MANCHESTER 1824

The University of Manchester

Non-cyclic sorts for first-order satisfiability

(or how to win first-order satisfiability at CASC)

Konstantin Korovin[1]

The University of Manchester

korovin@cs.man.ac.uk

FroCoS 2013

[1]supported by a Royal Society University Fellowship

First-order satisfiability

The problem: Given a set of first-order sentences S check whether S is satisfiable.

Complementary to proof finding: Given a set of first-order sentences check whether it is unsatisfiable.

Where satisfiability checking is used?

- in verification for finding errors in systems
- in combinatorial reasoning: scheduling, planning, etc. for finding solutions
- in checking consistency of ontologies, theories, axiomatisations
- disproving conjectures
- ...

First-order satisfiability

The problem: Given a set of first-order sentences S check whether S is satisfiable.

Complementary to proof finding: Given a set of first-order sentences check whether it is unsatisfiable.

Where satisfiability checking is used?

- in verification for finding errors in systems
- in combinatorial reasoning: scheduling, planning, etc. for finding solutions
- in checking consistency of ontologies, theories, axiomatisations
- disproving conjectures
- ...

Methods for finite model finding

General first-order satisfiability is not recursively enumerable.

Restrict to finite model finding (FMF).

- Finite model finding is recursively enumerable.
- But usual first-order reasoning methods such resolution/superposition are incomplete for finite model finding

Methods for finite model finding are based on encodings into:

- Propositional logic (FINDER, MACE, Paradox)
  [Slaney; McCune; Claessen, Sörensson]
  Paradox has been winning satisfiability at CASC for the last 10 years.
- Geometric logic (Geo) [de Nivelle, Meng]
- Effectively propositional logic (EPR) (DarwinFM, iProver)
  [Baumgartner, de Nivelle, Fuchs, Tinelli]

Methods for finite model finding

General first-order satisfiability is not recursively enumerable.

Restrict to finite model finding (FMF).

- Finite model finding is recursively enumerable.
- But usual first-order reasoning methods such resolution/superposition are incomplete for finite model finding

Methods for finite model finding are based on encodings into:

- Propositional logic (FINDER, MACE, Paradox)
  [Slaney; McCune; Claessen, Sörensson]
  Paradox has been winning satisfiability at CASC for the last 10 years.
- Geometric logic (Geo) [de Nivelle, Meng]
- Effectively propositional logic (EPR) (DarwinFM, iProver)
  [Baumgartner, de Nivelle, Fuchs, Tinelli]

Methods for finite model finding

General first-order satisfiability is not recursively enumerable.

Restrict to finite model finding (FMF).

- Finite model finding is recursively enumerable.
- But usual first-order reasoning methods such resolution/superposition are incomplete for finite model finding

Methods for finite model finding are based on encodings into:

- Propositional logic (FINDER, MACE, Paradox) [Slaney; McCune; Claessen, Sörensson]
  Paradox has been winning satisfiability at CASC for the last 10 years.
- Geometric logic (Geo) [de Nivelle, Meng]
- Effectively propositional logic (EPR) (DarwinFM, iProver) [Baumgartner, de Nivelle, Fuchs, Tinelli]

Methods for finite model finding

General first-order satisfiability is not recursively enumerable.

Restrict to finite model finding (FMF).

- Finite model finding is recursively enumerable.
- But usual first-order reasoning methods such resolution/superposition are incomplete for finite model finding

Methods for finite model finding are based on encodings into:

- Propositional logic (FINDER, MACE, Paradox)
  [Slaney; McCune; Claessen, Sörensson]
  Paradox has been winning satisfiability at CASC for the last 10 years.
- Geometric logic (Geo) [de Nivelle, Meng]
- Effectively propositional logic (EPR) (DarwinFM, iProver)
  [Baumgartner, de Nivelle, Fuchs, Tinelli]

Effectively Propositional Logic (EPR)

EPR: No functions except constants: $P(x, y) \vee \neg Q(c, y)$

Transitivity: $\neg P(x, y) \vee \neg P(y, z) \vee P(x, z)$

Symmetry: $P(x, y) \vee \neg P(y, x)$

Verification:

$$\forall A(\mathtt{wren}_{h1} \wedge A = \mathtt{wraddrFunc} \rightarrow \forall B(\mathtt{range}_{[35,0]}(B) \rightarrow (\mathtt{imem}'(A, B) \leftrightarrow \mathtt{iwrite}(B)))).$$

Applications many problems can be encoded into the EPR:

- Hardware Verification (Intel)
- Planning/Scheduling
- Finite model finding

Instantiation-based methods excel in the EPR fragment.

Effectively Propositional Logic (EPR)

EPR: No functions except constants: $P(x, y) \vee \neg Q(c, y)$

Transitivity: $\neg P(x, y) \vee \neg P(y, z) \vee P(x, z)$

Symmetry: $P(x, y) \vee \neg P(y, x)$

Verification:

$$\forall A(\mathtt{wren}_{h1} \wedge A = \mathtt{wraddrFunc} \rightarrow \forall B(\mathtt{range}_{[35,0]}(B) \rightarrow (\mathtt{imem}'(A, B) \leftrightarrow \mathtt{iwrite}(B)))).$$

Applications many problems can be encoded into the EPR:

- Hardware Verification (Intel)
- Planning/Scheduling
- Finite model finding

Instantiation-based methods excel in the EPR fragment.

Basic idea: Eliminate functions

- $C[t] \Rightarrow t \not\simeq x \vee C[x]$
- $Q(f(g(x)))$

$$\Rightarrow \neg P_f(y_1, y_2) \vee \neg P_g(x, y_1) \vee Q(y_2)$$

Step2. Replace functions by predicates:

- $f(x_1, \ldots, x_n) \simeq y$ can be represented by $P_f(x_1, \ldots, x_n, y)$ provided:
- P_f is right-unique:

$$\forall \bar{x}, y[(P_f(\bar{x}, y) \wedge P_f(\bar{x}, y')) \rightarrow y \simeq y']$$

function-free EPR (possible to drop)

- P_f right-total:

$$\forall \bar{x} \exists y P_f(\bar{x}, y)$$

for finite domains can be expressed using domain axiom:

$$\forall \bar{x}[P_f(\bar{x}, 1) \vee \ldots \vee P_f(\bar{x}, n)]$$

EPR-based finite model finding [Baumgartner, de Nivelle, Fuchs, Tinelli]

Basic idea: Eliminate functions

Step1. Flattening: replacing complex terms by flat terms:

- $C[t] \Rightarrow t \not\simeq x \vee C[x]$
- $Q(f(g(x)))$

$$\Rightarrow \neg P_f(y_1, y_2) \vee \neg P_g(x, y_1) \vee Q(y_2)$$

Step2. Replace functions by predicates:

- $f(x_1, \ldots, x_n) \simeq y$ can be represented by $P_f(x_1, \ldots, x_n, y)$ provided:
- P_f is right-unique:

$$\forall \bar{x}, y[(P_f(\bar{x}, y) \wedge P_f(\bar{x}, y')) \rightarrow y \simeq y']$$

function-free EPR (possible to drop)

- P_f right-total:

$$\forall \bar{x} \exists y P_f(\bar{x}, y)$$

for finite domains can be expressed using domain axiom:

$$\forall \bar{x}[P_f(\bar{x}, 1) \vee \ldots \vee P_f(\bar{x}, n)]$$

Basic idea: Eliminate functions

Step1. Flattening: replacing complex terms by flat terms:

- $C[t] \Rightarrow t \not\simeq x \vee C[x]$
- $Q(f(g(x)))$

$$\Rightarrow \neg P_f(y_1, y_2) \vee \neg P_g(x, y_1) \vee Q(y_2)$$

Step2. Replace functions by predicates:

- $f(x_1, \ldots, x_n) \simeq y$ can be represented by $P_f(x_1, \ldots, x_n, y)$ provided:
- P_f is right-unique:

$$\forall \bar{x}, y[(P_f(\bar{x}, y) \wedge P_f(\bar{x}, y')) \rightarrow y \simeq y']$$

function-free EPR (possible to drop)

- P_f right-total:

$$\forall \bar{x} \exists y P_f(\bar{x}, y)$$

for finite domains can be expressed using domain axiom:

$$\forall \bar{x}[P_f(\bar{x}, 1) \vee \ldots \vee P_f(\bar{x}, n)]$$

Basic idea: Eliminate functions

Step1. Flattening: replacing complex terms by flat terms:

- $C[t] \Rightarrow t \not\simeq x \vee C[x]$
- $Q(f(g(x))) \Rightarrow g(x) \not\simeq y_1 \vee Q(f(y_1))$

$$\Rightarrow \neg P_f(y_1, y_2) \vee \neg P_g(x, y_1) \vee Q(y_2)$$

Step2. Replace functions by predicates:

- $f(x_1, \ldots, x_n) \simeq y$ can be represented by $P_f(x_1, \ldots, x_n, y)$ provided:
- P_f is right-unique:

$$\forall \bar{x}, y[(P_f(\bar{x}, y) \wedge P_f(\bar{x}, y')) \rightarrow y \simeq y']$$

function-free EPR (possible to drop)

- P_f right-total:

$$\forall \bar{x} \exists y P_f(\bar{x}, y)$$

for finite domains can be expressed using domain axiom:

$$\forall x[P_f(x, 1) \vee \ldots \vee P_f(x, n)]$$

Basic idea: Eliminate functions

Step1. Flattening: replacing complex terms by flat terms:

- $C[t] \Rightarrow t \not\simeq x \vee C[x]$
- $Q(f(g(x))) \Rightarrow g(x) \not\simeq y_1 \vee Q(f(y_1)) \Rightarrow$
  $f(y_1) \not\simeq y_2 \vee g(x) \not\simeq y_1 \vee Q(y_2) \Rightarrow \neg P_f(y_1, y_2) \vee \neg P_g(x, y_1) \vee Q(y_2)$

Step2. Replace functions by predicates:

- $f(x_1, \ldots, x_n) \simeq y$ can be represented by $P_f(x_1, \ldots, x_n, y)$ provided:
- P_f is right-unique:

$$\forall \bar{x}, y[(P_f(\bar{x}, y) \wedge P_f(\bar{x}, y')) \rightarrow y \simeq y']$$

  function-free EPR (possible to drop)

- P_f right-total:

$$\forall \bar{x} \exists y P_f(\bar{x}, y)$$

  for finite domains can be expressed using domain axiom:

$$\forall \bar{x}[P_f(\bar{x}, 1) \vee \ldots \vee P_f(\bar{x}, n)]$$

Basic idea: Eliminate functions

Step1. Flattening: replacing complex terms by flat terms:

- $C[t] \Rightarrow t \not\simeq x \vee C[x]$
- $Q(f(g(x))) \Rightarrow g(x) \not\simeq y_1 \vee Q(f(y_1)) \Rightarrow$
  $f(y_1) \not\simeq y_2 \vee g(x) \not\simeq y_1 \vee Q(y_2) \Rightarrow \neg P_f(y_1, y_2) \vee \neg P_g(x, y_1) \vee Q(y_2)$

Step2. Replace functions by predicates:

- $f(x_1, \ldots, x_n) \simeq y$ can be represented by $P_f(x_1, \ldots, x_n, y)$ provided:
- P_f is right-unique:

$$\forall \bar{x}, y[(P_f(\bar{x}, y) \wedge P_f(\bar{x}, y')) \rightarrow y \simeq y']$$

  function-free EPR (possible to drop)

- P_f right-total:

$$\forall \bar{x} \exists y P_f(\bar{x}, y)$$

  for finite domains can be expressed using domain axiom:

$$\forall \bar{x}[P_f(\bar{x}, 1) \vee \ldots \vee P_f(\bar{x}, n)]$$

Basic idea: Eliminate functions

Step1. Flattening: replacing complex terms by flat terms:

- $C[t] \Rightarrow t \not\simeq x \vee C[x]$
- $Q(f(g(x))) \Rightarrow g(x) \not\simeq y_1 \vee Q(f(y_1)) \Rightarrow$
  $f(y_1) \not\simeq y_2 \vee g(x) \not\simeq y_1 \vee Q(y_2) \Rightarrow \neg P_f(y_1, y_2) \vee \neg P_g(x, y_1) \vee Q(y_2)$

Step2. Replace functions by predicates:

- $f(x_1, \ldots, x_n) \simeq y$ can be represented by $P_f(x_1, \ldots, x_n, y)$ provided:
- P_f is right-unique:

$$\forall \bar{x}, y[(P_f(\bar{x}, y) \wedge P_f(\bar{x}, y')) \rightarrow y \simeq y']$$

function-free EPR (possible to drop)

- P_f right-total:

$$\forall \bar{x} \exists y P_f(\bar{x}, y)$$

for finite domains can be expressed using domain axiom:

$$\forall \bar{x}[P_f(\bar{x}, 1) \vee \ldots \vee P_f(\bar{x}, n)]$$

Basic idea: Eliminate functions

Step1. Flattening: replacing complex terms by flat terms:

- $C[t] \Rightarrow t \not\simeq x \vee C[x]$
- $Q(f(g(x))) \Rightarrow g(x) \not\simeq y_1 \vee Q(f(y_1)) \Rightarrow$
  $f(y_1) \not\simeq y_2 \vee g(x) \not\simeq y_1 \vee Q(y_2) \Rightarrow \neg P_f(y_1, y_2) \vee \neg P_g(x, y_1) \vee Q(y_2)$

Step2. Replace functions by predicates:

- $f(x_1, \ldots, x_n) \simeq y$ can be represented by $P_f(x_1, \ldots, x_n, y)$ provided:
- P_f is right-unique:

$$\forall \bar{x}, y[(P_f(\bar{x}, y) \wedge P_f(\bar{x}, y')) \rightarrow y \simeq y']$$

  function-free EPR (possible to drop)

- P_f right-total:

$$\forall \bar{x} \exists y P_f(\bar{x}, y)$$

  for finite domains can be expressed using domain axiom

$$\forall x[P_f(\bar{x}, 1) \vee \ldots \vee P_f(\bar{x}, n)]$$

Basic idea: Eliminate functions

Step1. Flattening: replacing complex terms by flat terms:

- $C[t] \Rightarrow t \not\simeq x \vee C[x]$
- $Q(f(g(x))) \Rightarrow g(x) \not\simeq y_1 \vee Q(f(y_1)) \Rightarrow$
  $f(y_1) \not\simeq y_2 \vee g(x) \not\simeq y_1 \vee Q(y_2) \Rightarrow \neg P_f(y_1, y_2) \vee \neg P_g(x, y_1) \vee Q(y_2)$

Step2. Replace functions by predicates:

- $f(x_1, \ldots, x_n) \simeq y$ can be represented by $P_f(x_1, \ldots, x_n, y)$ provided:
- P_f is right-unique:

$$\forall \bar{x}, y[(P_f(\bar{x}, y) \wedge P_f(\bar{x}, y')) \rightarrow y \simeq y']$$

  function-free EPR (possible to drop)

- P_f right-total:

$$\forall \bar{x} \exists y P_f(\bar{x}, y)$$

  for finite domains can be expressed using domain axiom:

$$\forall \bar{x}[P_f(\bar{x}, 1) \vee \ldots \vee P_f(\bar{x}, n)]$$

Basic idea: Eliminate functions

Step1. Flattening: replacing complex terms by flat terms:

- $C[t] \Rightarrow t \not\simeq x \vee C[x]$
- $Q(f(g(x))) \Rightarrow g(x) \not\simeq y_1 \vee Q(f(y_1)) \Rightarrow$
  $f(y_1) \not\simeq y_2 \vee g(x) \not\simeq y_1 \vee Q(y_2) \Rightarrow \neg P_f(y_1, y_2) \vee \neg P_g(x, y_1) \vee Q(y_2)$

Step2. Replace functions by predicates:

- $f(x_1, \ldots, x_n) \simeq y$ can be represented by $P_f(x_1, \ldots, x_n, y)$ provided:
- P_f is right-unique:

$$\forall \bar{x}, y[(P_f(\bar{x}, y) \wedge P_f(\bar{x}, y')) \rightarrow y \simeq y']$$

  function-free EPR (possible to drop)

- P_f right-total:

$$\forall \bar{x} \exists y P_f(\bar{x}, y)$$

  for finite domains can be expressed using domain axiom:

$$\forall \bar{x}[P_f(\bar{x}, 1) \vee \ldots \vee P_f(\bar{x}, n)]$$

Basic idea: Eliminate functions

Step1. Flattening: replacing complex terms by flat terms:

- $C[t] \Rightarrow t \not\simeq x \vee C[x]$
- $Q(f(g(x))) \Rightarrow g(x) \not\simeq y_1 \vee Q(f(y_1)) \Rightarrow$
  $f(y_1) \not\simeq y_2 \vee g(x) \not\simeq y_1 \vee Q(y_2) \Rightarrow \neg P_f(y_1, y_2) \vee \neg P_g(x, y_1) \vee Q(y_2)$

Step2. Replace functions by predicates:

- $f(x_1, \ldots, x_n) \simeq y$ can be represented by $P_f(x_1, \ldots, x_n, y)$ provided:
- P_f is right-unique:

$$\forall \bar{x}, y[(P_f(\bar{x}, y) \wedge P_f(\bar{x}, y')) \rightarrow y \simeq y']$$

  function-free EPR (possible to drop)

- P_f right-total:

$$\forall \bar{x} \exists y P_f(\bar{x}, y)$$

  for finite domains can be expressed using domain axiom:

$$\forall \bar{x}[P_f(\bar{x}, 1) \vee \ldots \vee P_f(\bar{x}, n)]$$

Is flattening always good ?

Flattening

- essential for getting minimal wrt. size models
- but flattening can be bad for performance of reasoning systems

Example: trivial propositional problem

$$
\begin{array}{ccccc}
P_1(c_1) & \neg P_1(c_2) & \neg P_1(c_3) & \ldots & \neg P_1(c_n) \\
\neg P_2(c_1) & P_2(c_2) & \neg P_2(c_3) & \ldots & \neg P_2(c_n) \\
 & & \ldots & & \\
\neg P_n(c_1) & \neg P_2(c_2) & \neg P_2(c_3) & \ldots & P_n(c_n)
\end{array}
$$

Flattening:

$$
\begin{array}{ccccc}
\neg P_{c_1}(x) \vee P_1(x) & & \ldots & & \neg P_{c_n}(x) \vee \neg P_n(x) \\
 & & \ldots & & \\
\neg P_{c_1}(x) \vee \neg P_n(x) & & \ldots & & \neg P_{c_n}(x) \vee P_n(x)
\end{array}
$$

Domain k: $P_{c_1}(1) \vee \ldots \vee P_{c_1}(k) \quad \ldots \quad P_{c_n}(1) \vee \ldots \vee P_{c_n}(k)$

Non-trivial after flattening. Next: how to avoid unnecessary flattening

Is flattening always good ?

Flattening

- essential for getting minimal wrt. size models
- but flattening can be bad for performance of reasoning systems

Example: trivial propositional problem

$$
\begin{array}{ccccc}
P_1(c_1) & \neg P_1(c_2) & \neg P_1(c_3) & \ldots & \neg P_1(c_n) \\
\neg P_2(c_1) & P_2(c_2) & \neg P_2(c_3) & \ldots & \neg P_2(c_n) \\
 & & \ldots & & \\
\neg P_n(c_1) & \neg P_2(c_2) & \neg P_2(c_3) & \ldots & P_n(c_n)
\end{array}
$$

Flattening:

$$
\begin{array}{ccc}
\neg P_{c_1}(x) \vee P_1(x) & \ldots & \neg P_{c_n}(x) \vee \neg P_n(x) \\
 & \ldots & \\
\neg P_{c_1}(x) \vee \neg P_n(x) & \ldots & \neg P_{c_n}(x) \vee P_n(x)
\end{array}
$$

Domain k: $P_{c_1}(1) \vee \ldots \vee P_{c_1}(k) \quad \ldots \quad P_{c_n}(1) \vee \ldots \vee P_{c_n}(k)$

Non-trivial after flattening. Next: how to avoid unnecessary flattening

Is flattening always good ?

Flattening

- essential for getting minimal wrt. size models
- but flattening can be bad for performance of reasoning systems

Example: trivial propositional problem

$$
\begin{array}{ccccc}
P_1(c_1) & \neg P_1(c_2) & \neg P_1(c_3) & \ldots & \neg P_1(c_n) \\
\neg P_2(c_1) & P_2(c_2) & \neg P_2(c_3) & \ldots & \neg P_2(c_n) \\
 & & \ldots & & \\
\neg P_n(c_1) & \neg P_2(c_2) & \neg P_2(c_3) & \ldots & P_n(c_n)
\end{array}
$$

Flattening:

$$
\begin{array}{ccc}
\neg P_{c_1}(x) \vee P_1(x) & \ldots & \neg P_{c_n}(x) \vee \neg P_n(x) \\
 & \ldots & \\
\neg P_{c_1}(x) \vee \neg P_n(x) & \ldots & \neg P_{c_n}(x) \vee P_n(x)
\end{array}
$$

Domain k: $P_{c_1}(1) \vee \ldots \vee P_{c_1}(k) \quad \ldots \quad P_{c_n}(1) \vee \ldots \vee P_{c_n}(k)$

Non-trivial after flattening. Next: how to avoid unnecessary flattening

Extension EPR into many-sorted logic

Observe: if a problem is EPR we do not need to apply flattening.

Can we do more?

EPR is decidable because:

- the set of all ground terms (the Herbrand universe) is finite

In unsorted first-order logic Herbrand universe is finite if and only if all function symbols are constants.

Observation: [Abadi, Rabinovich, Sagiv] In the presence of sorts the Herbrand universe can be finite even in the presence of non-constant function symbols, under certain conditions.

Extension EPR into many-sorted logic

Observe: if a problem is EPR we do not need to apply flattening.

Can we do more?

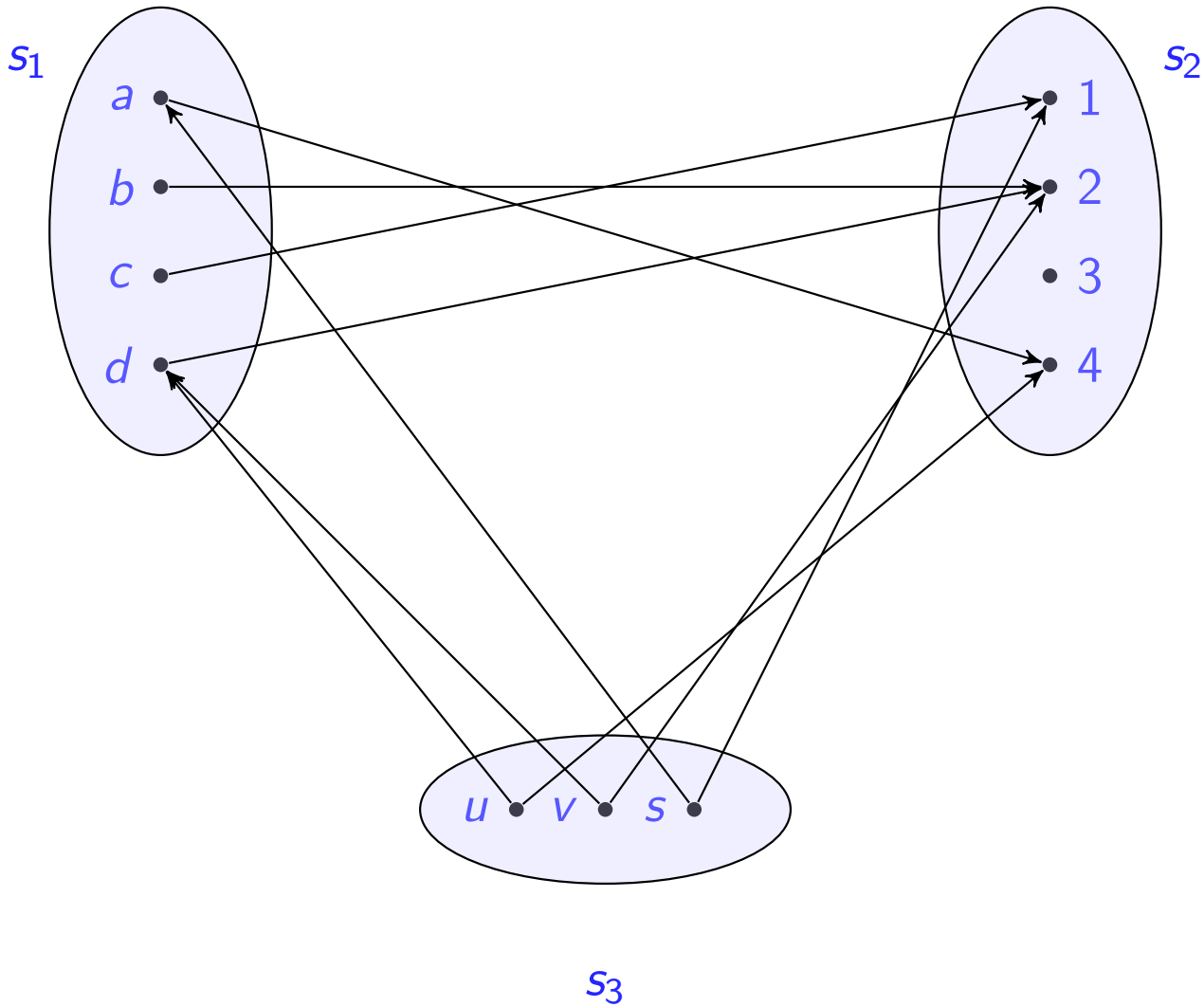
EPR is decidable because:

- the set of all ground terms (the Herbrand universe) is finite

In unsorted first-order logic Herbrand universe is finite if and only if all function symbols are constants.

Observation: [Abadi, Rabinovich, Sagiv] In the presence of sorts the Herbrand universe can be finite even in the presence of non-constant function symbols, under certain conditions.

Non-cyclic sorts



Restriction: no cyclic dependencies

Non-cyclic fragment

Consider: a signature $\Sigma = \langle \mathcal{S}, \mathcal{F}, \mathcal{P} \rangle$

- A sort dependency graph $SD(\Sigma) = \langle \mathcal{S}, \rightarrow \rangle$
  where $s_i \rightarrow s$ iff there is $f \in \mathcal{F}$ such that
  $$f : s_1 \times \ldots \times s_i \times \ldots \times s_n \mapsto s$$
- A signature is non-cyclic if there are no cycles in its sort dependency graph.
- The non-cyclic clausal fragment consists of sets of clauses over a non-cyclic signature.

Proposition. In many-sorted logic the Herbrand universe is finite if and only if the signature is non-cyclic.

Theorem. Non-cyclic fragment is decidable by instantiation based-methods.

Theorem. There is a linear-time algorithm for checking whether a clause set is in the non-cyclic fragment.

Non-cyclic fragment

Consider: a signature $\Sigma = \langle \mathcal{S}, \mathcal{F}, \mathcal{P} \rangle$

- A sort dependency graph $SD(\Sigma) = \langle \mathcal{S}, \rightarrow \rangle$
  where $s_i \rightarrow s$ iff there is $f \in \mathcal{F}$ such that
  $$f : s_1 \times \ldots \times s_i \times \ldots \times s_n \mapsto s$$
- A signature is non-cyclic if there are no cycles in its sort dependency graph.
- The non-cyclic clausal fragment consists of sets of clauses over a non-cyclic signature.

Proposition. In many-sorted logic the Herbrand universe is finite if and only if the signature is non-cyclic.

Theorem. Non-cyclic fragment is decidable by instantiation based-methods.

Theorem. There is a linear-time algorithm for checking whether a clause set is in the non-cyclic fragment.

Non-cyclic fragment

Consider: a signature $\Sigma = \langle \mathcal{S}, \mathcal{F}, \mathcal{P} \rangle$

- A sort dependency graph $SD(\Sigma) = \langle \mathcal{S}, \rightarrow \rangle$
  where $s_i \rightarrow s$ iff there is $f \in \mathcal{F}$ such that
  $$f : s_1 \times \ldots \times s_i \times \ldots \times s_n \mapsto s$$
- A signature is non-cyclic if there are no cycles in its sort dependency graph.
- The non-cyclic clausal fragment consists of sets of clauses over a non-cyclic signature.

Proposition. In many-sorted logic the Herbrand universe is finite if and only if the signature is non-cyclic.

Theorem. Non-cyclic fragment is decidable by instantiation based-methods.

Theorem. There is a linear-time algorithm for checking whether a clause set is in the non-cyclic fragment.

Non-cyclic fragment

Consider: a signature $\Sigma = \langle \mathcal{S}, \mathcal{F}, \mathcal{P} \rangle$

- A sort dependency graph $SD(\Sigma) = \langle \mathcal{S}, \rightarrow \rangle$
  where $s_i \rightarrow s$ iff there is $f \in \mathcal{F}$ such that
  $$f : s_1 \times \ldots \times s_i \times \ldots \times s_n \mapsto s$$
- A signature is non-cyclic if there are no cycles in its sort dependency graph.
- The non-cyclic clausal fragment consists of sets of clauses over a non-cyclic signature.

Proposition. In many-sorted logic the Herbrand universe is finite if and only if the signature is non-cyclic.

Theorem. Non-cyclic fragment is decidable by instantiation based-methods.

Theorem. There is a linear-time algorithm for checking whether a clause set is in the non-cyclic fragment.

Back to finite model finding

Observe: We do not need to apply flattening if the problem is in the non-cyclic fragment.

Unfortunately:

- Many problems are almost in the non-cyclic fragment.

Main idea: decompose the sort dependency graph into

- cyclic and non-cyclic sorts.

Back to finite model finding

Observe: We do not need to apply flattening if the problem is in the non-cyclic fragment.

Unfortunately:

- Many problems are almost in the non-cyclic fragment.

Main idea: decompose the sort dependency graph into

- cyclic and non-cyclic sorts.

Back to finite model finding

Observe: We do not need to apply flattening if the problem is in the non-cyclic fragment.

Unfortunately:

- Many problems are almost in the non-cyclic fragment.

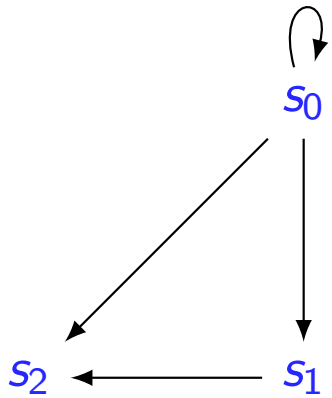
Main idea: decompose the sort dependency graph into

- cyclic and non-cyclic sorts.

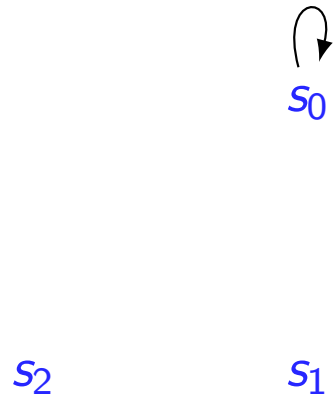
Sort decomposition

Non-cyclic decomposition: Decompose all sorts into cyclic and non-cyclic.

Sort dependency graph



Non-cyclic decomposition



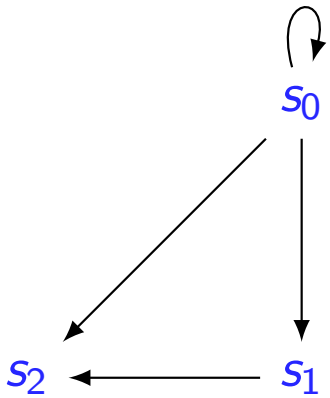
Theorem There is a linear-time algorithm for non-cyclic decomposition.

The proof is based on Tarjan's algorithm for decomposing directed graphs into strongly connected components.

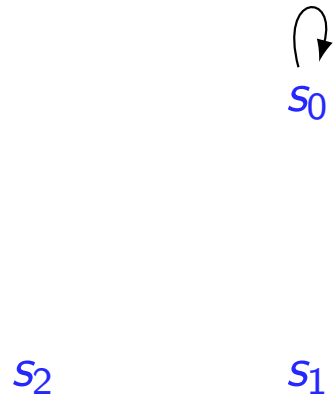
Sort decomposition

Non-cyclic decomposition: Decompose all sorts into cyclic and non-cyclic.

Sort dependency graph



Non-cyclic decomposition



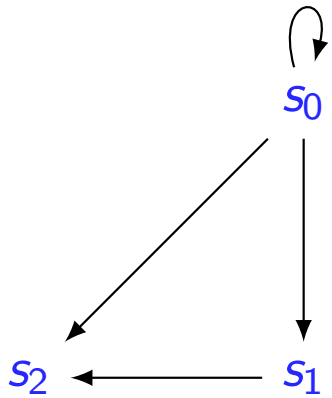
Theorem There is a linear-time algorithm for non-cyclic decomposition.

The proof is based on Tarjan's algorithm for decomposing directed graphs into strongly connected components.

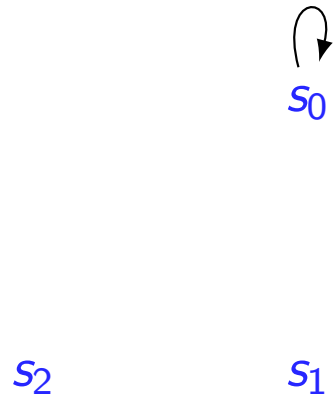
Sort decomposition

Non-cyclic decomposition: Decompose all sorts into cyclic and non-cyclic.

Sort dependency graph



Non-cyclic decomposition



Theorem There is a linear-time algorithm for non-cyclic decomposition.

The proof is based on Tarjan's algorithm for decomposing directed graphs into strongly connected components.

Sort-restricted finite model finding

Consider:

- A set of clauses S over a signature Σ and
- the non-cyclic decomposition of Σ.

Sort-restricted finite model finding:

- restrict flattening to terms of cyclic sorts,
- apply instantiation-based methods to the obtained non-cyclic clauses

Theorem. Sort-restricted finite-model finding is complete for finite satisfiability.

Sort-restricted finite model finding

Consider:

- A set of clauses S over a signature Σ and
- the non-cyclic decomposition of Σ.

Sort-restricted finite model finding:

- restrict flattening to terms of cyclic sorts,
- apply instantiation-based methods to the obtained non-cyclic clauses

Theorem. Sort-restricted finite-model finding is complete for finite satisfiability.

Sort-restricted finite model finding

Note: Cyclic and non-cyclic sorts can be interleaved inside a term.

Consider: a signature with function symbols f, g, h, c with one cyclic sort: $\mathit{valSort}(g) = \mathit{argSort}(1, g) = \mathit{argSort}(h, 1) = s$

$$f(x, h(g(z, h(x)))) \simeq f(x, c) \vee h(x) \simeq c$$

Sort-restricted finite model finding

Note: Cyclic and non-cyclic sorts can be interleaved inside a term.

Consider: a signature with function symbols f, g, h, c with one cyclic sort: $\mathit{valSort}(g) = \mathit{argSort}(1, g) = \mathit{argSort}(h, 1) = s$

$$f(x, h(g(z, h(x)))) \simeq f(x, c) \vee h(x) \simeq c \quad \Rightarrow$$

SR Flattening : $\quad g(z, h(x)) \not\simeq y \vee f(x, h(y)) \simeq f(x, c_1) \vee h(x) \simeq c_1$

Sort-restricted finite model finding

Note: Cyclic and non-cyclic sorts can be interleaved inside a term.

Consider: a signature with function symbols f, g, h, c with one cyclic sort: $valSort(g) = argSort(1, g) = argSort(h, 1) = s$

$$f(x, h(g(z, h(x)))) \simeq f(x, c) \vee h(x) \simeq c \quad \Rightarrow$$

SR Flattening : $g(z, h(x)) \not\simeq y \vee f(x, h(y)) \simeq f(x, c_1) \vee h(x) \simeq c_1 \quad \Rightarrow$

$$\neg P_g(z, h(x), y) \vee f(x, h(y)) \simeq f(x, c_1) \vee h(x) \simeq c_1$$

Sort-restricted finite model finding

Note: Cyclic and non-cyclic sorts can be interleaved inside a term.

Consider: a signature with function symbols f, g, h, c with one cyclic sort: $\mathit{valSort}(g) = \mathit{argSort}(1, g) = \mathit{argSort}(h, 1) = s$

$$f(x, h(g(z, h(x)))) \simeq f(x, c) \vee h(x) \simeq c \quad \Rightarrow$$

SR Flattening : $g(z, h(x)) \not\simeq y \vee f(x, h(y)) \simeq f(x, c_1) \vee h(x) \simeq c_1 \quad \Rightarrow$

$$\neg P_g(z, h(x), y) \vee f(x, h(y)) \simeq f(x, c_1) \vee h(x) \simeq c_1$$

Domain : $P_g(x, y, 1) \vee \ldots \vee P_g(x, y, n)$

Sort-restricted vs EPR transformation

Initial: $f(x, h(g(z, h(x)))) \simeq f(x, c) \vee h(x) \simeq c$

Sort restricted transformation:

$$\neg P_g(z, h(x), y) \vee f(x, h(y)) \simeq f(x, c_1) \vee h(x) \simeq c_1$$

$$P_g(x, y, 1) \vee \ldots \vee P_g(x, y, n)$$

EPR-transformation:

$$\neg P_h(x, y_0) \vee \neg P_g(z, y_0, y_1) \vee \neg P_h(y_1, y_2) \vee \neg P_{c_1}(y_3) \vee$$
$$\neg P_f(x, y_2, y_4) \vee \neg P_f(x, y_3, y_5) \vee \neg P_h(x, y_6)$$
$$\vee$$
$$y_4 \simeq y_5 \vee y_6 \simeq y_3$$
$$P_g(x_0, 1) \vee \ldots \vee P_g(x_0, n)$$
$$P_h(x_0, 1) \vee \ldots \vee P_h(x_0, n)$$
$$P_{c_1}(1) \vee \ldots \vee P_{c_1}(n)$$
$$P_f(x_0, x_1, 1) \vee \ldots \vee P_f(x_0, x_1, n).$$

Sort-restricted vs EPR transformation

Initial: $f(x, h(g(z, h(x)))) \simeq f(x, c) \vee h(x) \simeq c$

Sort restricted transformation:

$$\neg P_g(z, h(x), y) \vee f(x, h(y)) \simeq f(x, c_1) \vee h(x) \simeq c_1$$

$$P_g(x, y, 1) \vee \ldots \vee P_g(x, y, n)$$

EPR-transformation:

$$\neg P_h(x, y_0) \vee \neg P_g(z, y_0, y_1) \vee \neg P_h(y_1, y_2) \vee \neg P_{c_1}(y_3) \vee$$
$$\neg P_f(x, y_2, y_4) \vee \neg P_f(x, y_3, y_5) \vee \neg P_h(x, y_6)$$
$$\vee$$
$$y_4 \simeq y_5 \vee y_6 \simeq y_3$$
$$P_g(x_0, 1) \vee \ldots \vee P_g(x_0, n)$$
$$P_h(x_0, 1) \vee \ldots \vee P_h(x_0, n)$$
$$P_{c_1}(1) \vee \ldots \vee P_{c_1}(n)$$
$$P_f(x_0, x_1, 1) \vee \ldots \vee P_f(x_0, x_1, n).$$

TPTP benchmarks

TPTP benchmarks: $15,550$ first-order problems.

Unfortunately: all these problems are unsorted.

Solution:

- infer sorts automatically (linear-time)

  [Claessen, Sörensson; Claessen, Lilliestrom, Smallbone]
- $4,090$ problems have more than one inferred sort.

Non-cyclic/EPR sorts in TPTP

- $1,383$ pure EPR problems
- $2,578$ problems have at least one non-cyclic sort
- $56,679$ collective number of sorts
- $9,569$ EPR sorts
- $18,502$ non-cyclic sorts
- most problems combine non-cyclic/cyclic/EPR sorts

Summary: Non-cyclic sorts are in more than half of sortified problems.

TPTP benchmarks

TPTP benchmarks: $15,550$ first-order problems.

Unfortunately: all these problems are unsorted.

Solution:

- infer sorts automatically (linear-time)

  [Claessen, Sörensson; Claessen, Lilliestrom, Smallbone]

- $4,090$ problems have more than one inferred sort.

Non-cyclic/EPR sorts in TPTP

- $1,383$ pure EPR problems
- $2,578$ problems have at least one non-cyclic sort
- $56,679$ collective number of sorts
- $9,569$ EPR sorts
- $18,502$ non-cyclic sorts
- most problems combine non-cyclic/cyclic/EPR sorts

Summary: Non-cyclic sorts are in more than half of sortified problems.

TPTP benchmarks

TPTP benchmarks: 15,550 first-order problems.

Unfortunately: all these problems are unsorted.

Solution:

- infer sorts automatically (linear-time)

  [Claessen, Sörensson; Claessen, Lilliestrom, Smallbone]

- 4,090 problems have more than one inferred sort.

Non-cyclic/EPR sorts in TPTP

- 1,383 pure EPR problems
- 2,578 problems have at least one non-cyclic sort
- 56,679 collective number of sorts
- 9,569 EPR sorts
- 18,502 non-cyclic sorts
- most problems combine non-cyclic/cyclic/EPR sorts

Summary: Non-cyclic sorts are in more than half of sortified problems.

TPTP benchmarks

TPTP benchmarks: $15,550$ first-order problems.

Unfortunately: all these problems are unsorted.

Solution:

- infer sorts automatically (linear-time)

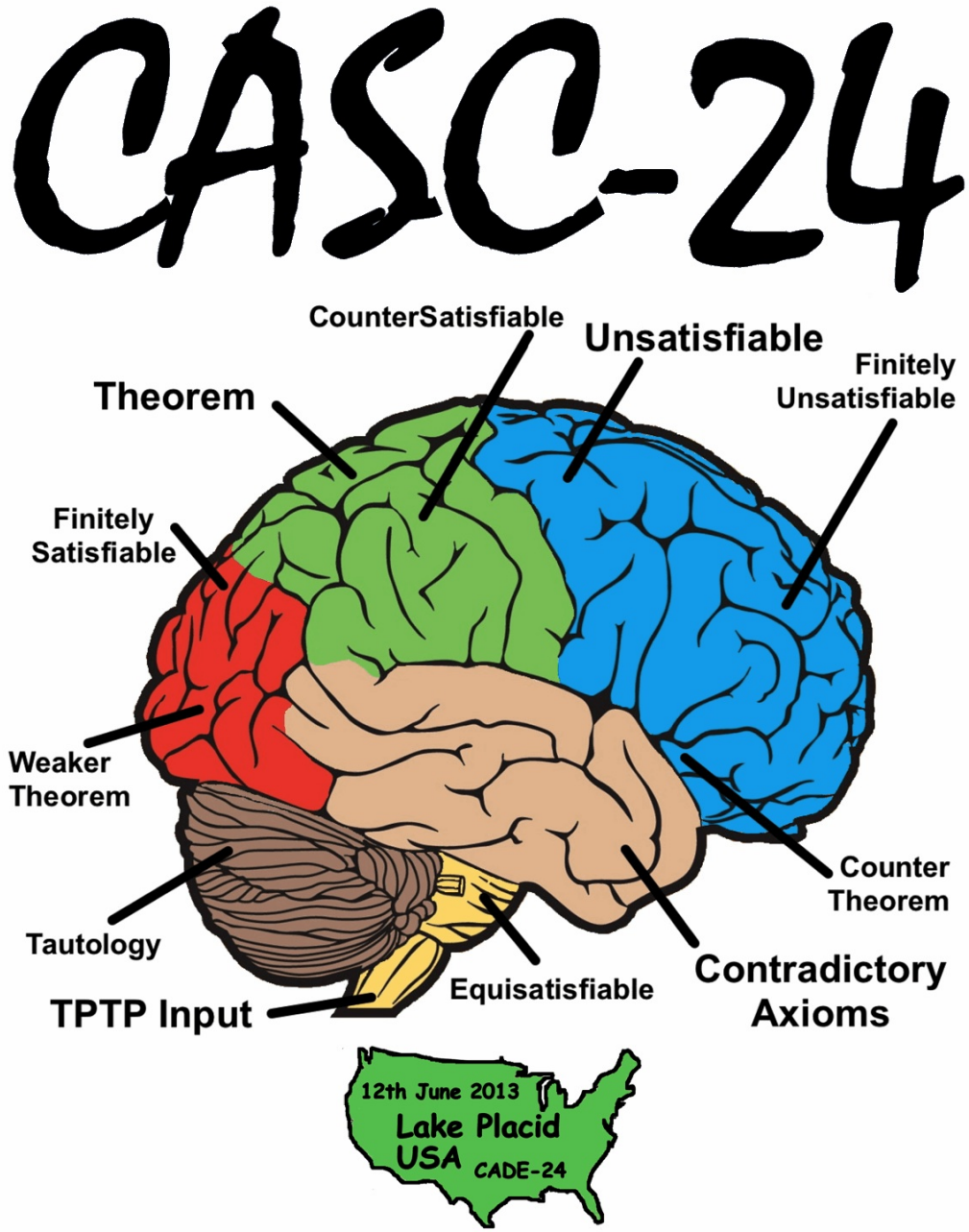
  [Claessen, Sörensson; Claessen, Lilliestrom, Smallbone]

- $4,090$ problems have more than one inferred sort.

Non-cyclic/EPR sorts in TPTP

- $1,383$ pure EPR problems
- $2,578$ problems have at least one non-cyclic sort
- $56,679$ collective number of sorts
- $9,569$ EPR sorts
- $18,502$ non-cyclic sorts
- most problems combine non-cyclic/cyclic/EPR sorts

Summary: Non-cyclic sorts are in more than half of sortified problems.


CASC-24
CounterSatisfiable
Unsatisfiable
Finitely Unsatisfiable
Theorem
Finitely Satisfiable
Weaker Theorem
Counter Theorem
Tautology
Equisatisfiable
Contradictory Axioms
TPTP Input
12th June 2013
Lake Placid
USA
CADE-24

CASC 2013

iProver – an instantiation-based reasoner for first-order logic

- based on the Inst-Gen calculus
- modular combination of first-order reasoning with MiniSAT
- redundancy elimination, indexing, ...
- implemented in OCaml
- sort-restricted finite model finding, symmetry reduction

First-order satisfiability (FNT) 150 problems

	iProver	Paradox	CVC4	E	Nitrox	Vampire
prob	122	99	96	79	79	78
time	52	2	25	20	29	30

- For the first time in 10 years the reign of Paradox has been successfully challenged...
- Paradox is still... a paradox – very efficient finite model finder.

CASC 2013

iProver – an instantiation-based reasoner for first-order logic

- based on the Inst-Gen calculus
- modular combination of first-order reasoning with MiniSAT
- redundancy elimination, indexing, ...
- implemented in OCaml
- sort-restricted finite model finding, symmetry reduction

First-order satisfiability (FNT) 150 problems

	iProver	Paradox	CVC4	E	Nitrox	Vampire
prob	122	99	96	79	79	78
time	52	2	25	20	29	30

- For the first time in 10 years the reign of Paradox has been successfully challenged...
- Paradox is still.... a paradox – very efficient finite model finder.

CASC 2013

iProver – an instantiation-based reasoner for first-order logic

- based on the Inst-Gen calculus
- modular combination of first-order reasoning with MiniSAT
- redundancy elimination, indexing, ...
- implemented in OCaml
- sort-restricted finite model finding, symmetry reduction

First-order satisfiability (FNT) 150 problems

	iProver	Paradox	CVC4	E	Nitrox	Vampire
prob	122	99	96	79	79	78
time	52	2	25	20	29	30

- For the first time in 10 years the reign of Paradox has been successfully challenged...
- Paradox is still.... a paradox – very efficient finite model finder.

Summary

Summary

- The non-cyclic fragment is decidable by instantiation-based methods.
- Non-cyclic sort decomposition in linear-time.
- Sort-restricted flattening and finite model finding.
- More than half of sortified problems in TPTP contain non-cyclic sorts.
- Instantiation + sort-restricted finite model finding is a winning combination.

Future

- There is flexibility which sorts to flatten, what is the best way?
- Can we gain from non-cyclic sorts in theorem proving ?
- Combination on non-cyclic fragment with other fragments/theories.
- Integration into iProver-Eq.

Summary

Summary

- The non-cyclic fragment is decidable by instantiation-based methods.
- Non-cyclic sort decomposition in linear-time.
- Sort-restricted flattening and finite model finding.
- More than half of sortified problems in TPTP contain non-cyclic sorts.
- Instantiation + sort-restricted finite model finding is a winning combination.

Future

- There is flexibility which sorts to flatten, what is the best way?
- Can we gain from non-cyclic sorts in theorem proving ?
- Combination on non-cyclic fragment with other fragments/theories.
- Integration into iProver-Eq.