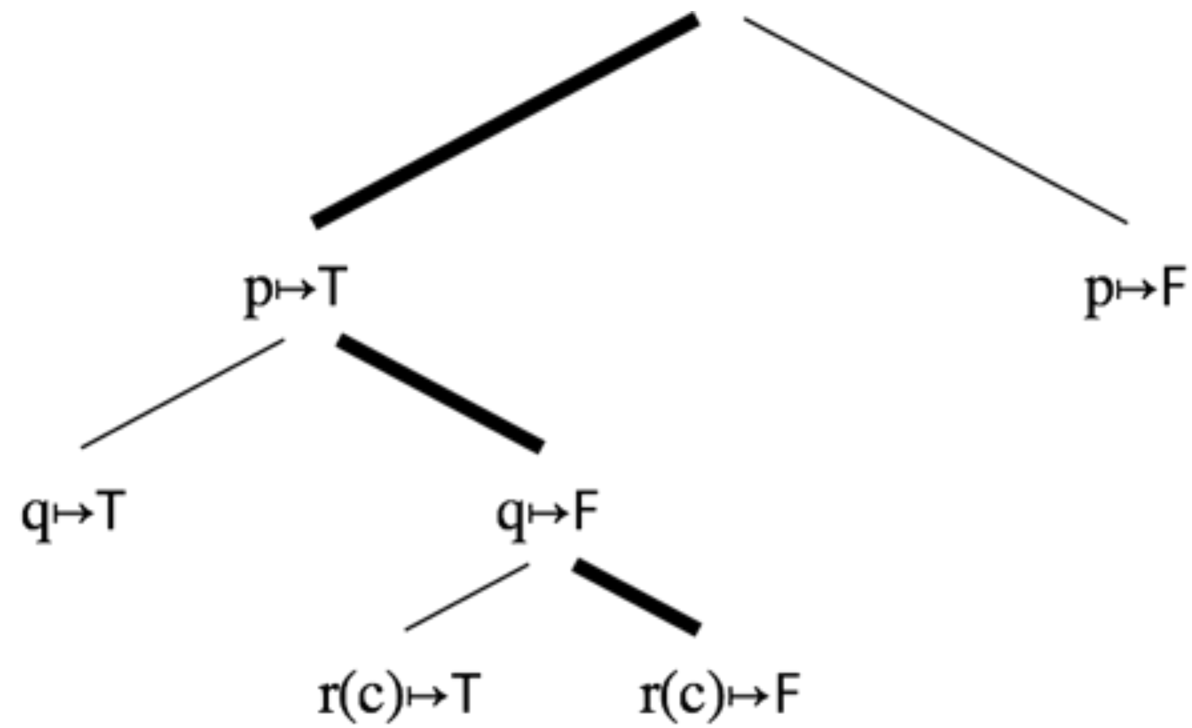
```
type_synonym path = bool list
```



Formalized semantic trees

Finite trees:

```
datatype tree =
  Leaf
| Branching tree tree
```



Paths:

```
type_synonym path = bool list
```

Possibly infinite trees:

```
type_synonym inftree = path set

abbreviation wf_tree :: path set ⇒ bool where
  wf_tree T ≡ (∀ds d. (ds @ d) ∈ T ⟶ ds ∈ T)
```

Falsification by partial interpretation

Falsification by partial interpretation

Falsification of ground clause:

$\{p \mapsto T, q \mapsto F, r(c) \mapsto T\}$ falsifies $\{q, \neg r(c)\}$

Falsification by partial interpretation

Falsification of ground clause:

$\{p \mapsto T, q \mapsto F, r(c) \mapsto T\}$ falsifies $\{q, \neg r(c)\}$

```
abbreviation falsifiesg :: path ⇒ fterm clause ⇒ bool where
   falsifiesg G C ≡ ground C ∧ (∀l ∈ C. falsifies G l)
```

Falsification by partial interpretation

Falsification of ground clause:

$\{p \mapsto T, q \mapsto F, r(c) \mapsto T\}$ falsifies $\{q, \neg r(c)\}$

```
abbreviation falsifies_g :: path ⇒ fterm clause ⇒ bool where
  falsifies_g G C ≡ ground C ∧ (∀l ∈ C. falsifies G l)
```

Falsification of FO clause:

$\{p \mapsto T, q \mapsto F, r(c) \mapsto T\}$ falsifies $\{q, \neg r(x)\}$

Falsification by partial interpretation

Falsification of ground clause:

$\{p \mapsto T, q \mapsto F, r(c) \mapsto T\}$ falsifies $\{q, \neg r(c)\}$

```
abbreviation falsifies_g :: path ⇒ fterm clause ⇒ bool where
   falsifies_g G C ≡ ground C ∧ (∀l ∈ C. falsifies G l)
```

Falsification of FO clause:

$\{p \mapsto T, q \mapsto F, r(c) \mapsto T\}$ falsifies $\{q, \neg r(x)\}$

```
abbreviation falsifies :: path ⇒ fterm clause ⇒ bool where
  falsifies G C ≡ (∃C'. instance_of C' C ∧ falsifies_g G C')
```

Closed semantic tree

Definition of closed semantic tree:

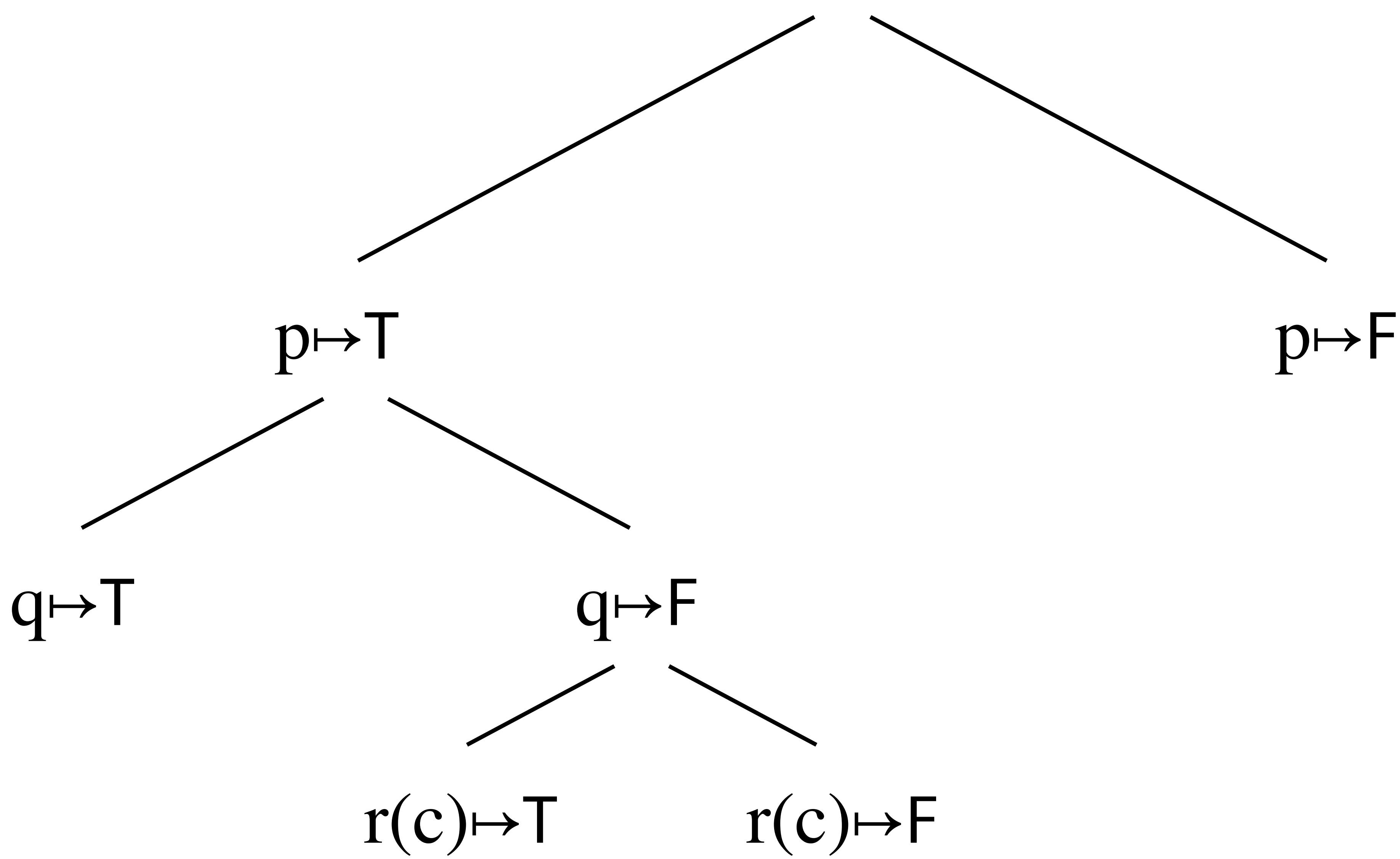
All branches falsify a ground instance of a clause in Cs

Closed semantic tree

Definition of closed semantic tree:

All branches falsify a ground instance of a clause in Cs

$Cs = \{\ \{\neg q, \neg p\}, \{r(x)\}, \{\neg p, q, \neg r(y)\}, \{p\}\}$

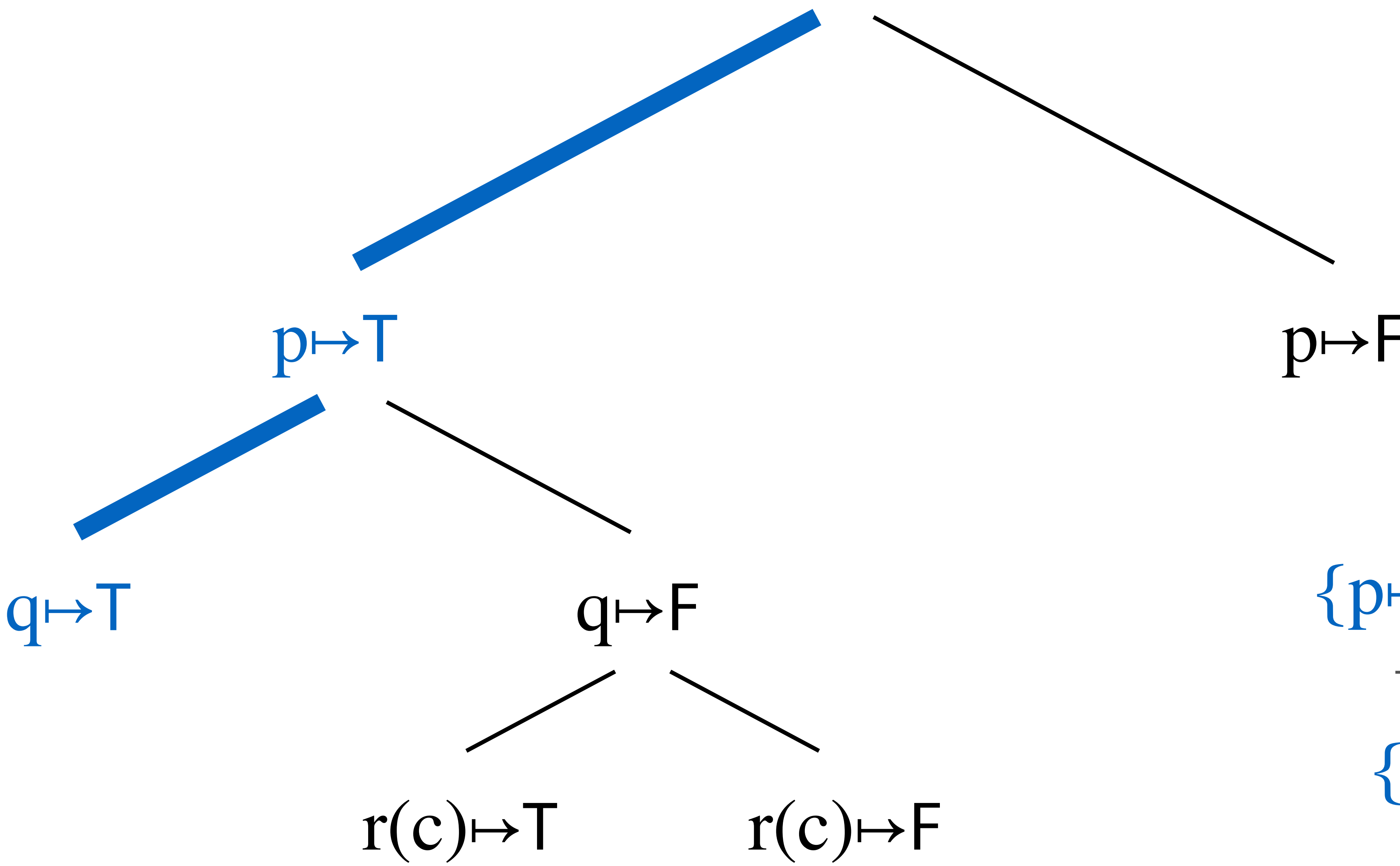


Closed semantic tree

Definition of closed semantic tree:

All branches falsify a ground instance of a clause in Cs

$Cs = \{ \{\neg q, \neg p\}, \{r(x)\}, \{\neg p, q, \neg r(y)\}, \{p\}\}$



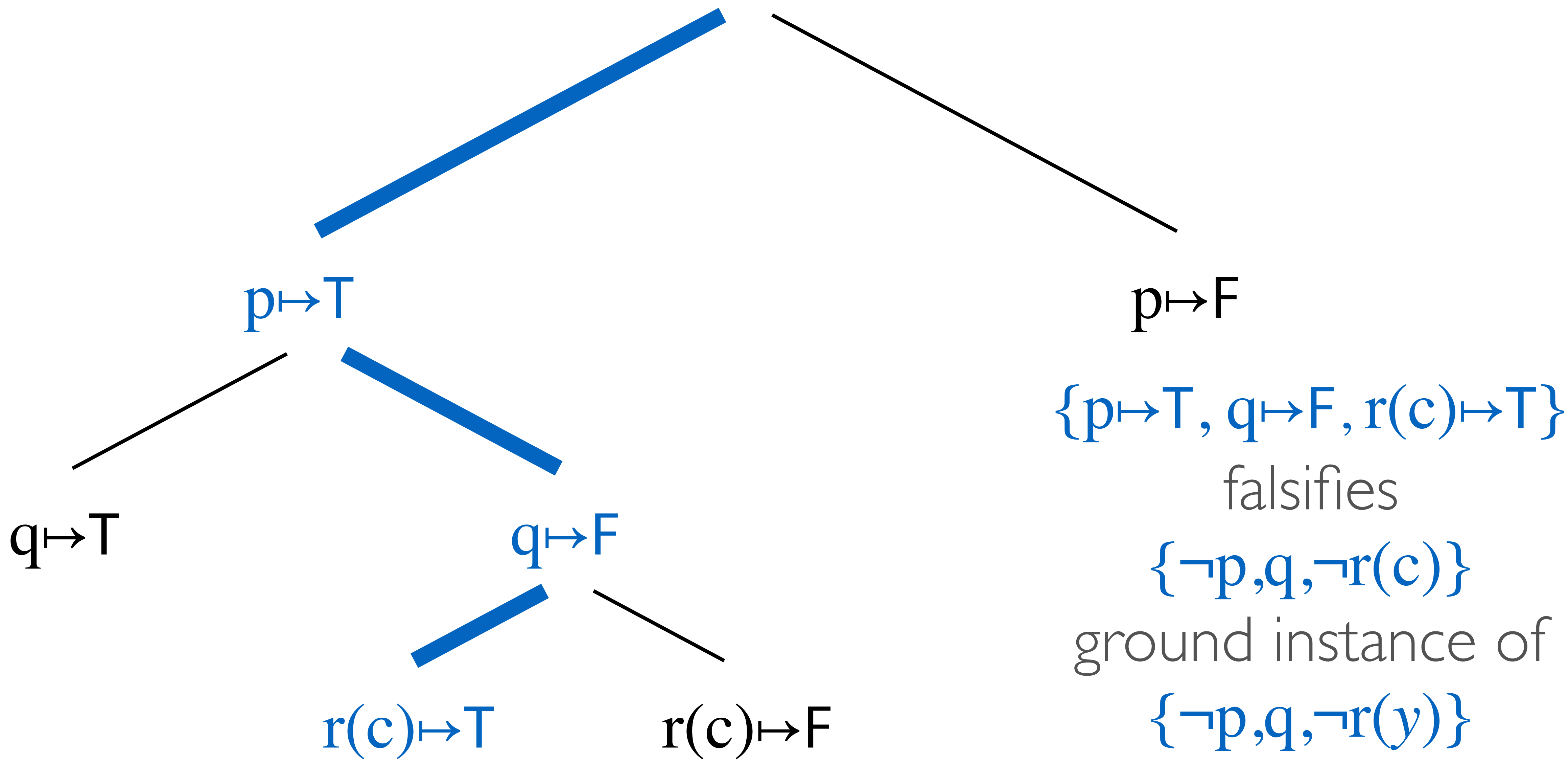
$\{p \mapsto T, q \mapsto T\}$
falsifies
$\{\neg q, \neg p\}$

Closed semantic tree

Definition of closed semantic tree:

All branches falsify a ground instance of a clause in Cs

$Cs = \{\ \{\neg q,\neg p\}, \{r(x)\}, \{\neg p, q, \neg r(y)\}, \{p\}\}$

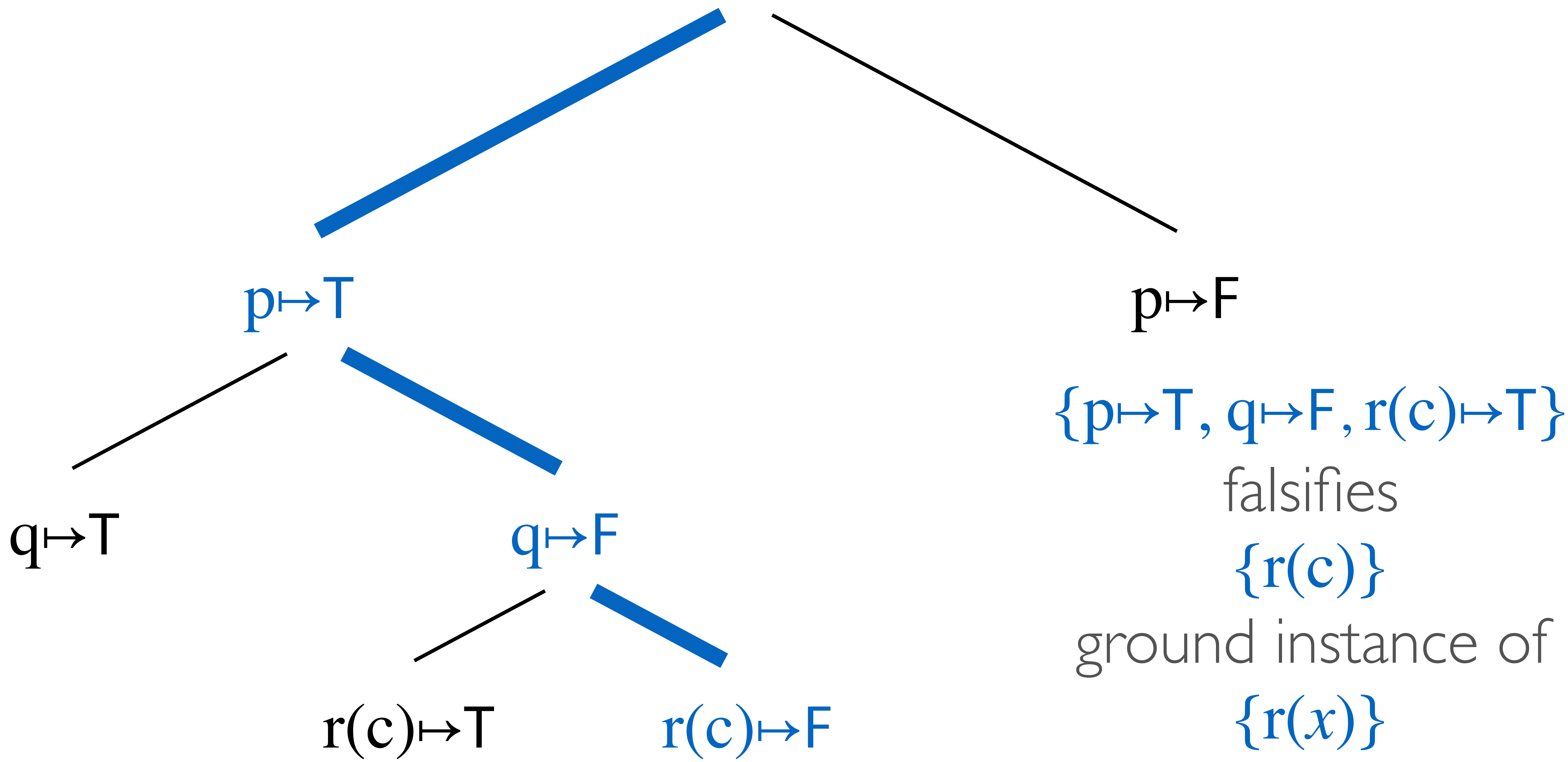


Closed semantic tree

Definition of closed semantic tree:

All branches falsify a ground instance of a clause in Cs

$Cs = \{\ \{\neg q, \neg p\}, \{r(x)\}, \{\neg p, q, \neg r(y)\}, \{p\}\}$

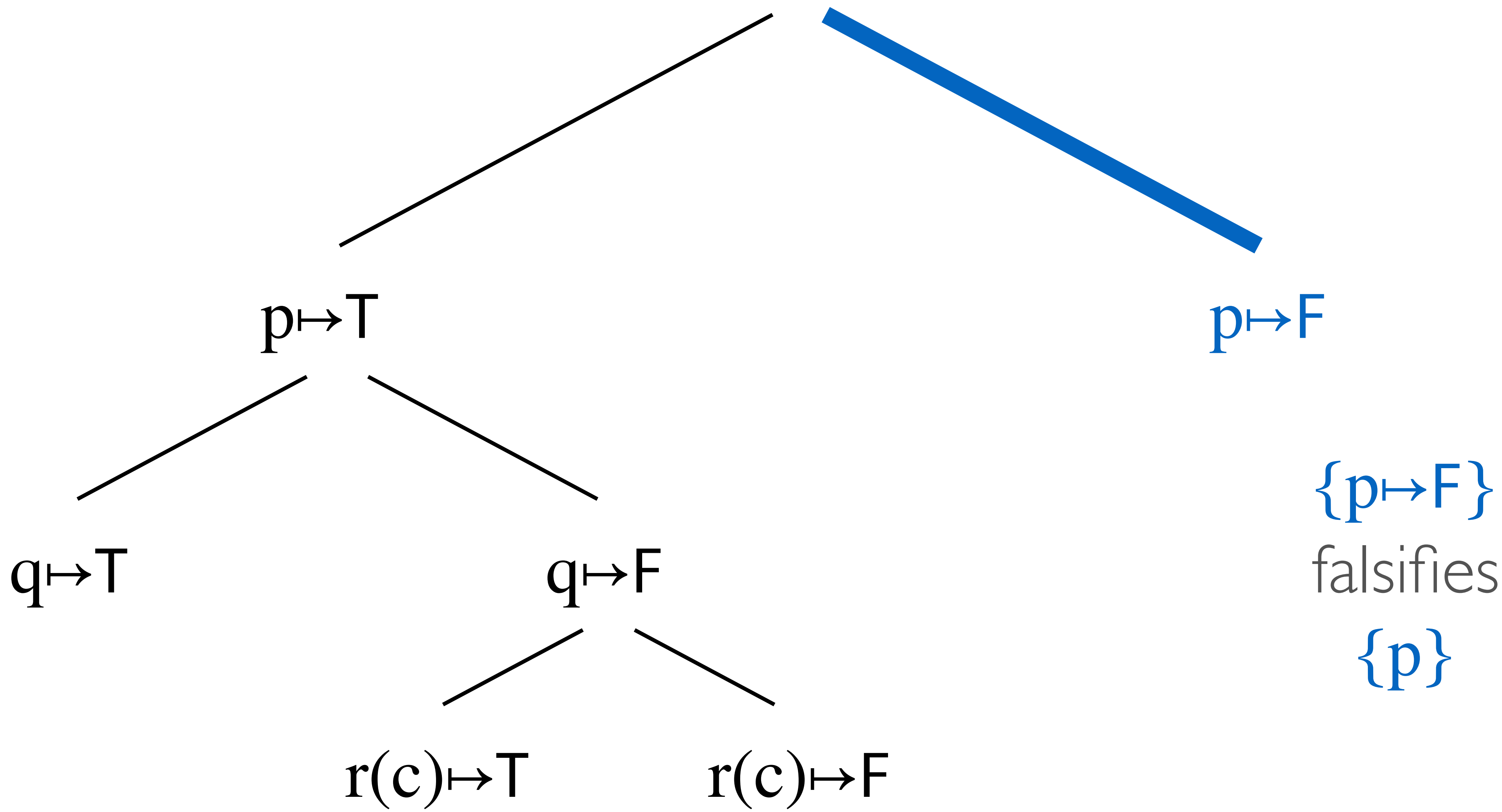


Closed semantic tree

Definition of closed semantic tree:

All branches falsify a ground instance of a clause in Cs

$Cs = \{ \{\neg q, \neg p\}, \{r(x)\}, \{\neg p, q, \neg r(y)\}, \{p\} \}$



Completeness proof

1. Herbrand's theorem:
   Any unsatisfiable set of clauses has a finite closed semantic tree.

2. {} is derivable from any set of clauses with a closed semantic tree.

The proof follows Chang & Lee (1973).

Completeness proof

↳ 1. Herbrand's theorem

2. Deriving {}

Herbrand's theorem:

Any unsatisfiable set of clauses $\boldsymbol{Cs}$ *has a finite closed semantic tree.*

Proof:

Let $\boldsymbol{T}$ *be a full infinite semantic tree.*
Consider any infinite $\boldsymbol{p}$ *path in* $\boldsymbol{T}$*.*
$\boldsymbol{p}$ *is an interpretation and thus falsifies* $\boldsymbol{Cs}$*.*
A (finite) prefix also falsifies $\boldsymbol{Cs}$*.*
Let $\boldsymbol{T'}$ *be a copy of* $\boldsymbol{T}$ *with all paths replaced with finite falsifying prefixes.*
$\boldsymbol{T'}$ *is finite by König's lemma.*

Completeness proof

↳ 1. Herbrand's theorem

2. Deriving {}

Herbrand's theorem:

Any unsatisfiable set of clauses $\boldsymbol{Cs}$ *has a finite closed semantic tree.*

Proof:

Let $\boldsymbol{T}$ *be a full infinite semantic tree.*

Consider any infinite $\boldsymbol{p}$ *path in* $\boldsymbol{T}$*.*

$\boldsymbol{p}$ *is an interpretation and thus falsifies* $\boldsymbol{Cs}$*.*

A (finite) prefix also falsifies $\boldsymbol{Cs}$*.*

Let $\boldsymbol{T'}$ *be a copy of* $\boldsymbol{T}$ *with all paths replaced with finite falsifying prefixes.*

$\boldsymbol{T'}$ *is finite by König's lemma.*

p is an interpretation?
A path is a list of bools.
An interpretation is a
`fun_sym ⇒ 'u list ⇒ 'u`
and a
`pred_sym ⇒ 'u list ⇒ bool`

Completeness proof

↳ 1. Herbrand's theorem

2. Deriving {}

Herbrand's theorem:

Any unsatisfiable set of clauses $\boldsymbol{Cs}$ *has a finite closed semantic tree.*

Proof:

Let $\boldsymbol{T}$ *be a full infinite semantic tree.*
Consider any infinite $\boldsymbol{p}$ *path in* $\boldsymbol{T}$*.*
$\boldsymbol{p}$ *is an interpretation and thus falsifies* $\boldsymbol{Cs}$*.*
A (finite) prefix also falsifies $\boldsymbol{Cs}$*.*
Let $\boldsymbol{T'}$ *be a copy of* $\boldsymbol{T}$ *with all paths replaced with finite falsifying prefixes.*
$\boldsymbol{T'}$ *is finite by König's lemma.*

p is an interpretation?
A path is a list of bools.
An interpretation is a
`fun_sym ⇒ 'u list ⇒ 'u`
and a
`pred_sym ⇒ 'u list ⇒ bool`

Yes, we can make a conversion function `extend`.

Completeness proof

↳ 1. Herbrand's theorem

2. Deriving {}

Herbrand's theorem:

Any unsatisfiable set of clauses ***Cs*** *has a finite closed semantic tree.*

Proof:

Let ***T*** *be a full infinite semantic tree.*
Consider any infinite ***p*** *path in* ***T****.*
p *is an interpretation and thus falsifies* ***Cs****.*
A (finite) prefix also falsifies ***Cs****.* Does it?
Let ***T'*** *be a copy of* ***T*** *with all paths replaced with finite falsifying prefixes.*
T' *is finite by König's lemma.*

Completeness proof

↳ 1. Herbrand's theorem

2. Deriving {}

If an infinite path falsifies a set of clauses, then so does a finite prefix.

	FO clause set
Interpretation	Cs falsified by `extend p` $\Downarrow$
Partial interpretation	Cs falsified by prefix of `p`

Completeness proof

↳ 1. Herbrand's theorem

2. Deriving {}

If an infinite path falsifies a set of clauses, then so does a finite prefix.

	FO clause set	Ground clause set
Interpretation	Cs falsified by `extend p` ⇓	
Partial interpretation	Cs falsified by prefix of `p`	

Completeness proof

↳ 1. Herbrand's theorem

2. Deriving {}

If an infinite path falsifies a set of clauses, then so does a finite prefix.

	FO clause set	Ground clause set
Interpretation	Cs falsified by `extend p` $\Downarrow$	Cs' falsified by `extend p`
Partial interpretation	Cs falsified by prefix of `p`	

Completeness proof

↳ 1. Herbrand's theorem

2. Deriving {}

If an infinite path falsifies a set of clauses, then so does a finite prefix.

	FO clause set	Ground clause set
Interpretation	Cs falsified by `extend p` $\Longrightarrow$	Cs' falsified by `extend p`
Partial interpretation	$\Downarrow$ Cs falsified by prefix of `p`	

Completeness proof

↳ 1. Herbrand's theorem

2. Deriving {}

If an infinite path falsifies a set of clauses, then so does a finite prefix.

	FO clause set	Ground clause set
Interpretation	Cs falsified by `extend p` $\Rightarrow$ $\Downarrow$	Cs' falsified by `extend p`
Partial interpretation	Cs falsified by prefix of `p`	Cs' falsified by prefix of `p`

Completeness proof

↳ 1. Herbrand's theorem

2. Deriving {}

If an infinite path falsifies a set of clauses, then so does a finite prefix.

	FO clause set		Ground clause set
Interpretation	Cs falsified by `extend p` ⇓	$\Longrightarrow$	Cs' falsified by `extend p` ⇓
Partial interpretation	Cs falsified by prefix of `p`		Cs' falsified by prefix of `p`

Completeness proof

↳ 1. Herbrand's theorem

2. Deriving {}

If an infinite path falsifies a set of clauses, then so does a finite prefix.

	FO clause set		Ground clause set
Interpretation	Cs falsified by `extend p` $\Downarrow$	$\Longrightarrow$	Cs' falsified by `extend p` $\Downarrow$
Partial interpretation	Cs falsified by prefix of `p`	$\Longleftarrow$	Cs' falsified by prefix of `p`

Completeness proof

↳ 1. Herbrand's theorem

2. Deriving {}

Completeness proof

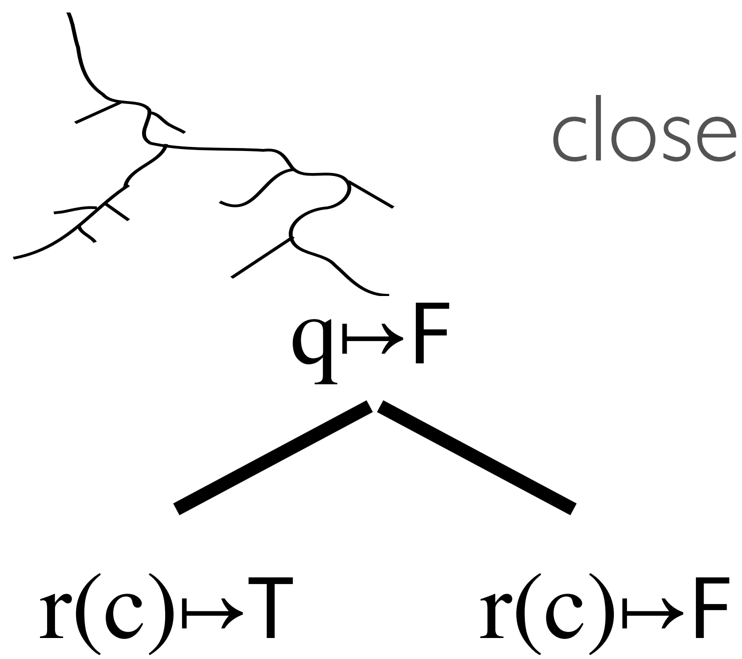
1. Herbrand's theorem

↳ 2. Deriving {}

Completeness proof

1. Herbrand's theorem

↳2. Deriving {}



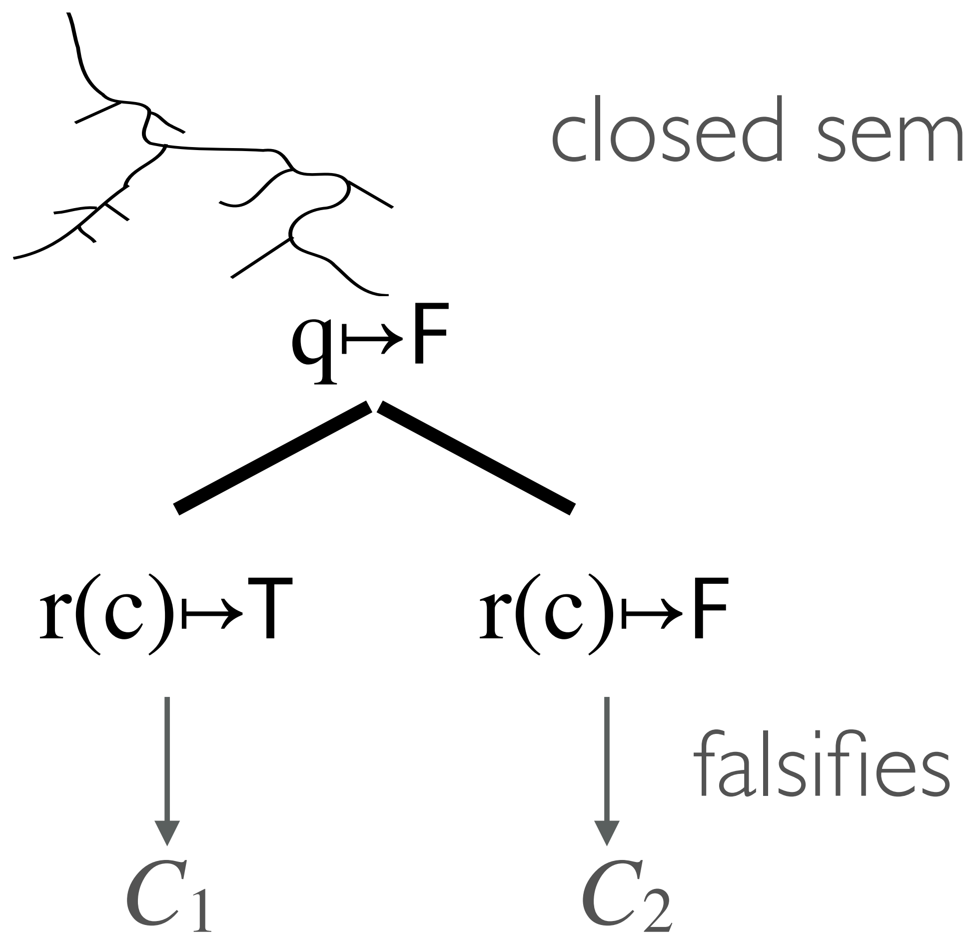
closed semantic tree for *Cs*

Completeness proof

1. Herbrand's theorem

↳ 2. Deriving {}

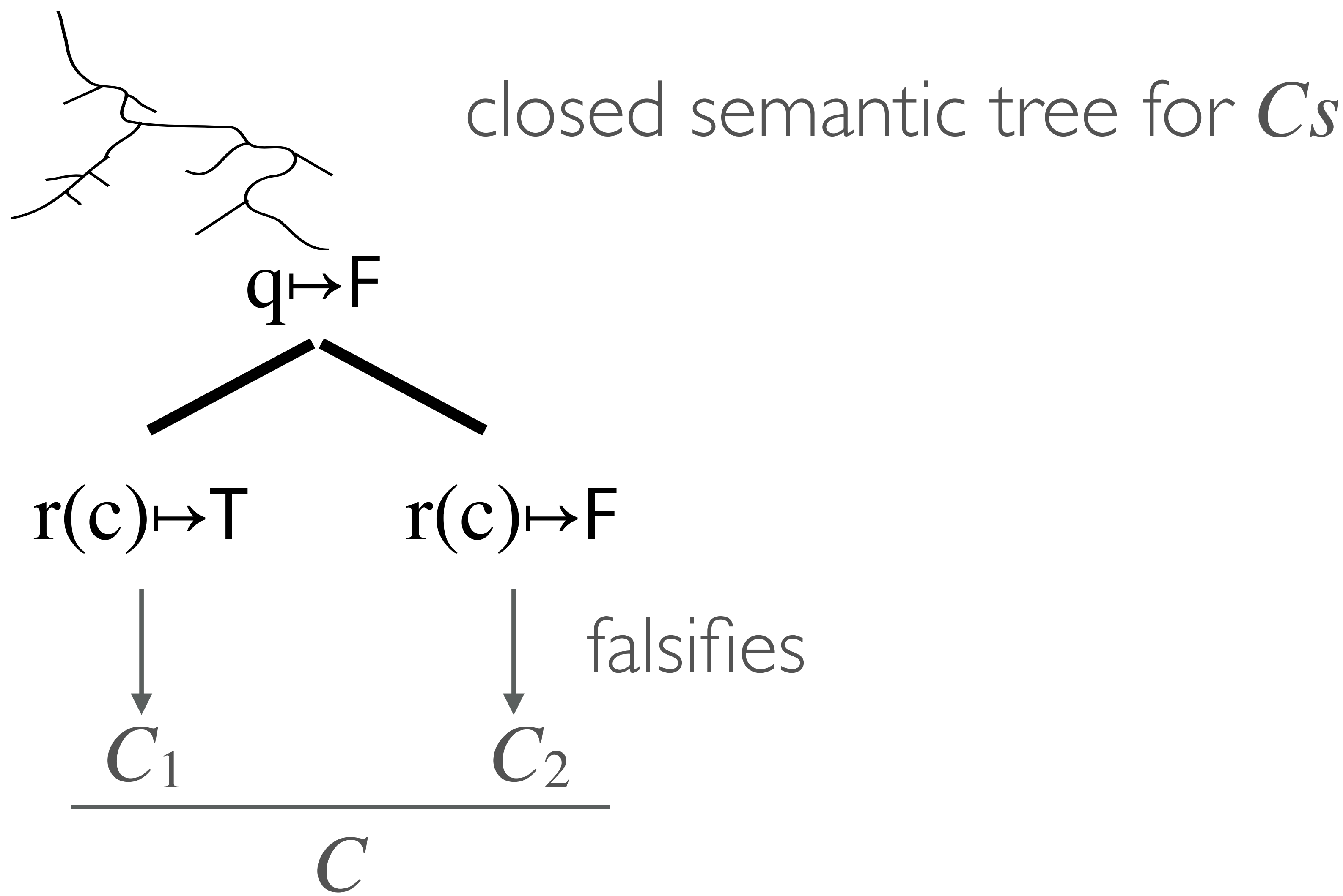
closed semantic tree for Cs



Completeness proof

1. Herbrand's theorem

↳ 2. Deriving {}

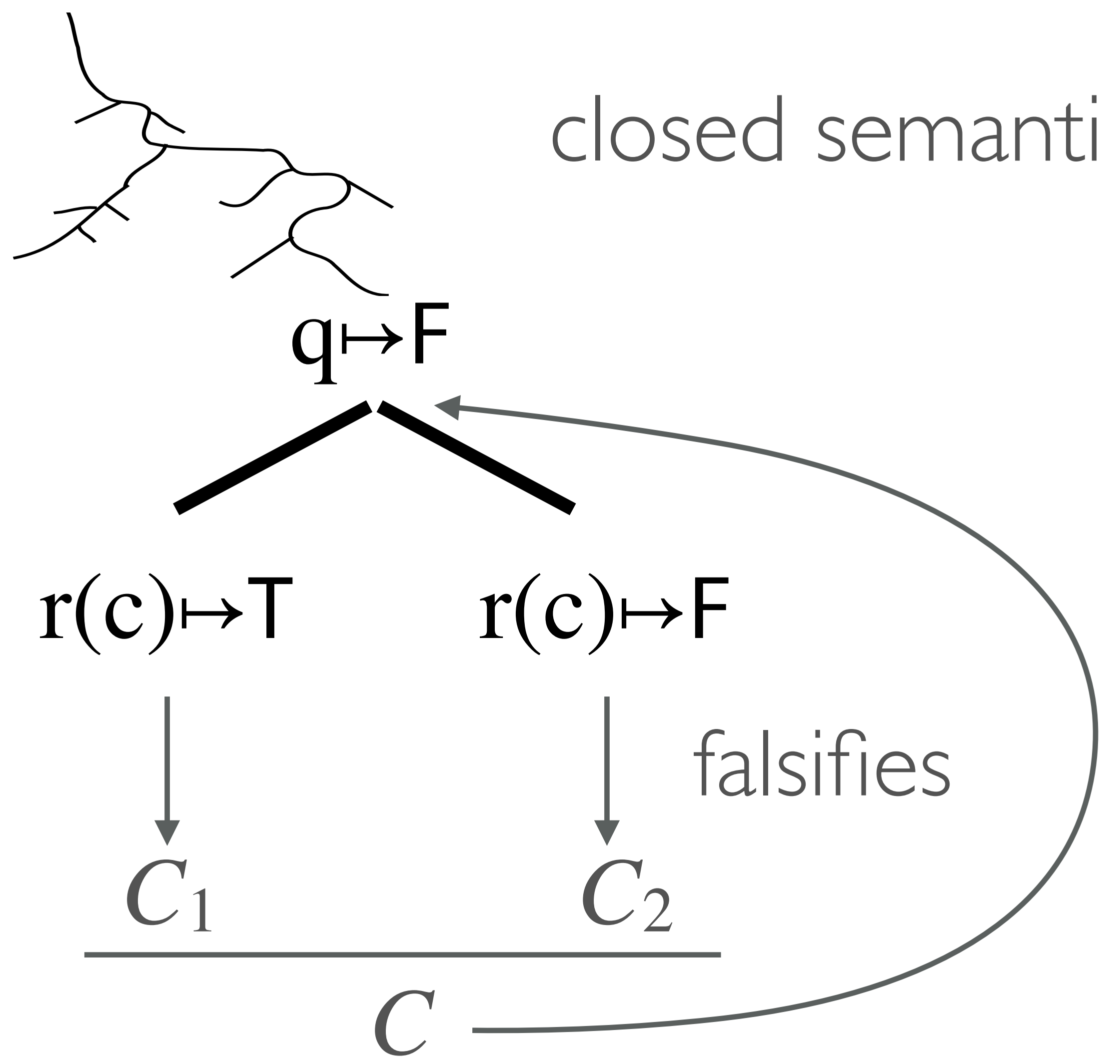


Completeness proof

1. Herbrand's theorem

↳ 2. Deriving {}

closed semantic tree for *Cs*

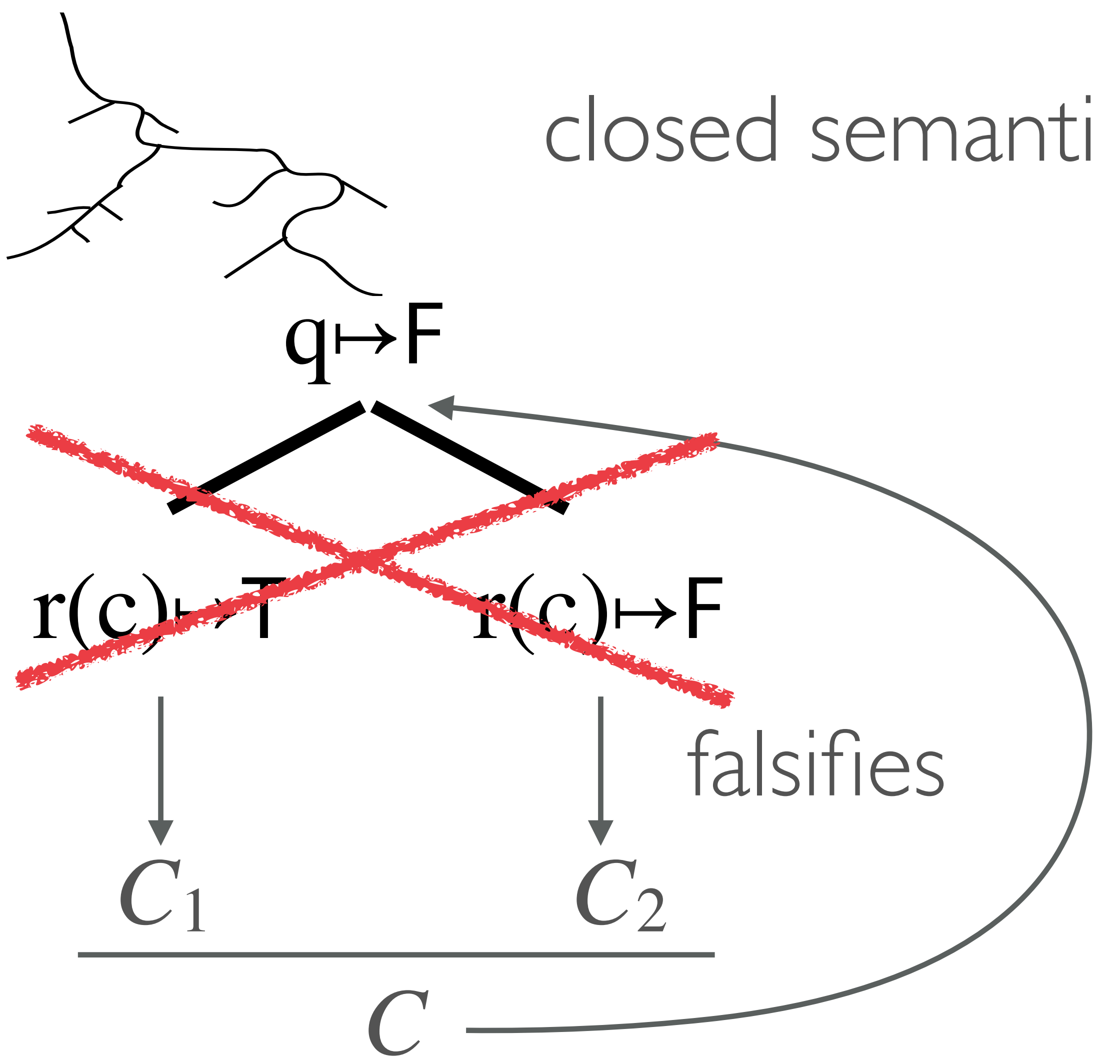


Completeness proof

1. Herbrand's theorem

↳ 2. Deriving {}

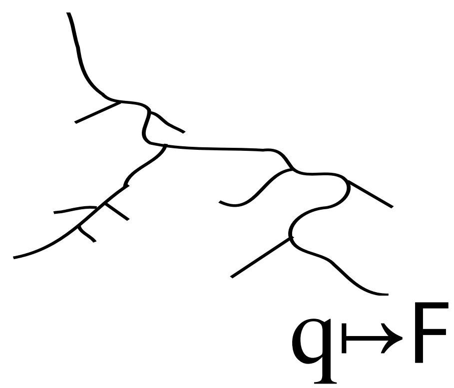
closed semantic tree for $Cs \cup \{C\}$



Completeness proof

1. Herbrand's theorem

↳ 2. Deriving {}

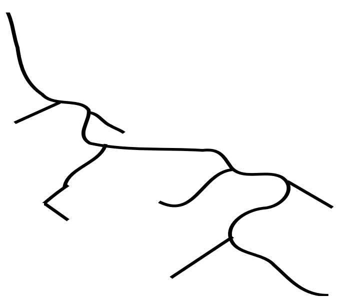


closed semantic tree for $Cs \cup \{C\}$

Completeness proof

1. Herbrand's theorem

↳2. Deriving {}



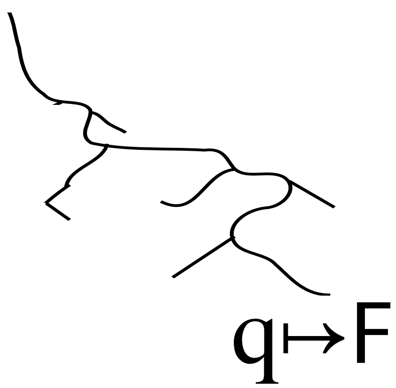
$q \mapsto F$

closed semantic tree for $Cs \cup \{C\}$

Completeness proof

1. Herbrand's theorem

↳2. Deriving {}



closed semantic tree for $Cs \cup \{C\}$

Completeness proof

1. Herbrand's theorem

↳ 2. Deriving {}

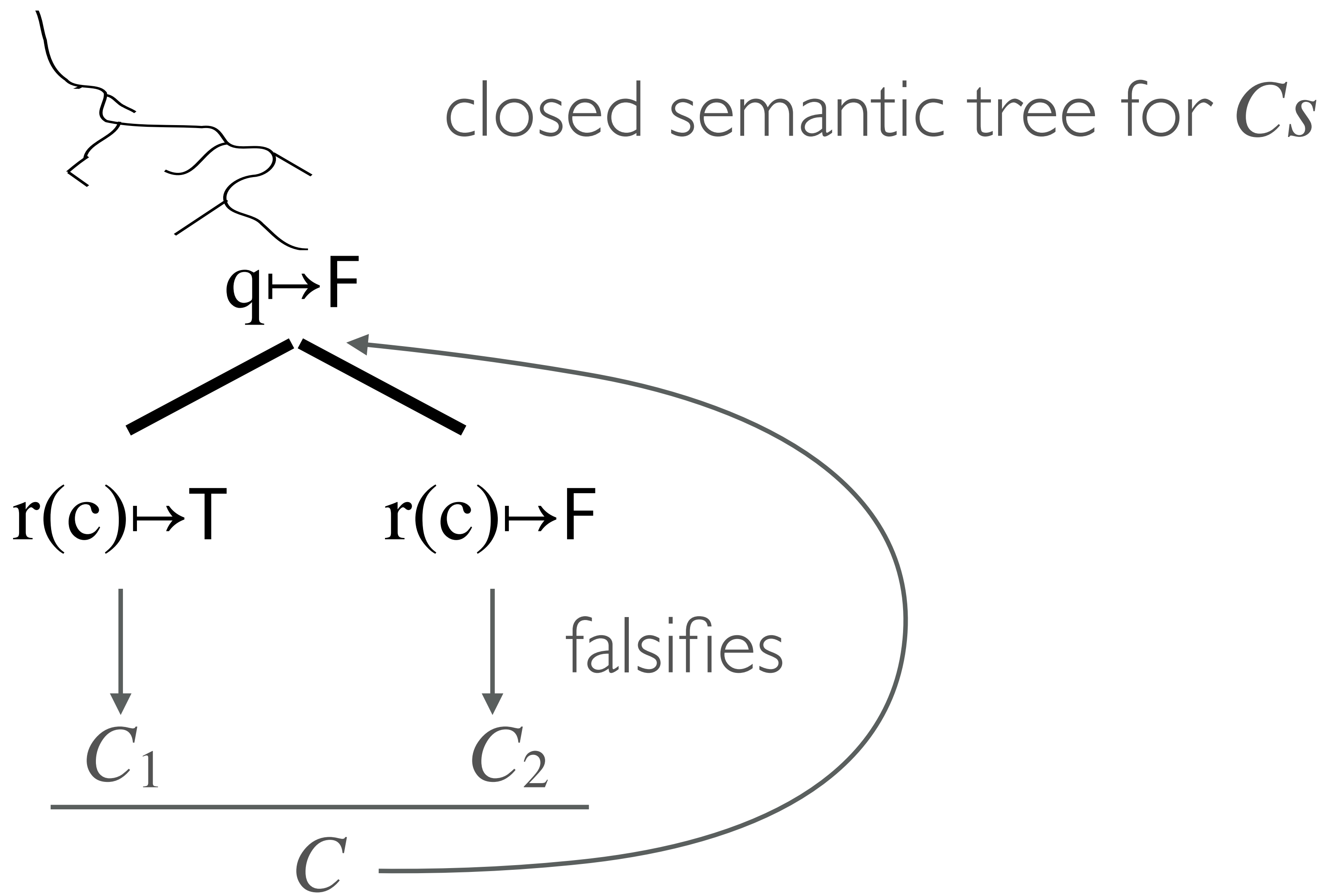
Eventually the empty tree is closed for our Cs.

Then we have derived $\{\}$.

Completeness proof

1. Herbrand's theorem

↳ 2. Deriving {}

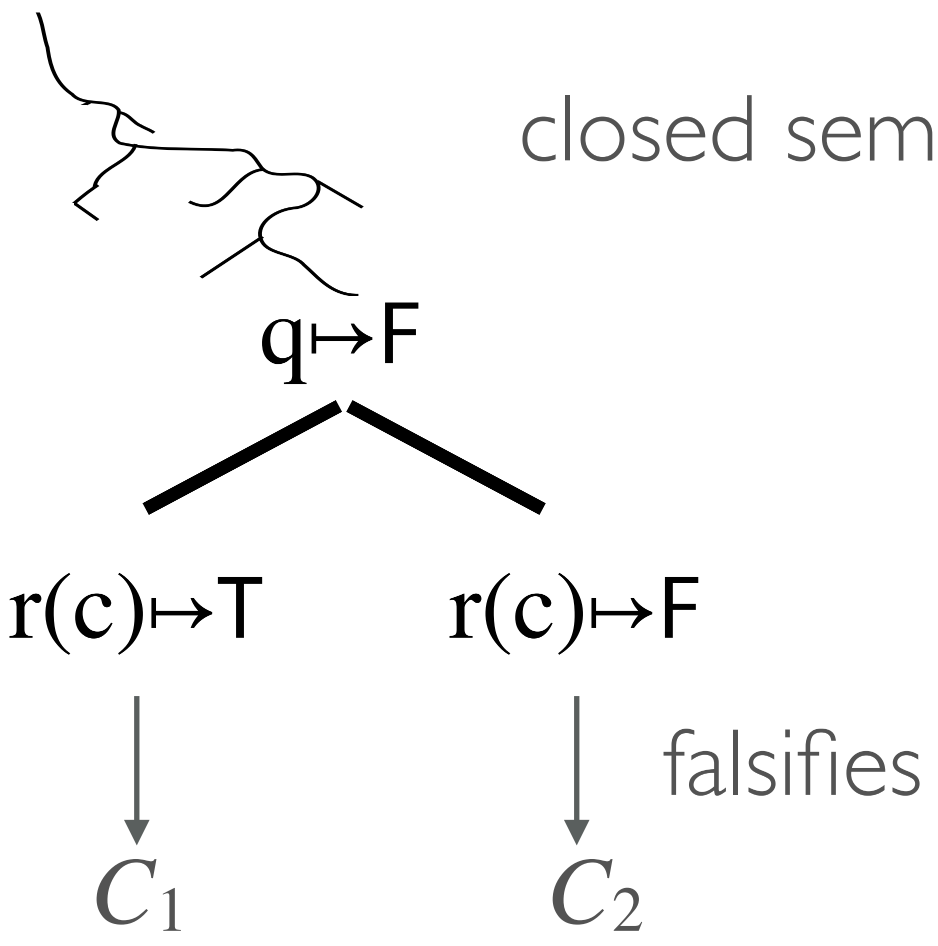


Completeness proof

1. Herbrand's theorem

↳ 2. Deriving {}

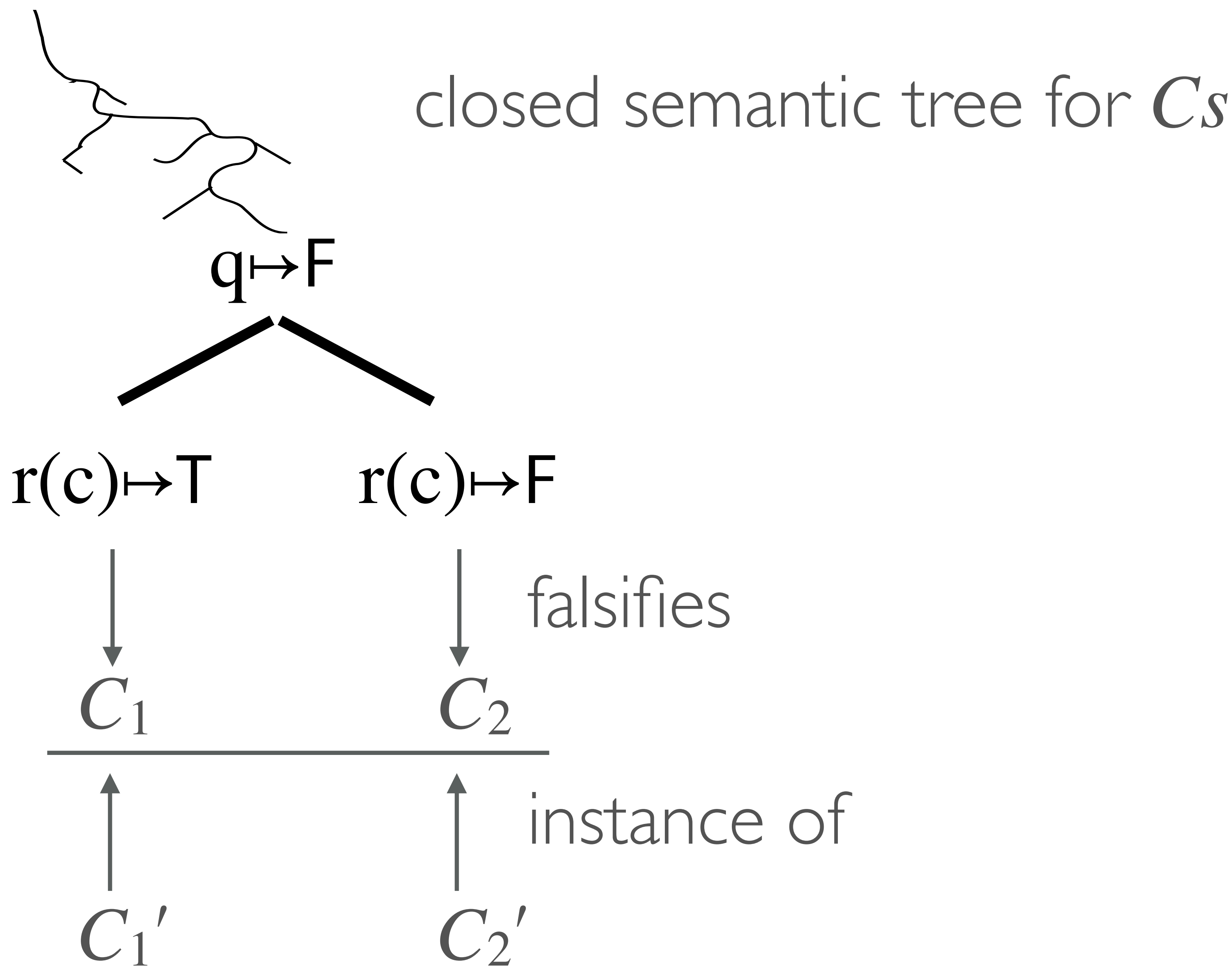
closed semantic tree for *Cs*



Completeness proof

1. Herbrand's theorem

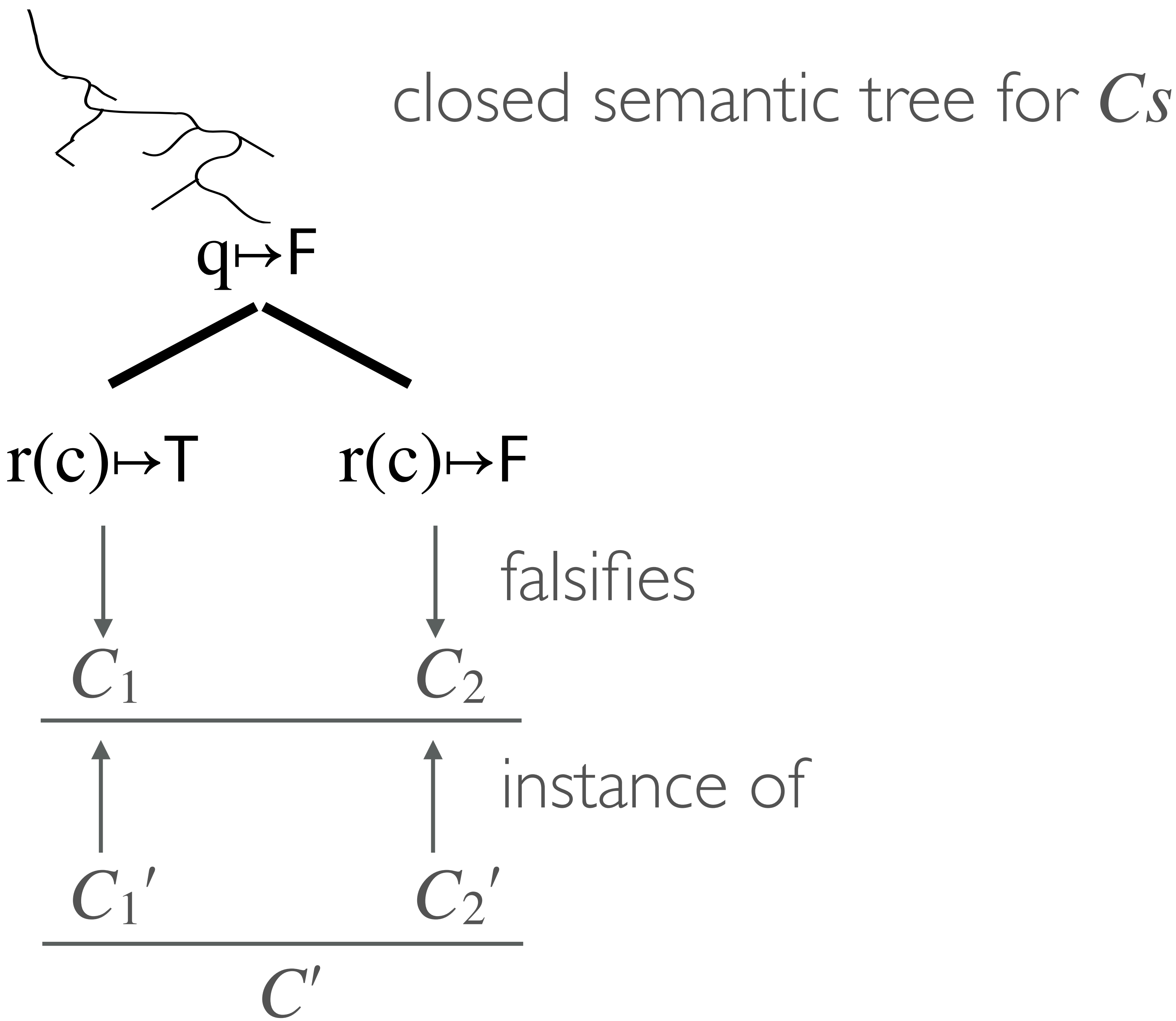
↳ 2. Deriving {}



Completeness proof

1. Herbrand's theorem

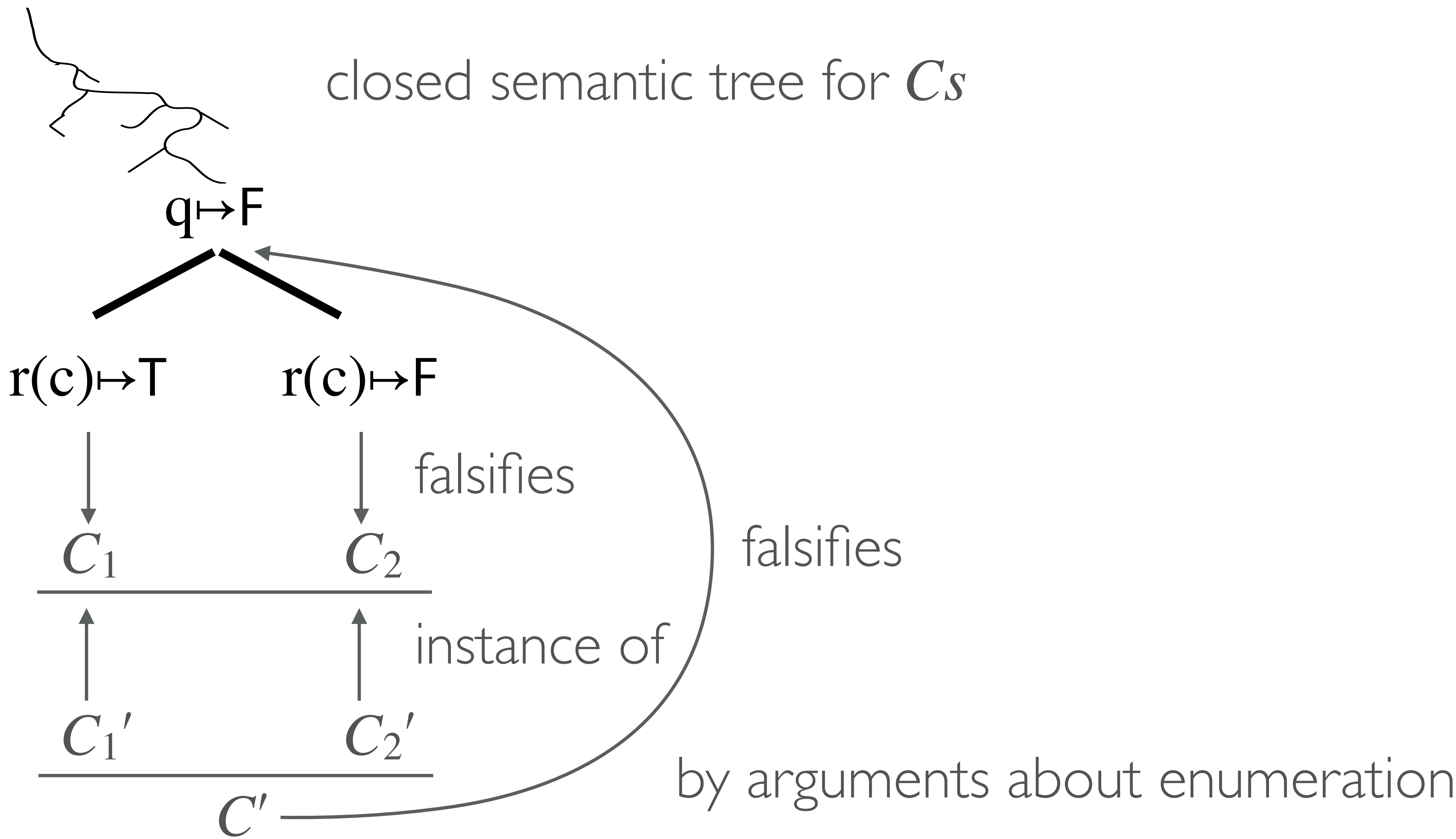
↳ 2. Deriving {}



Completeness proof

1. Herbrand's theorem

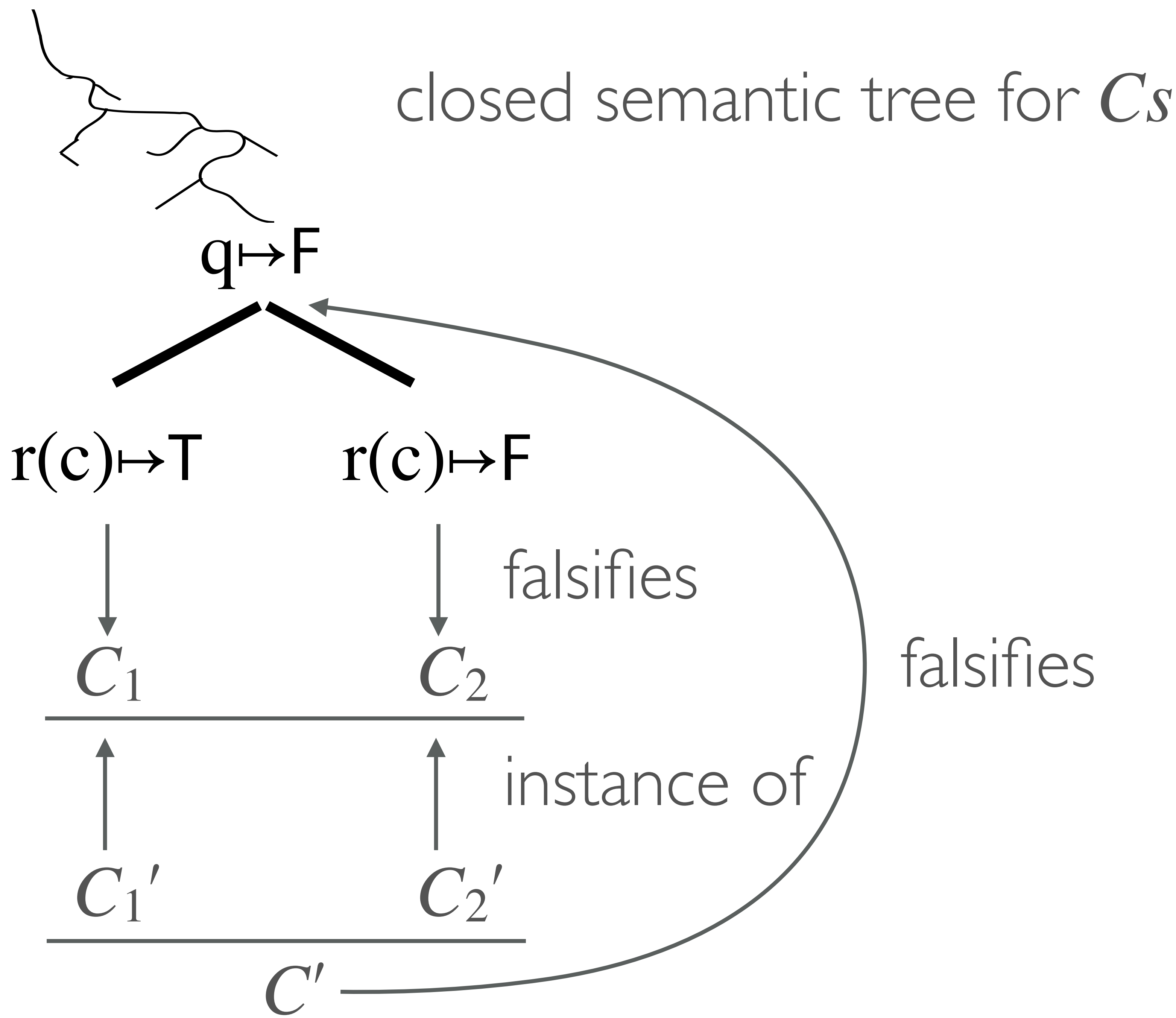
↳ 2. Deriving {}



Completeness proof

1. Herbrand's theorem

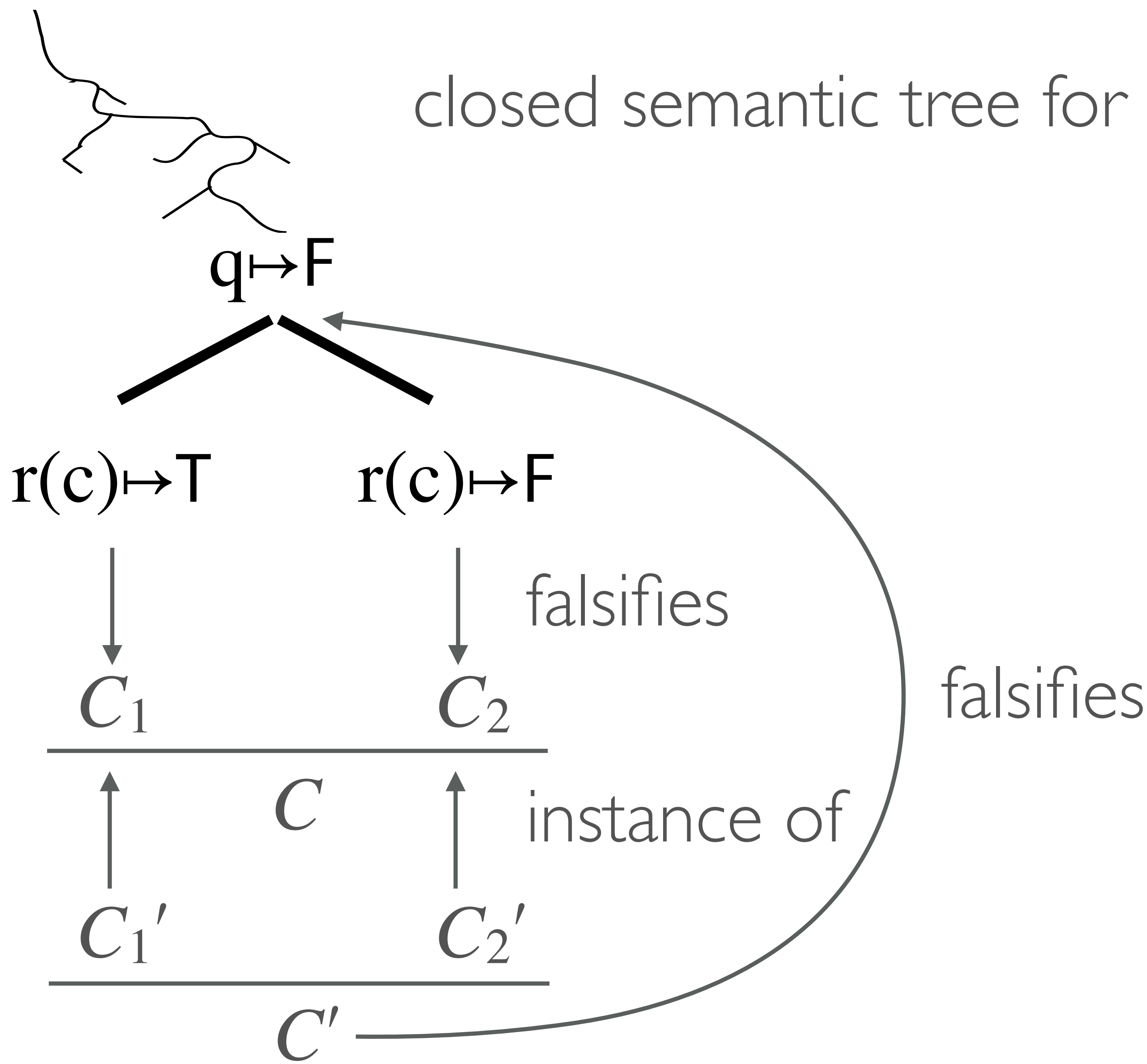
↳2. Deriving {}



Completeness proof

1. Herbrand's theorem

↳ 2. Deriving {}

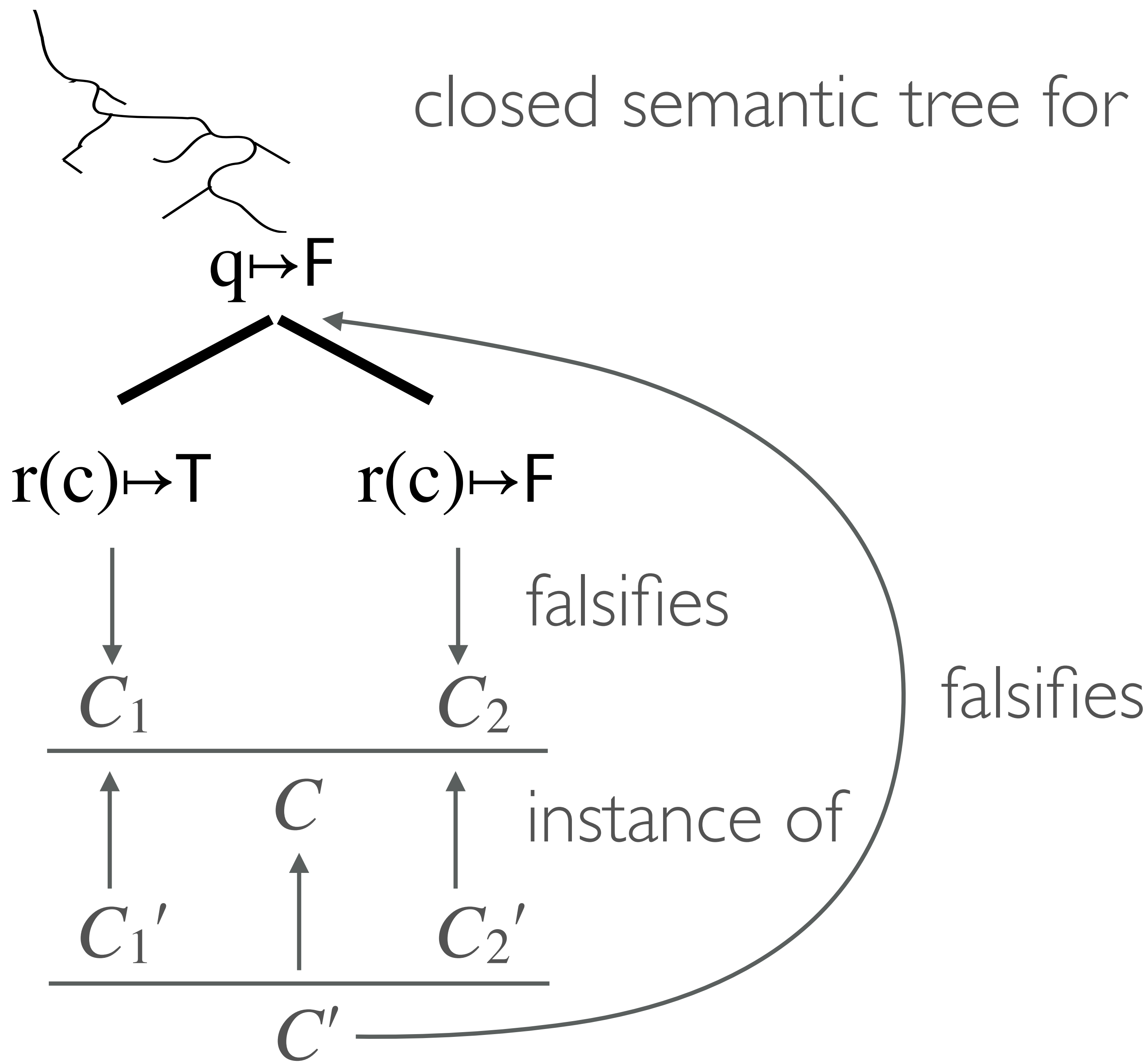


Completeness proof

1. Herbrand's theorem

↳ 2. Deriving {}

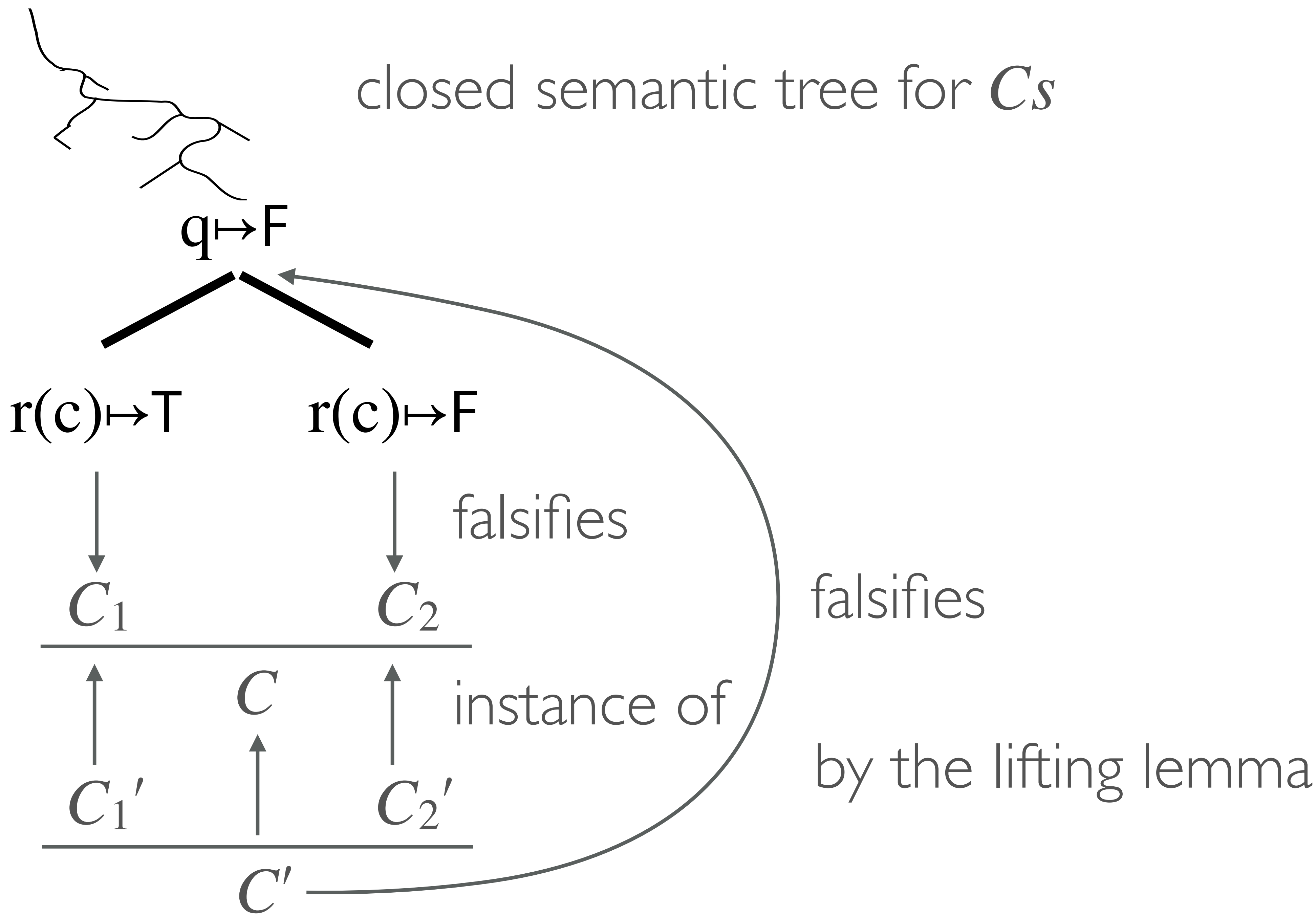
closed semantic tree for *Cs*



Completeness proof

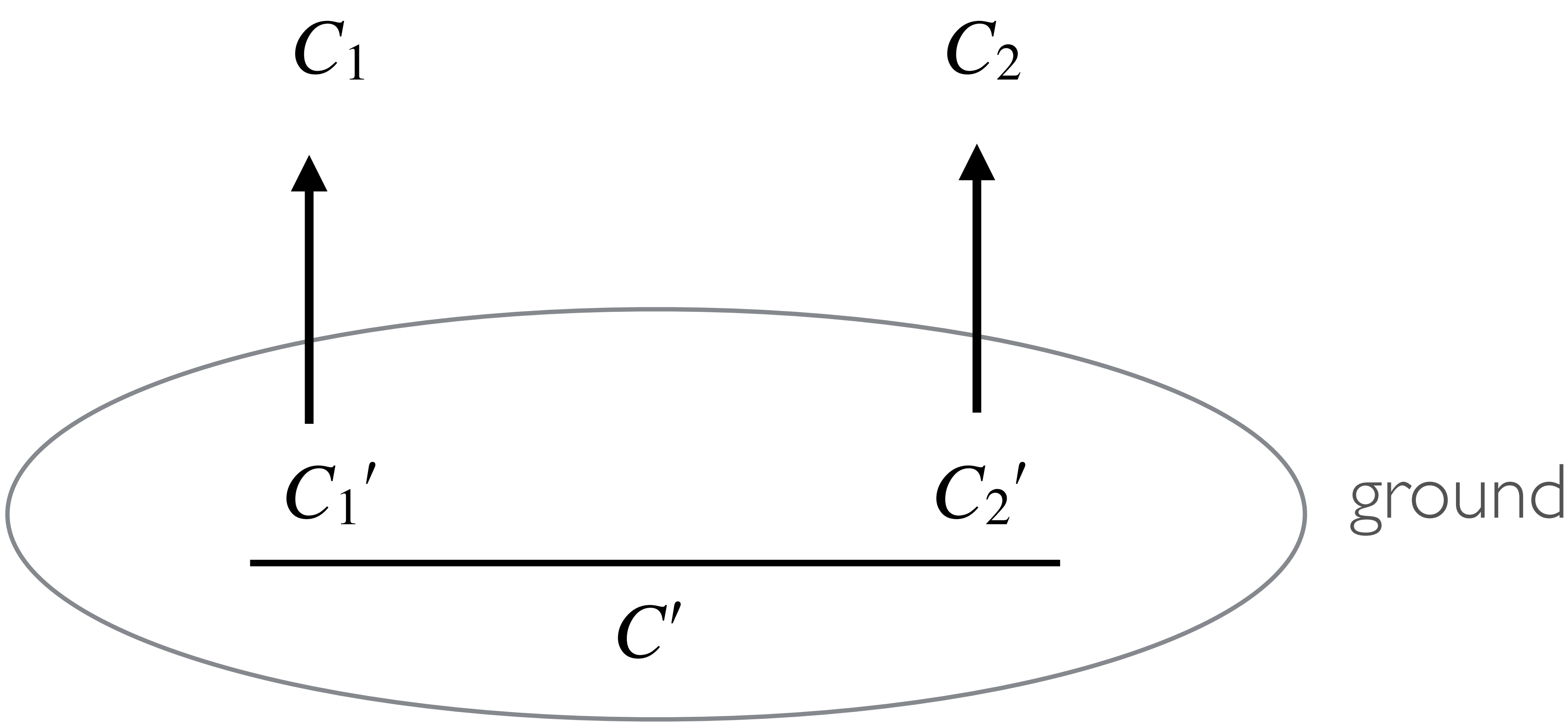
1. Herbrand's theorem

↳ 2. Deriving {}



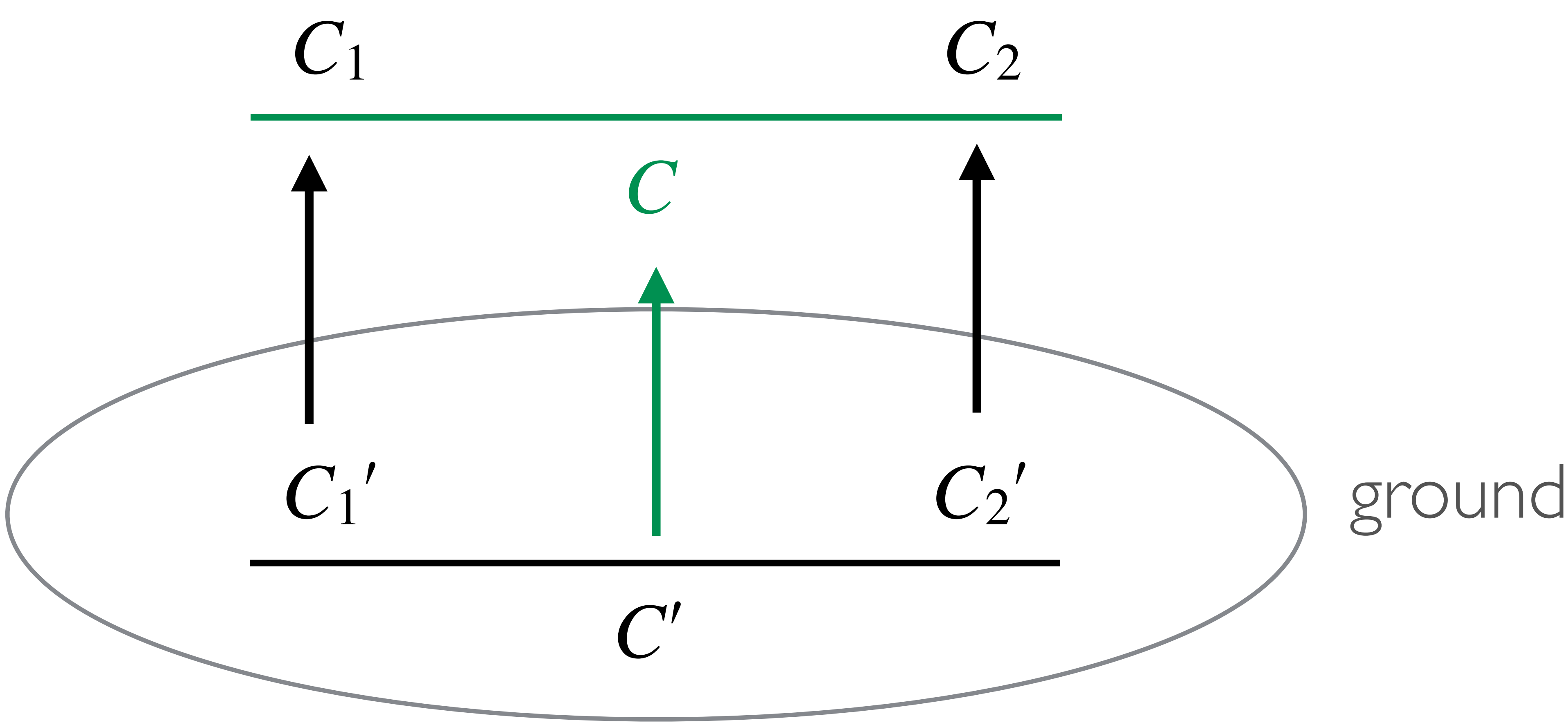
Lifting lemma

↑ means instantiation, e.g. C_1' instance of C_1



Lifting lemma

↑ means instantiation, e.g. C_1' instance of C_1
Black: Assumptions
Green: Established by lemma



Lifting lemma

$\uparrow$ means instantiation, e.g. C_1' instance of C_1

Black: Assumptions

Green: Established by lemma

$$\{\mathrm{p}(x), \mathrm{p}(y), \mathrm{q}(y)\} \qquad \{\neg \mathrm{r}, \neg \mathrm{p}(z)\}$$

$$\uparrow \qquad\qquad\qquad\qquad \uparrow$$

$$\frac{\{\mathrm{p}(\mathrm{c}), \mathrm{q}(\mathrm{c})\} \qquad\qquad \{\neg \mathrm{r}, \neg \mathrm{p}(\mathrm{c})\}}{\{\mathrm{q}(\mathrm{c}), \neg \mathrm{r}\}}$$

ground

Lifting lemma

$\uparrow$ means instantiation, e.g. C_1' instance of C_1

Black: Assumptions

Green: Established by lemma

$$\frac{\{\mathrm{p}(x), \mathrm{p}(y), \mathrm{q}(y)\} \quad \{\neg \mathrm{r}, \neg \mathrm{p}(z)\}}{\{\mathrm{q}(z), \neg \mathrm{r}\}}$$

$$\frac{\{\mathrm{p}(\mathrm{c}), \mathrm{q}(\mathrm{c})\} \quad \{\neg \mathrm{r}, \neg \mathrm{p}(\mathrm{c})\}}{\{\mathrm{q}(\mathrm{c}), \neg \mathrm{r}\}}$$

ground

Lifting lemma

Challenge 1: Showing the existence of MGUs.

Solution: Reuse theorem from IsaFoR.

Challenge 2: Proof by Chang & Lee (1973) is flawed.

Lifting lemma

Let

$$
\begin{aligned}
C &= ((C_1\lambda)\sigma - L_1\sigma) \cup ((C_2\lambda)\sigma - L_2\sigma) \\
&= ((C_1\lambda)\sigma - (\{L_1{}^1, \ldots, L_1^{r_1}\}\lambda)\sigma) \cup ((C_2\lambda)\sigma - (\{L_2{}^1, \ldots, L_2^{r_2}\}\lambda)\sigma) \\
&= (C_1(\lambda \circ \sigma) - \{L_1{}^1, \ldots, L_1^{r_1}\}(\lambda \circ \sigma)) \cup (C_2(\lambda \circ \sigma) - \{L_2{}^1, \ldots, L_2^{r_2}\}(\lambda \circ \sigma)).
\end{aligned}
$$

C is a resolvent of C_1 and C_2. Clearly, C' is an instance of C since

$$
\begin{aligned}
C' &= (C_1{}'\gamma - L_1{}'\gamma) \cup (C_2{}'\gamma - L_2{}'\gamma) \\
&= ((C_1\theta)\gamma - (\{L_1{}^1, \ldots, L_1^{r_1}\}\theta)\gamma) \cup ((C_2\theta)\gamma - (\{L_2{}^1, \ldots, L_2^{r_2}\}\theta)\gamma) \\
&= (C_1(\theta \circ \gamma) - \{L_1{}^1, \ldots, L_1^{r_1}\}(\theta \circ \gamma)) \cup (C_2(\theta \circ \gamma) - \{L_2{}^1, \ldots, L_2^{r_2}\}(\theta \circ \gamma))
\end{aligned}
$$

and $\lambda \circ \sigma$ is more general than $\theta \circ \gamma$. Thus we complete the proof of this lemma.

- Chang & Lee (1973)

Lifting lemma

Let

$$
\begin{aligned}
C &= ((C_1\lambda)\sigma - L_1\sigma) \cup ((C_2\lambda)\sigma - L_2\sigma) \\
&= ((C_1\lambda)\sigma - (\{L_1{}^1, \ldots, L_1^{r_1}\}\lambda)\sigma) \cup ((C_2\lambda)\sigma - (\{L_2{}^1, \ldots, L_2^{r_2}\}\lambda)\sigma) \\
&= (C_1(\lambda \circ \sigma) - \{L_1{}^1, \ldots, L_1^{r_1}\}(\lambda \circ \sigma)) \cup (C_2(\lambda \circ \sigma) - \{L_2{}^1, \ldots, L_2^{r_2}\}(\lambda \circ \sigma)).
\end{aligned}
$$

C is a resolvent of C_1 and C_2. Clearly, C' is an instance of C since

$$
\begin{aligned}
C' &= (C_1{}'\gamma - L_1{}'\gamma) \cup (C_2{}'\gamma - L_2{}'\gamma) \\
&= ((C_1\,\theta)\gamma - (\{L_1{}^1, \ldots, L_1^{r_1}\}\theta)\gamma) \cup ((C_2\,\theta)\gamma - (\{L_2{}^1, \ldots, L_2^{r_2}\}\theta)\gamma) \\
&= (C_1(\theta \circ \gamma) - \{L_1{}^1, \ldots, L_1^{r_1}\}(\theta \circ \gamma)) \cup (C_2(\theta \circ \gamma) - \{L_2{}^1, \ldots, L_2^{r_2}\}(\theta \circ \gamma))
\end{aligned}
$$

and $\lambda \circ \sigma$ is more general than $\theta \circ \gamma$. Thus we complete the proof of this lemma.

- Chang & Lee (1973)

Lifting lemma

The flaw was already discovered by Leitsch (Mathematical Logic Quarterly, 1989).

Chang & Lee do resolution on factors of clauses and remove literals *before* applying substitution.

Other calculi (e.g. by Leitsch (1997)) remove literals *after* applying substitution.

This allows for a simple proof of the lifting lemma.

Completeness

The lifting lemma completes the completeness proof.

```
theorem completeness:
  assumes finite Cs ∧ (∀C∈Cs. finite C)
  assumes ∀(F::hterm fun_denot) (G::hterm pred_denot). ¬eval F G Cs
  shows ∃Cs'. resolution_deriv Cs Cs' ∧ {} ∈ Cs'
<proof>
```

Conclusion

Soundness and completeness of resolution is formalized.

It was particularly challenging to formalize the lifting lemma.

Available in the IsaFoL repository + AFP:
bitbucket.org/jasmin_blanchette/isafol/
isa-afp.org/entries/Resolution_FOL.shtml

I am now working on **extensions** (ordered resolution, redundancy, selection) to get closer to the theory of modern ATP's that use the **superposition** calculus.

References

A machine-oriented logic based on the resolution principle
J. A. Robinson, J. ACM, 1965

Mathematical Logic for Computer Science
M. Ben-Ari, 3rd ed, Springer, 2012

Symbolic Logic and Mechanical Theorem Proving
C. L. Chang and R. C. T. Lee, Academic Press, 1973

The Resolution Calculus
A. Leitsch, Springer, 1997

IsaFoR (Isabelle Formalization of Rewriting)
cl-informatik.uibk.ac.at/software/ceta/
IsaFoR developers

On different concepts of resolution
A. Leitsch, Mathematical Logic Quarterly, 1989

For precise references to the related work, see my paper.

Picture of J. A. Robinson by D. Monniaux [CC BY-SA 3.0], via Wikimedia Commons