Formalizing Bachmair and Ganzinger's Ordered Resolution Prover

Anders Schlichtkrull

Jasmin Blanchette

Dmitriy Traytel

Uwe Waldmann

DTU Technical University of Denmark

VU VRIJE UNIVERSITEIT AMSTERDAM

ETH zürich

mpii max planck institut informatik

Formalizing Automated Reasoning in Proof Assistants

Theorem

Proof assistants are mature enough to be used by researchers in AR.

Proof

We have formalized

- Bachmair and Ganzinger's resolution prover from Handbook of AR
- its soundness theorem
- its completeness theorem.

Corollary

We contribute to a growing library of formalized results in AR.

(Which makes the theorem even more true.)

IsaFoL: Isabelle Formalization of Logic

Framework and methodology

for reasoning about AR in Isabelle.

Adoption by AR researchers

as a convenient way to develop metatheory.

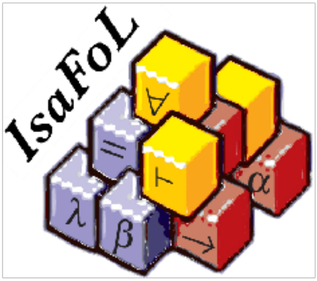
ITP benefits from ATP

… **why not the other way round?**

Isabelle @ RTA, Coq @ POPL

… **now Isabelle @ IJCAR.**

Eat our own dog food!



bitbucket.org/isafol

Ordered Ground Resolution

Ordered resolution with selection

$$\frac{C_1 \vee A_1 \vee \cdots \vee A_1 \quad \ldots \quad C_n \vee A_n \vee \cdots \vee A_n \quad \neg A_1 \vee \cdots \vee \neg A_n \vee D}{C_1 \vee \cdots \vee C_n \vee D}$$

where

(i) either the subclause $\neg A_1 \vee \cdots \vee \neg A_n$ is selected by S in D, or else $S(D)$ is empty, $n = 1$, and A_1 is maximal with respect to D,

(ii) each atom A_i is strictly maximal with respect to C_i, and

(iii) no clause $C_i \vee A_i \vee \ldots \vee A_i$ contains a selected atom.

Ordered Ground Resolution

$$\frac{C_1 \vee A_1 \vee \cdots \vee A_1 \quad \ldots \quad C_n \vee A_n \vee \cdots \vee A_n \quad \neg A_1 \vee \cdots \vee \neg A_n \vee D}{C_1 \vee \cdots \vee C_n \vee D}$$

```
inductive
 ord_resolve :: "'a clause list ⇒ 'a clause ⇒ 'a clause ⇒ bool"
 where
  "length (CAs :: 'a clause list) = n ⟹
   length (Cs  :: 'a clause list) = n ⟹
   length (AAs :: 'a multiset list) = n ⟹
   length (As  :: 'a list) = n ⟹
   n ≠ 0 ⟹
   ∀i < n. (CAs ! i) = (Cs ! i + (poss (AAs ! i))) ⟹
   ∀i < n. (∀A ∈# AAs ! i. A = As ! i) ⟹
   ∀i < n. AAs ! i ≠ {#} ⟹
   eligible As (D + negs (mset As)) ⟹
   ∀i < n. str_maximal_in (As ! i) (Cs ! i) ⟹
   ∀i < n. S (CAs ! i) = {#} ⟹
   ord_resolve CAs (negs (mset As) + D) ((∪# (mset Cs)) + D)"
```

Ordered Ground Resolution

$$\frac{C_1 \vee A_1 \vee \cdots \vee A_1 \quad \ldots \quad C_n \vee A_n \vee \cdots \vee A_n \quad \neg A_1 \vee \cdots \vee \neg A_n \vee D}{C_1 \vee \cdots \vee C_n \vee D}$$

```
inductive
 ord_resolve :: "'a clause list ⇒ 'a clause ⇒ 'a clause ⇒ bool"
 where
  "length (CAs :: 'a clause list) = n ⟹
   length (Cs  :: 'a clause list) = n ⟹
   length (AAs :: 'a multiset list) = n ⟹
   length (As  :: 'a list) = n ⟹
   n ≠ 0 ⟹
   ∀i < n. (CAs ! i) = (Cs ! i + (poss (AAs ! i))) ⟹
   ∀i < n. (∀A ∈# AAs ! i. A = As ! i) ⟹
   ∀i < n. AAs ! i ≠ {#} ⟹
   eligible As (D + negs (mset As)) ⟹
   ∀i < n. str_maximal_in (As ! i) (Cs ! i) ⟹
   ∀i < n. S (CAs ! i) = {#} ⟹
   ord_resolve CAs (negs (mset As) + D) ((∪# (mset Cs)) + D)"
```

} Length of lists

Ordered Ground Resolution

$$\frac{C_1 \vee A_1 \vee \cdots \vee A_1 \quad \ldots \quad C_n \vee A_n \vee \cdots \vee A_n \quad \neg A_1 \vee \cdots \vee \neg A_n \vee D}{C_1 \vee \cdots \vee C_n \vee D}$$

```
inductive
 ord_resolve :: "'a clause list ⇒ 'a clause ⇒ 'a clause ⇒ bool"
 where
  "length (CAs :: 'a clause list) = n ⟹
   length (Cs  :: 'a clause list) = n ⟹
   length (AAs :: 'a multiset list) = n ⟹
   length (As  :: 'a list) = n ⟹
   n ≠ 0 ⟹
   ∀i < n. (CAs ! i) = (Cs ! i + (poss (AAs ! i))) ⟹
   ∀i < n. (∀A ∈# AAs ! i. A = As ! i) ⟹
   ∀i < n. AAs ! i ≠ {#} ⟹
   eligible As (D + negs (mset As)) ⟹
   ∀i < n. str_maximal_in (As ! i) (Cs ! i) ⟹
   ∀i < n. S (CAs ! i) = {#} ⟹
   ord_resolve CAs (negs (mset As) + D) ((∪# (mset Cs)) + D)"
```

Length of lists

Ordered Ground Resolution

$$\frac{C_1 \vee A_1 \vee \cdots \vee A_1 \quad \ldots \quad C_n \vee A_n \vee \cdots \vee A_n \quad \neg A_1 \vee \cdots \vee \neg A_n \vee D}{C_1 \vee \cdots \vee C_n \vee D}$$

```
inductive
 ord_resolve :: "'a clause list ⇒ 'a clause ⇒ 'a clause ⇒ bool"
 where
  "length (CAs :: 'a clause list) = n ⟹
  length (Cs  :: 'a clause list) = n ⟹
   length (AAs :: 'a multiset list) = n ⟹
   length (As  :: 'a list) = n ⟹
   n ≠ 0 ⟹
   ∀i < n. (CAs ! i) = (Cs ! i + (poss (AAs ! i))) ⟹
   ∀i < n. (∀A ∈# AAs ! i. A = As ! i) ⟹
   ∀i < n. AAs ! i ≠ {#} ⟹
   eligible As (D + negs (mset As)) ⟹
   ∀i < n. str_maximal_in (As ! i) (Cs ! i) ⟹
   ∀i < n. S (CAs ! i) = {#} ⟹
   ord_resolve CAs (negs (mset As) + D) ((∪# (mset Cs)) + D)"
```

} Length of lists

Ordered Ground Resolution

$$\frac{C_1 \vee \boxed{A_1 \vee \cdots \vee A_1} \quad \ldots \quad C_n \vee \boxed{A_n \vee \cdots \vee A_n} \quad \neg A_1 \vee \cdots \vee \neg A_n \vee D}{C_1 \vee \cdots \vee C_n \vee D}$$

```
inductive
 ord_resolve :: "'a clause list ⇒ 'a clause ⇒ 'a clause ⇒ bool"
 where
  "length (CAs :: 'a clause list) = n ⟹
   length (Cs  :: 'a clause list) = n ⟹
   length (AAs :: 'a multiset list) = n ⟹
   length (As  :: 'a list) = n ⟹
   n ≠ 0 ⟹
   ∀i < n. (CAs ! i) = (Cs ! i + (poss (AAs ! i))) ⟹
   ∀i < n. (∀A ∈# AAs ! i. A = As ! i) ⟹
   ∀i < n. AAs ! i ≠ {#} ⟹
   eligible As (D + negs (mset As)) ⟹
   ∀i < n. str_maximal_in (As ! i) (Cs ! i) ⟹
   ∀i < n. S (CAs ! i) = {#} ⟹
   ord_resolve CAs (negs (mset As) + D) ((∪# (mset Cs)) + D)"
```

} Length of lists

Ordered Ground Resolution

$$\frac{C_1 \vee A_1 \vee \cdots \vee A_1 \quad \ldots \quad C_n \vee A_n \vee \cdots \vee A_n \quad \neg A_1 \vee \cdots \vee \neg A_n \vee D}{C_1 \vee \cdots \vee C_n \vee D}$$

```
inductive
 ord_resolve :: "'a clause list ⇒ 'a clause ⇒ 'a clause ⇒ bool"
 where
  "length (CAs :: 'a clause list) = n ⟹
   length (Cs  :: 'a clause list) = n ⟹
   length (AAs :: 'a multiset list) = n ⟹
   length (As  :: 'a list) = n ⟹
   n ≠ 0 ⟹
   ∀i < n. (CAs ! i) = (Cs ! i + (poss (AAs ! i))) ⟹
   ∀i < n. (∀A ∈# AAs ! i. A = As ! i) ⟹
   ∀i < n. AAs ! i ≠ {#} ⟹
   eligible As (D + negs (mset As)) ⟹
   ∀i < n. str_maximal_in (As ! i) (Cs ! i) ⟹
   ∀i < n. S (CAs ! i) = {#} ⟹
   ord_resolve CAs (negs (mset As) + D) ((∪# (mset Cs)) + D)"
```

} Length of lists

Ordered Ground Resolution

$$\frac{C_1 \vee A_1 \vee \cdots \vee A_1 \quad \ldots \quad C_n \vee A_n \vee \cdots \vee A_n \quad \neg A_1 \vee \cdots \vee \neg A_n \vee D}{C_1 \vee \cdots \vee C_n \vee D}$$

```
inductive
 ord_resolve :: "'a clause list ⇒ 'a clause ⇒ 'a clause ⇒ bool"
 where
  "length (CAs :: 'a clause list) = n ⟹
   length (Cs  :: 'a clause list) = n ⟹
   length (AAs :: 'a multiset list) = n ⟹
   length (As  :: 'a list) = n ⟹
   n ≠ 0 ⟹
   ∀i < n. (CAs ! i) = (Cs ! i + (poss (AAs ! i))) ⟹
   ∀i < n. (∀A ∈# AAs ! i. A = As ! i) ⟹
   ∀i < n. AAs ! i ≠ {#} ⟹
   eligible As (D + negs (mset As)) ⟹
   ∀i < n. str_maximal_in (As ! i) (Cs ! i) ⟹
   ∀i < n. S (CAs ! i) = {#} ⟹
   ord_resolve CAs (negs (mset As) + D) ((∪# (mset Cs)) + D)"
```

Length of lists

Composition of clauses

Side conditions

Side Conditions

Ordered resolution with selection

$$\frac{C_1 \vee A_1 \vee \cdots \vee A_1 \quad \ldots \quad C_n \vee A_n \vee \cdots \vee A_n \quad \neg A_1 \vee \cdots \vee \neg A_n \vee D}{C_1 \vee \cdots \vee C_n \vee D}$$

where

(i) either the subclause $\neg A_1 \vee \cdots \vee \neg A_n$ is selected by S in D, or else $S(D)$ is empty, $n = 1$, and A_1 is maximal with respect to D,

(ii) each atom A_i is strictly maximal with respect to C_i, and

(iii) no clause $C_i \vee A_i \vee \ldots \vee A_i$ contains a selected atom.

Side Conditions

Ordered resolution with selection

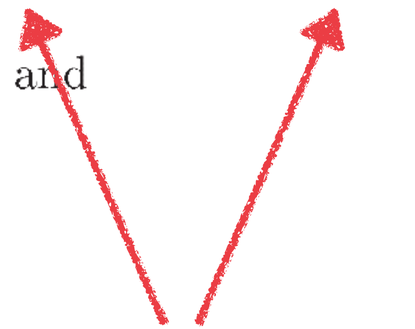
$$\frac{C_1 \vee A_1 \vee \cdots \vee A_1 \quad \ldots \quad C_n \vee A_n \vee \cdots \vee A_n \quad \neg A_1 \vee \cdots \vee \neg A_n \vee D}{C_1 \vee \cdots \vee C_n \vee D}$$

where

(i) either the subclause $\neg A_1 \vee \cdots \vee \neg A_n$ is selected by S in D, or else $S(D)$ is empty, $n = 1$, and A_1 is maximal with respect to D,

(ii) each atom A_i is strictly maximal with respect to C_i, and

(iii) no clause $C_i \vee A_i \vee \ldots \vee A_i$ contains a selected atom.



S in $D \vee \neg A_1 \vee \ldots \vee \neg A_n$

Abstract Redundancy

Γ is the set of inferences that makes up an inference system.

Abstract redundancy is defined:

```
locale redundancy_criterion = inference_system +
  fixes
    Rf :: "'a clause set ⇒ 'a clause set" and
    Ri :: "'a clause set ⇒ 'a inference set"
  assumes
    "Ri N ⊆ Γ" and
    "N ⊆ N' ⟹ Rf N ⊆ Rf N'" and
    "N ⊆ N' ⟹ Ri N ⊆ Ri N'" and
    "N' ⊆ Rf N ⟹ Rf N ⊆ Rf (N - N')" and
    "N' ⊆ Rf N ⟹ Ri N ⊆ Ri (N - N')" and
    "satisfiable (N - Rf N) ⟹ satisfiable N"
```

Standard Redundancy

```
definition Rf :: "'a clause set ⇒ 'a clause set" where
  "Rf N =
    {C. ∃DD. set_mset DD ⊆ N
         ∧ (∀I. I ⊨m DD ⟶ I ⊨ C)
         ∧ (∀D. D ∈# DD ⟶ D < C)}"
```

```
definition Ri :: "'a clause set ⇒ 'a inference set" where
  "Ri N =
    {γ ∈ Γ. ∃DD. set_mset DD ⊆ N
          ∧ (∀I. I ⊨m DD + side_prems_of γ ⟶ I ⊨ concl_of γ)
          ∧ (∀D. D ∈# DD ⟶ D < main_prem_of γ)}"
```

Theorem Proving Processes

```
definition "▷" :: "'a clause set ⇒ 'a clause set ⇒ bool" where
  "M ▷ N ⟷ N - M ⊆ concls_of (inferences_from M) ∧ M - N ⊆ Rf N"
```

The deduced clauses

The deleted clauses

Saturation up to redundancy:

```
definition saturated :: "'a clause set ⇒ bool" where
  "saturated N ⟷ inferences_from (N - Rf N) ⊆ Ri N"
```

Completeness of Ordered Ground Resolution

```
lemma saturated_complete_if:
  assumes
    "saturated N" and
    "¬ satisfiable N"
  shows "{#} ∈ N"
```

Ordered First-Order Resolution

Ordered resolution with selection

$$\frac{C_1 \vee A_{11} \vee \ldots \vee A_{1k_1} \quad \ldots \quad C_n \vee A_{1n} \vee \ldots \vee A_{nk_n} \quad \neg A_1 \vee \ldots \vee \neg A_n \vee D}{C_1\sigma \vee \ldots \vee C_n\sigma \vee D\sigma}$$

where σ is a most general simultaneous solution of all unification problems $A_{i1} = \ldots = A_{ik_i} = A_i$, where $1 \leq i \leq n$, and

(i) either $A_1, \ldots, A_n$ are selected in D, or else nothing is selected in D, $n = 1$, and $A_1\sigma$ is maximal in $D\sigma$,

(ii) each atom $A_{ii}\sigma$ is strictly maximal with respect to $C_i\sigma$, and

(iii) no clause $C_i \vee A_{i1} \vee \ldots \vee A_{ik_i}$ contains a selected atom.

Ordered First-Order Resolution

$$\frac{C_1 \vee A_{11} \vee \ldots \vee A_{1k_1} \quad \ldots \quad C_n \vee A_{1n} \vee \ldots \vee A_{nk_n} \quad \neg A_1 \vee \ldots \vee \neg A_n \vee D}{C_1\sigma \vee \ldots \vee C_n\sigma \vee D\sigma}$$

```
inductive
  ord_resolve:: "'a clause list ⇒ 'a clause ⇒ 's ⇒ 'a clause ⇒ bool"
  where
   "length (CAs :: 'a clause list) = n ⟹
    length (Cs  :: 'a clause list) = n ⟹
    length (AAs :: 'a multiset list) = n ⟹
    length (As  :: 'a list) = n ⟹
    n ≠ 0 ⟹
    ∀i < n. (CAs ! i) = (Cs ! i + (poss (AAs ! i))) ⟹
    ∀i < n. AAs ! i ≠ {#} ⟹
    Some σ = mgu (set_mset ` (set (map2 add_mset As AAs))) ⟹
    eligible σ As (D + negs (mset As)) ⟹
    ∀i. i < n ⟶ str_maximal_in (As ! i ·a σ) ((Cs ! i) · σ) ⟹
    ∀i < n. S (CAs ! i) = {#} ⟹
    ord_resolve CAs (D + negs (mset As)) σ (((∪# (mset Cs)) + D) · σ)"
```

Length of lists

Composition of clauses

Side conditions

Ordered First-Order Resolution

Ordered resolution with selection

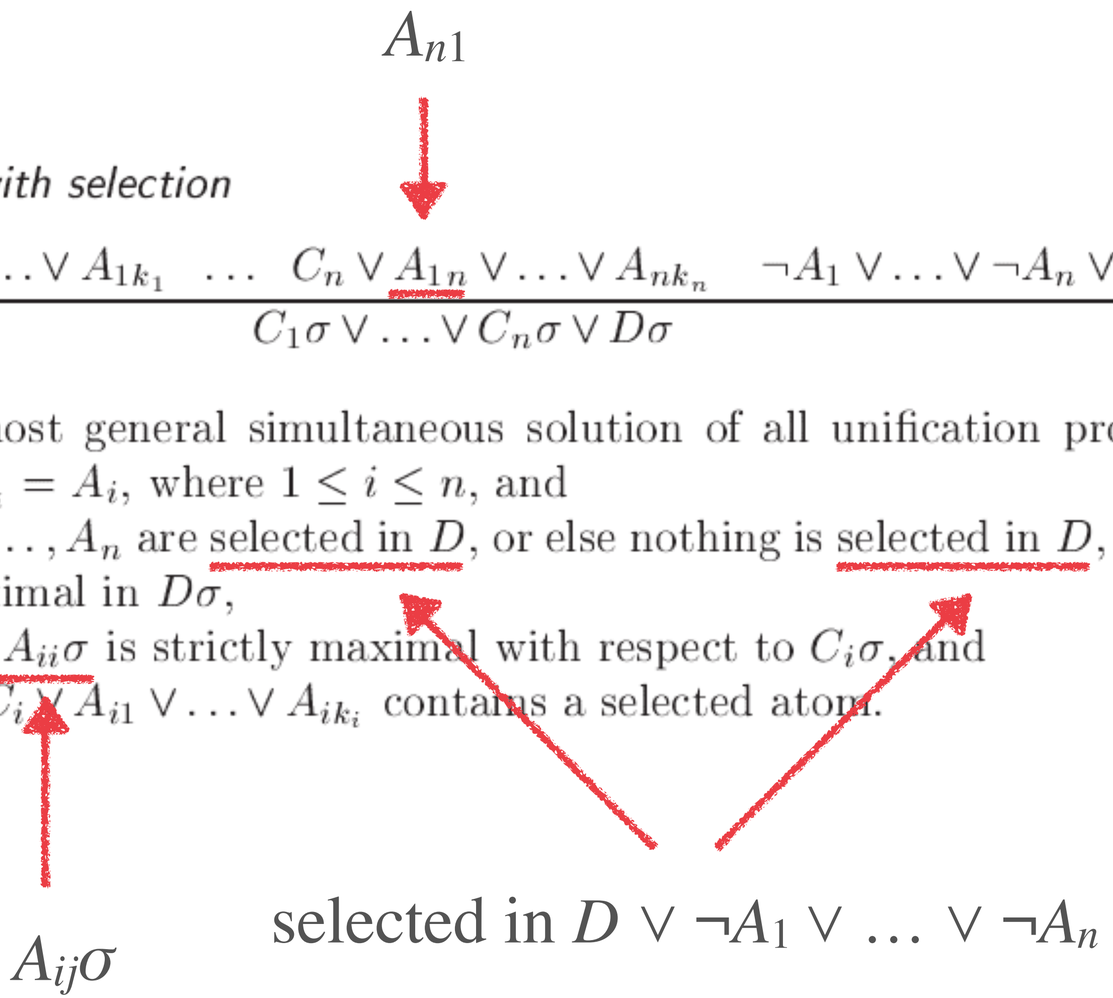
$$\frac{C_1 \vee A_{11} \vee \ldots \vee A_{1k_1} \quad \ldots \quad C_n \vee A_{1n} \vee \ldots \vee A_{nk_n} \quad \neg A_1 \vee \ldots \vee \neg A_n \vee D}{C_1\sigma \vee \ldots \vee C_n\sigma \vee D\sigma}$$

where σ is a most general simultaneous solution of all unification problems $A_{i1} = \ldots = A_{ik_i} = A_i$, where $1 \leq i \leq n$, and

(i) either $A_1, \ldots, A_n$ are selected in D, or else nothing is selected in D, $n = 1$, and $A_1\sigma$ is maximal in $D\sigma$,

(ii) each atom $A_{ii}\sigma$ is strictly maximal with respect to $C_i\sigma$, and

(iii) no clause $C_i \vee A_{i1} \vee \ldots \vee A_{ik_i}$ contains a selected atom.



Prover

Tautology deletion

$$\mathcal{N} \cup \{C\} \mid \mathcal{P} \mid \mathcal{O} \implies \mathcal{N} \mid \mathcal{P} \mid \mathcal{O} \qquad \text{if } C \text{ is a tautology}$$

Forward subsumption

$$\mathcal{N} \cup \{C\} \mid \mathcal{P} \mid \mathcal{O} \implies \mathcal{N} \mid \mathcal{P} \mid \mathcal{O} \qquad \text{if some clause in } \mathcal{P} \cup \mathcal{O} \text{ subsumes } C$$

Backward subsumption

$$\mathcal{N} \mid \mathcal{P} \cup \{C\} \mid \mathcal{O} \implies \mathcal{N} \mid \mathcal{P} \mid \mathcal{O}$$
$$\mathcal{N} \mid \mathcal{P} \mid \mathcal{O} \cup \{C\} \implies \mathcal{N} \mid \mathcal{P} \mid \mathcal{O}$$
$$\text{if some clause in } \mathcal{N} \text{ properly subsumes } C$$

Forward reduction

$$\mathcal{N} \cup \{C \vee L\} \mid \mathcal{P} \mid \mathcal{O} \implies \mathcal{N} \cup \{C\} \mid \mathcal{P} \mid \mathcal{O}$$
$$\text{if there is a clause } D \vee L' \text{ in } \mathcal{P} \cup \mathcal{O} \text{ such that } \overline{L} = L'\sigma \text{ and } D\sigma \subseteq C$$

Backward reduction

$$\mathcal{N} \mid \mathcal{P} \cup \{C \vee L\} \mid \mathcal{O} \implies \mathcal{N} \mid \mathcal{P} \cup \{C\} \mid \mathcal{O}$$
$$\mathcal{N} \mid \mathcal{P} \mid \mathcal{O} \cup \{C \vee L\} \implies \mathcal{N} \mid \mathcal{P} \cup \{C\} \mid \mathcal{O}$$
$$\text{if there is a clause } D \vee L' \text{ in } \mathcal{N} \text{ such that } \overline{L} = L'\sigma \text{ and } D\sigma \subseteq C$$

Clause processing

$$\mathcal{N} \cup \{C\} \mid \mathcal{P} \mid \mathcal{O} \implies \mathcal{N} \mid \mathcal{P} \cup \{C\} \mid \mathcal{O}$$

Inference computation

$$\emptyset \mid \mathcal{P} \cup \{C\} \mid \mathcal{O} \implies \mathcal{N} \mid \mathcal{P} \mid \mathcal{O} \cup \{C\}$$

where $\mathcal{N}$ = all conclusions of inferences where one premise is C and the others are in $\mathcal{O}$

Prover

A state is a triple ($\mathcal{N}$,$\mathcal{P}$,$\mathcal{O}$) of sets of respectively

$\mathcal{N}$ew, $\mathcal{P}$rocessed, and $\mathcal{O}$ld clauses.

Let's look at three of the rules:

Forward reduction

$$\mathcal{N} \cup \{C \vee L\} \mid \mathcal{P} \mid \mathcal{O} \implies \mathcal{N} \cup \{C\} \mid \mathcal{P} \mid \mathcal{O}$$

if there is a clause $D \vee L'$ in $\mathcal{P} \cup \mathcal{O}$ such that $\overline{L} = L'\sigma$ and $D\sigma \subset C$

Clause processing

$$\mathcal{N} \cup \{C\} \mid \mathcal{P} \mid \mathcal{O} \implies \mathcal{N} \mid \mathcal{P} \cup \{C\} \mid \mathcal{O}$$

Inference computation

$$\emptyset \mid \mathcal{P} \cup \{C\} \mid \mathcal{O} \implies \mathcal{N} \mid \mathcal{P} \mid \mathcal{O} \cup \{C\}$$

where $\mathcal{N}$ = all conclusions of inferences where one premise is C and the others are in $\mathcal{O}$

A Simple Proof?

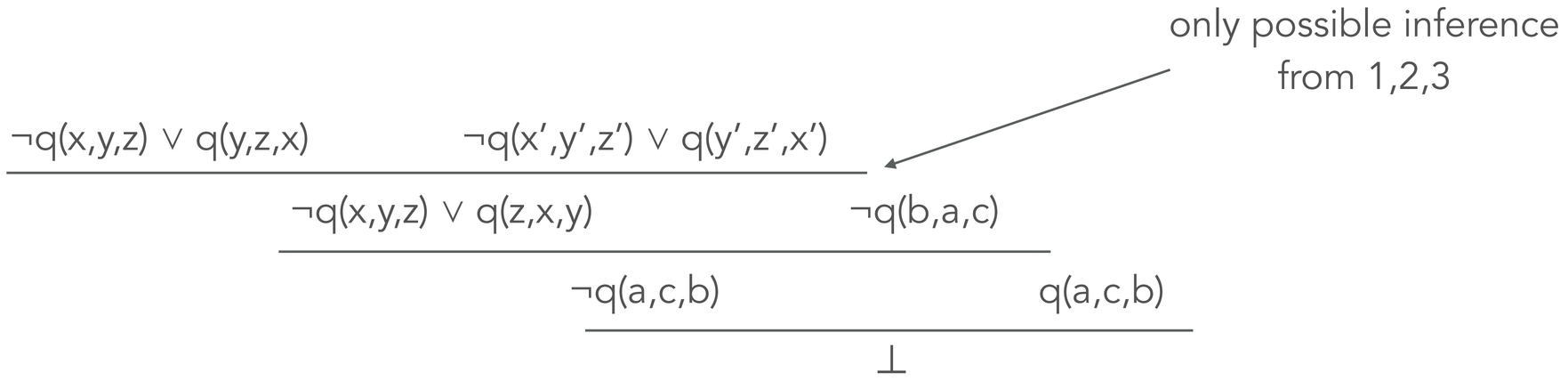
Consider the set containing and the selection function $S(C)=\varnothing$ for all C.

1) q(a,c,b)

2) ¬q(x,y,z) ∨ q(y,z,x)

3) ¬q(b,a,c)

with ordering q(c, b, a) > q(b, a, c) > q(a, c, b).



However, the prover will not do this inference!

Repairing the Incompleteness

Inference computation

$$\emptyset \mid \mathcal{P} \cup \{C\} \mid \mathcal{O} \implies \mathcal{N} \mid \mathcal{P} \mid \mathcal{O} \cup \{C\}$$

where $\mathcal{N}$ = all conclusions of inferences where one premise is C

and the others are in $\mathcal{O} \cup \{C\}$

Prover

Tautology deletion

$$\mathcal{N} \cup \{C\} \mid \mathcal{P} \mid \mathcal{O} \implies \mathcal{N} \mid \mathcal{P} \mid \mathcal{O} \qquad \text{if } C \text{ is a tautology}$$

Forward subsumption

$$\mathcal{N} \cup \{C\} \mid \mathcal{P} \mid \mathcal{O} \implies \mathcal{N} \mid \mathcal{P} \mid \mathcal{O} \qquad \text{if some clause in } \mathcal{P} \cup \mathcal{O} \text{ subsumes } C$$

Backward subsumption

$$\mathcal{N} \mid \mathcal{P} \cup \{C\} \mid \mathcal{O} \implies \mathcal{N} \mid \mathcal{P} \mid \mathcal{O}$$
$$\mathcal{N} \mid \mathcal{P} \mid \mathcal{O} \cup \{C\} \implies \mathcal{N} \mid \mathcal{P} \mid \mathcal{O}$$
$$\text{if some clause in } \mathcal{N} \text{ properly subsumes } C$$

Forward reduction

$$\mathcal{N} \cup \{C \vee L\} \mid \mathcal{P} \mid \mathcal{O} \implies \mathcal{N} \cup \{C\} \mid \mathcal{P} \mid \mathcal{O}$$
$$\text{if there is a clause } D \vee L' \text{ in } \mathcal{P} \cup \mathcal{O} \text{ such that } \overline{L} = L'\sigma \text{ and } D\sigma \subseteq C$$

Backward reduction

$$\mathcal{N} \mid \mathcal{P} \cup \{C \vee L\} \mid \mathcal{O} \implies \mathcal{N} \mid \mathcal{P} \cup \{C\} \mid \mathcal{O}$$
$$\mathcal{N} \mid \mathcal{P} \mid \mathcal{O} \cup \{C \vee L\} \implies \mathcal{N} \mid \mathcal{P} \cup \{C\} \mid \mathcal{O}$$
$$\text{if there is a clause } D \vee L' \text{ in } \mathcal{N} \text{ such that } \overline{L} = L'\sigma \text{ and } D\sigma \subseteq C$$

Clause processing

$$\mathcal{N} \cup \{C\} \mid \mathcal{P} \mid \mathcal{O} \implies \mathcal{N} \mid \mathcal{P} \cup \{C\} \mid \mathcal{O}$$

Inference computation

$$\emptyset \mid \mathcal{P} \cup \{C\} \mid \mathcal{O} \implies \mathcal{N} \mid \mathcal{P} \mid \mathcal{O} \cup \{C\}$$

where $\mathcal{N}$ = all conclusions of inferences where one premise is C and the others are in $\mathcal{O} \cup \{C\}$

Repaired Prover, Formalized

```
inductive resolution_prover :: "'a state ⇒ 'a state ⇒ bool"
  tautology_deletion:
    "Neg A ∈# C ⟹ Pos A ∈# C ⟹ (N ∪ {C}, P, Q) ↝ (N, P, Q)"
| forward_subsumption:
    "(∃D ∈ P ∪ Q. subsumes D C) ⟹ (N ∪ {C}, P, Q) ↝ (N, P, Q)"
| backward_subsumption_P:
    "(∃D ∈ N. properly_subsumes D C) ⟹ (N, P ∪ {C}, Q) ↝ (N, P, Q)"
| backward_subsumption_Q:
    "(∃D ∈ N. properly_subsumes D C) ⟹ (N, P, Q ∪ {C}) ↝ (N, P, Q)"
| forward_reduction:
    "(∃D L'. D + {#L'#} ∈ P ∪ Q ∧ - L = L' ·l σ ∧ D · σ ≤# C) ⟹
      (N ∪ {C + {#L#}}, P, Q) ↝ (N ∪ {C}, P, Q)"
| backward_reduction_P:
    "(∃D L'. D + {#L'#} ∈ N ∧ - L = L' ·l σ ∧ D · σ ≤# C) ⟹
      (N, P ∪ {C + {#L#}}, Q) ↝ (N, P ∪ {C}, Q)"
| backward_reduction_Q:
    "(∃D L'. D + {#L'#} ∈ N ∧ - L = L' ·l σ ∧ D · σ ≤# C) ⟹
      (N, P, Q ∪ {C + {#L#}}) ↝ (N, P ∪ {C}, Q)"
| clause_processing:
    "(N ∪ {C}, P, Q) ↝ (N, P ∪ {C}, Q)"
| inference_computation:
    "N = concls_of (inferences_between Q C) ⟹
     ({}, P ∪ {C}, Q) ↝ (N, P, Q ∪ {C})"
```

Completeness

Bachmair and Ganzinger prove completeness under the assumption of fairness.

A sequence of states is fair if

- no clause eventually always stays in $\mathscr{N}$, and
- no clause eventually always stays in $\mathscr{P}$.

The big picture of the proof is roughly:

1. Assume the initial set of clauses is unsatisfied.
2. Project the sequence of states down to the ground level.
3. By fairness, any non-redundant clause eventually gets old.
4. Therefore the grounding of the old clauses is saturated.
5. Using completeness of ground resolution we get the empty clause.

Completeness

Bachmair and Ganzinger prove completeness under the assumption of fairness.

A sequence of states is fair if

- no clause eventually always stays in $\mathscr{N}$, and
- no clause eventually always stays in $\mathscr{P}$.

The big picture of the proof is roughly:

1. Assume the initial set of clauses is unsatisfied.
2. Project the sequence of states down to the ground level.
3. By fairness, any non-redundant clause eventually gets old.
4. Therefore the grounding of the old clauses is saturated.
5. Using completeness of ground resolution we get the empty clause.

Any Nonredundant Ground Clause Eventually Gets Old?

A flawed proof of this lemma is provided by the authors.

The big picture of their proof is:

1. Consider a non-redundant ground clause C in the grounding of $\mathcal{N}$ or $\mathcal{P}$.
2. It is the instance of some clause D in $\mathcal{N}$ or $\mathcal{P}$.
3. By fairness D cannot stay forever in $\mathcal{N}$ or $\mathcal{P}$ and thus must have been removed by some rule.
4. By inspection of the rules we can see that D must eventually be in $\mathcal{O}$.
5. Therefore C must be in the grounding of $\mathcal{O}$.

Counterexample to 4:

({p(x)}, {p(f(y))}, {}) $\Longrightarrow$ ({p(x)}, {}, {})

Backward subsumption

$$\mathcal{N} \mid \mathcal{P} \cup \{D\} \mid \mathcal{O} \implies \mathcal{N} \mid \mathcal{P} \mid \mathcal{O}$$

if some clause in $\mathcal{N}$ properly subsumes D

Any Nonredundant Ground Clause Eventually Gets Old?

smallest w.r.t. subsumption

A flawed proof of this lemma is provided by the authors.

The big picture of their proof is:

1. Consider a non-redundant ground clause C in the grounding of $\mathcal{N}$ or $\mathcal{P}$.
2. It is the instance of some clause D in $\mathcal{N}$ or $\mathcal{P}$.
3. By fairness D cannot stay forever in $\mathcal{N}$ or $\mathcal{P}$ and thus must have been removed by some rule.
4. By inspection of the rules we can see that D must eventually be in $\mathcal{O}$.
5. Therefore C must be in the grounding of $\mathcal{O}$.

We know D exists since strict subsumption is well founded.

Completeness of FO Ordered Resolution Prover

```
theorem completeness:
  assumes
    renaming: "∧ρ C. is_renaming ρ ⟹ Sel (C · ρ) = Sel C · ρ"
  assumes
    "derivation (op ⇝) S" and
    "fair_state_seq S" and
    "¬ satisfiable (grounding_of_state S₀)" and
  shows "{#} ∈ clss_of_state S∞"
```

What Was Tricky?

Many lemmas and their proofs have flaws:

- Lemma 3.4 contains a wrong claim.
- Lemma 3.7 has a counter example, but the lemma is fortunately never used.
- Section 4.1 contains results that require soundness, but it is claimed that consistency-preservability is enough.
- Theorem 4.3's proof does not cover all cases.
- Lemma 4.10 uses a selection function, but it is not clear which.
- Lemma 4.10 concerns the extension of a proof system, but it is not clear which.
- Lemma 4.10 is only proved for finite sequences, but later used for infinite ones.
- Lemma 4.12 has a 2 sentence proof. In Isabelle the proof is about 500 LOC.

General issue: Lemmas are stated, but not referenced later.

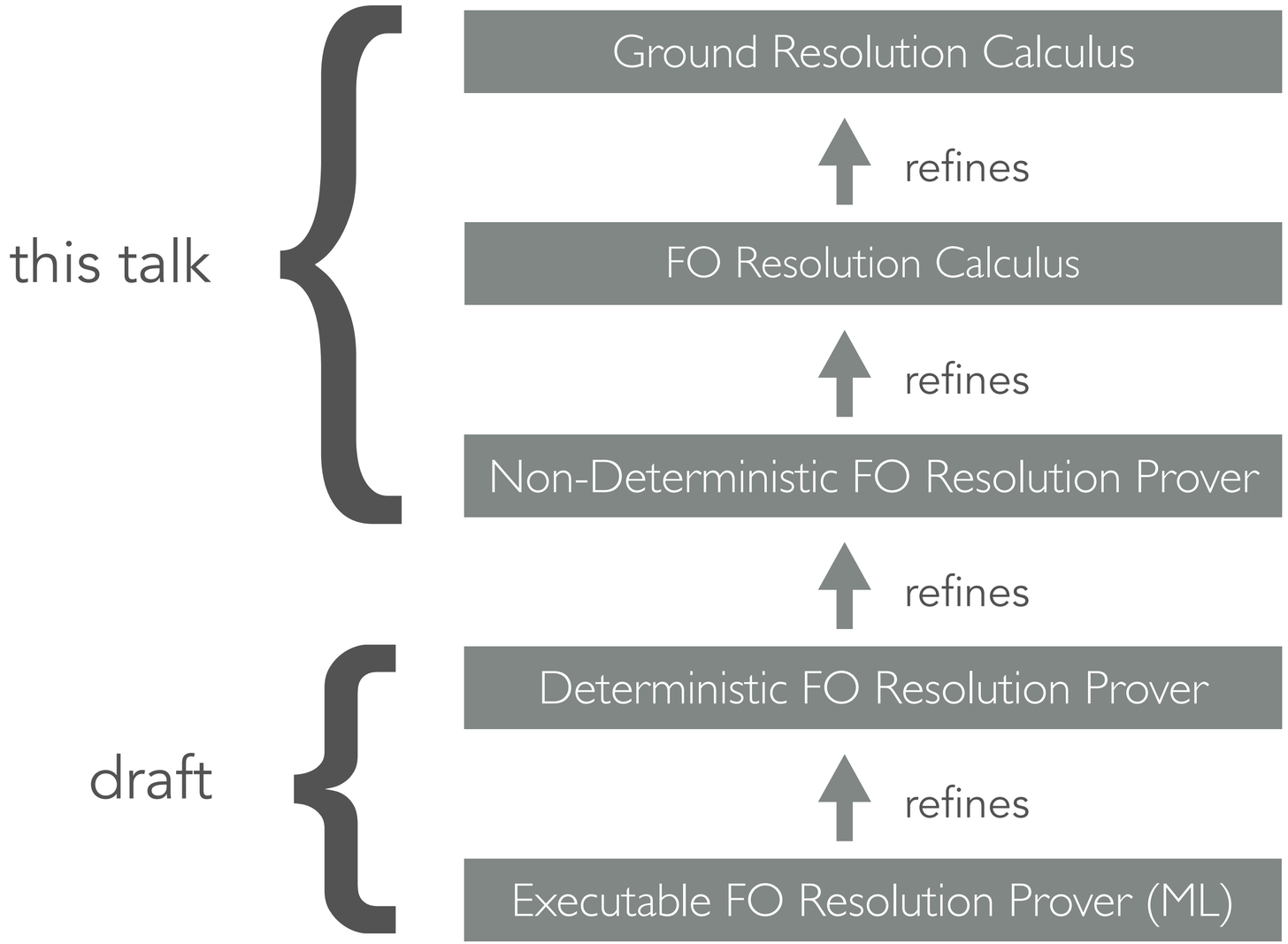
What Was Tricky?

Procedure for dealing with these problems:

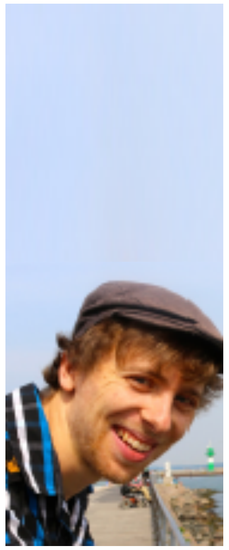
1. Rewrite the chapter's proofs to handwritten pseudo-Isabelle.
2. Fill in the gaps and write where lemmas are used.
3. Turn the pseudo-Isabelle into real Isabelle, but with **sorry** instead of proofs.
4. Replace the **sorry**s with proofs.

Backtrack when necessary.

Refinement Chain

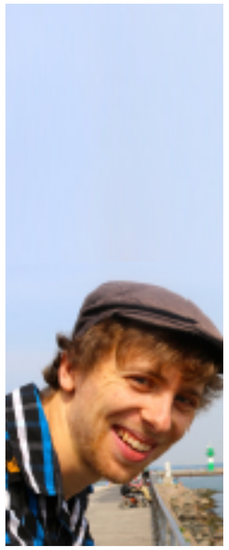


What Can We Learn from Formalization?



I have a question about Chapter 2 of the Handbook of AR. On p. 45, they introduce a function S_M to select literals in ground instances of clauses in M. The definition says "(ii) S_M(C) = S(C), if C is not in K."
Do you know why they define the function for clauses not in K? Is it because they don't want to leave S_M underspecified or does this serve a special purpose?

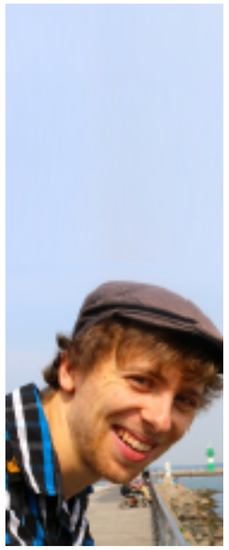
What Can We Learn from Formalization?



I have a question about Chapter 2 of the Handbook of AR. On p. 45, they introduce a function S_M to select literals in ground instances of clauses in M. The definition says "(ii) S_M(C) = S(C), if C is not in K."
Do you know why they define the function for clauses not in K? Is it because they don't want to leave S_M underspecified or does this serve a special purpose?

As far as I can see, S_M is only needed for ground instances, and then case (ii) is irrelevant. **I guess** they just wanted to define S_M as a total function over (possibly non-ground) clauses.

What Can We Learn from Formalization?



I have a question about Chapter 2 of the Handbook of AR. On p. 45, they introduce a function S_M to select literals in ground instances of clauses in M. The definition says "(ii) S_M(C) = S(C), if C is not in K."
Do you know why they define the function for clauses not in K? Is it because they don't want to leave S_M underspecified or does this serve a special purpose?

As far as I can see, S_M is only needed for ground instances, and then case (ii) is irrelevant. **I guess** they just wanted to define S_M as a total function over (possibly non-ground) clauses.

I tried to change the definition in the formalization to return {} if C is not in K. With this definition the key properties also hold, and **the proof goes through**.

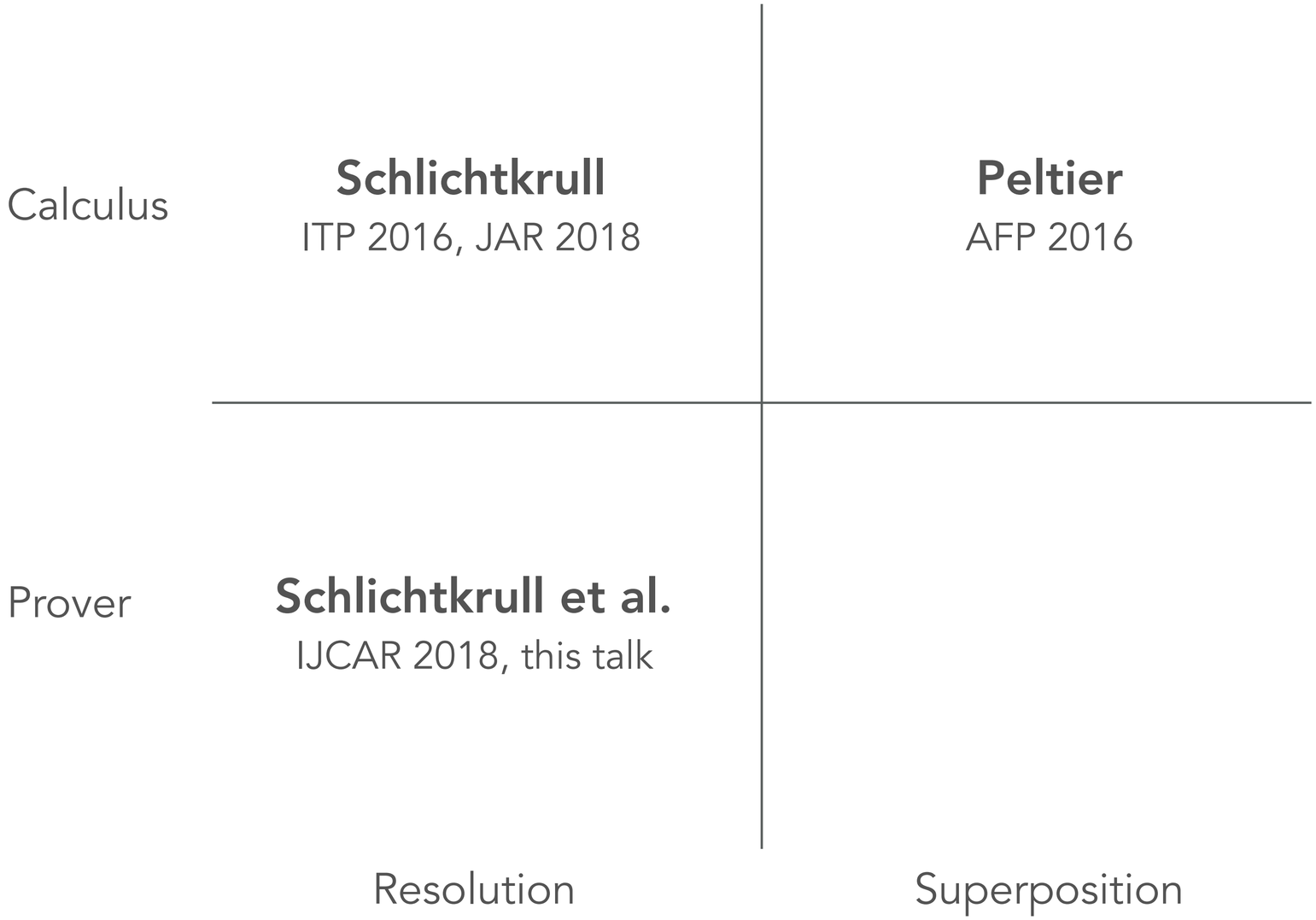
Suitability of Isabelle

Isabelle was entirely suitable for this development.

We benefitted from

- codatatypes
- Isabelle/jEdit
- Isar proofs
- locales
- Sledgehammer.

Related Work



Conclusion

We have formalized Bachmair and Ganzinger's prover.

The chapter withstood the test of formalization after

- allowing the prover to do self-inferences
- repairing some of the proofs.

The formalization

- clarifies the chapter's content
- contributes to the effort of making useful tools for developing AR metatheory.