Constraint systems for proof-search modulo a theory

Damien Rouhling

ENS Lyon

Joint work with M. Farooque, S. Graham-Lengrand, A. Mahboubi and J.-M. Notin

Context

- Automated proof-search
- Modulo theories
- PSYCHE
- Quantifiers handling

Sequent calculus modulo a theory

- A theory as parameter
- Some predicate/function symbols are interpreted by the axioms of the theory
  $\exists x.\,(P\,(x+1) \Rightarrow P\,(2))$
- Focused sequent calculus for polarised logic, without quantifiers:

$$\frac{}{\Gamma \vdash^{\mathcal{P},p} [p]}\ \Gamma_{lit} \models_{\mathcal{T}} p$$

$$\frac{}{\Gamma \vdash^{\mathcal{P}}}\ \Gamma_{lit} \models_{\mathcal{T}}$$

PSYCHE

- Proof Search factorY for Collaborative HEuristics

PSYCHE

- Proof Search factorY for Collaborative HEuristics
- Modular platform: kernel-plugins-decision procedures interaction
- The theory is implemented as a decision procedure checking the consistency of a set of literals, used at the leaves of the proof-tree
- Produces proof objects

Outline

1. First order proof-search
   - Delaying the instantiation of variables
   - How to close branches
2. Constraint systems
   - Constraint-producing system
   - Constraint-refining system

Outline

1. First order proof-search
   - Delaying the instantiation of variables
   - How to close branches
2. Constraint systems
   - Constraint-producing system
   - Constraint-refining system

First order rules

$$\frac{\Gamma \vdash A}{\Gamma \vdash \forall x.A}\; x \notin FV(\Gamma) \qquad\qquad \frac{\Gamma \vdash A[t/x]}{\Gamma \vdash \exists x.A}$$

First order rules

$$\frac{\Gamma \vdash A}{\Gamma \vdash \forall x.A}\ x \notin FV(\Gamma) \qquad \frac{\Gamma \vdash A[t/x]}{\Gamma \vdash \exists x.A}$$

Eigen- and meta-variables:

$$\frac{\Gamma \vdash^{l}_{n+1} A[x := e_{n+1}]}{\Gamma \vdash^{l}_{n} \forall x.A} \qquad \frac{\Gamma \vdash^{n::l}_{n} A\left[x := ?_{|l|+1}\right]}{\Gamma \vdash^{l}_{n} \exists x.A}$$

Example: the drinker paradox

$$\vdash_{0}^{[]} \exists x.P(x) \Rightarrow \forall y.P(y)$$

Example: the drinker paradox

$$\frac{\vdash_0^{[0]} P(?_1) \Rightarrow \forall y.P(y)}{\vdash_0^{[]} \exists x.P(x) \Rightarrow \forall y.P(y)}$$

Example: the drinker paradox

$$\dfrac{\dfrac{P\left(?_1\right) \vdash_0^{[0]} \forall y.P\left(y\right)}{\vdash_0^{[0]} P\left(?_1\right) \Rightarrow \forall y.P\left(y\right)}}{\vdash_0^{[]} \exists x.P\left(x\right) \Rightarrow \forall y.P\left(y\right)}$$

Example: the drinker paradox

$$\dfrac{\dfrac{\dfrac{P\,(?_1) \vdash_1^{[0]} P\,(e_1)}{P\,(?_1) \vdash_0^{[0]} \forall y.P\,(y)}}{\vdash_0^{[0]} P\,(?_1) \Rightarrow \forall y.P\,(y)}}{\vdash_0^{[]} \exists x.P\,(x) \Rightarrow \forall y.P\,(y)}$$

The instantiation $?_1 := e_1$ is forbidden: $?_1$ may not depend on any eigenvariable

Pure first order: closing branches with unification constraints

Idea: use unification to find an adapted instantiation of the meta-variables

Pure first order: closing branches with unification constraints

Idea: use unification to find an adapted instantiation of the meta-variables

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{P(?_2) \vdash_1^{[1,1]} P(f(?_1))}{\vdash_1^{[1,1]} P(?_2) \Rightarrow P(f(?_1))} \qquad \dfrac{P(f(e_1)) \vdash_1^{[1,1]} P(?_1)}{\vdash_1^{[1,1]} P(f(e_1)) \Rightarrow P(?_1)}}{\vdash_1^{[1,1]} (P(?_2) \Rightarrow P(f(?_1))) \wedge (P(f(e_1)) \Rightarrow P(?_1))}}{\vdash_1^{[1]} \exists y.((P(y) \Rightarrow P(f(?_1))) \wedge (P(f(e_1)) \Rightarrow P(?_1)))}}{\vdash_1^{[]} \exists x.\exists y.((P(y) \Rightarrow P(f(x))) \wedge (P(f(e_1)) \Rightarrow P(x)))}}{\vdash_0^{[]} \forall z.\exists x.\exists y.((P(y) \Rightarrow P(f(x))) \wedge (P(f(z)) \Rightarrow P(x)))}$$

$\sigma = [?_1 \mapsto f(e_1), ?_2 \mapsto f(f(e_1))]$ closes the branches

With a theory: closing branches with theory-specific constraints

Refinement: deal with theory-specific constraints in the mean time using an abstract constraint structure

With a theory: closing branches with theory-specific constraints

Refinement: deal with theory-specific constraints in the mean time using an abstract constraint structure

- Constraints have a domain: they are local to a branch
- A meta-variable can be shared by several branches
- We want to propagate and combine constraints
- Our goal: get a satisfiable constraint at the root of the tree
- A possibility: backtracking

Abstract constraint structures

Definition

A constraint structure is:

- a family of sets $(\Psi_l)_l$: the elements of Ψ_l are the constraints of domain l (the meta-variables with their dependencies)
- a family of projections from $\Psi_{n::l}$ to Ψ_l, denoted by $.\downarrow$

Abstract constraint structures

Definition

A constraint structure is:

- a family of sets $(\Psi_I)_I$: the elements of Ψ_I are the constraints of domain I (the meta-variables with their dependencies)
- a family of projections from $\Psi_{n::I}$ to Ψ_I, denoted by $\cdot_\downarrow$

Example (pure first order):
Ψ_I is the set of maps of domain I assigning a term to each meta-variable respecting the dependencies between the variables
Most general unifiers allow to combine two constraints

Outline

1. First order proof-search
   - Delaying the instantiation of variables
   - How to close branches
2. Constraint systems
   - Constraint-producing system
   - Constraint-refining system

Branching

Two possibilities:

- Explore the two branches in parallel and combine the constraints they produce
  $\Rightarrow$ constraint-producing system
- The constraint produced by a branch might direct the exploration of the other one: sequentialize
  $\Rightarrow$ constraint-refining system

Constraint producing system: meet constraint structures

The constraint structure is refined with a (family of) meet operator(s): $(\sigma, \sigma') \mapsto \sigma \wedge \sigma'$ on Ψ_I

Constraint producing system: meet constraint structures

The constraint structure is refined with a (family of) meet operator(s): $(\sigma, \sigma') \mapsto \sigma \wedge \sigma'$ on Ψ_l

$$\frac{}{\vdash^l_n \Gamma \to \sigma} \models^l \Gamma_{lit} \to \sigma$$

$$\frac{\vdash^l_n \Gamma, A \to \sigma_1 \qquad \vdash^l_n \Gamma, B \to \sigma_2}{\vdash^l_n \Gamma, A \wedge B \to \sigma_1 \wedge \sigma_2}$$

Example

$$\dfrac{\dfrac{\dfrac{\vdash_0^{[0,0]} ?_2 < 2?_1 \qquad \vdash_0^{[0,0]} ?_1 > 3 \qquad \vdash_0^{[0,0]} ?_1 < 6}{\vdash_0^{[0,0]} (?y < 2?x) \wedge (?x > 3) \wedge (?x < 6)}}{\vdash_0^{[0]} \exists y. ((y < 2?x) \wedge (?x > 3) \wedge (?x < 6))}}{\vdash_0^{[]} \exists x. \exists y. ((y < 2x) \wedge (x > 3) \wedge (x < 6))}$$

Example

$$\dfrac{\dfrac{\dfrac{\vdash_0^{[0,0]} ?_2 < 2?_1 \to \sigma_0 \qquad \vdash_0^{[0,0]} ?_1 > 3 \to \sigma_1 \qquad \vdash_0^{[0,0]} ?_1 < 6 \to \sigma_2}{\vdash_0^{[0,0]} (?y < 2?x) \wedge (?x > 3) \wedge (?x < 6)}}{\vdash_0^{[0]} \exists y.((y < 2?x) \wedge (?x > 3) \wedge (?x < 6))}}{\vdash_0^{[]} \exists x.\exists y.((y < 2x) \wedge (x > 3) \wedge (x < 6))}$$

$\sigma_0 = (?_2 \in]-\infty, 2?_1[)$, $\sigma_1 = (?_1 \in]3, +\infty[)$ and $\sigma_2 = (?_1 \in]-\infty, 6[)$

Example

$$\dfrac{\dfrac{\dfrac{\vdash_0^{[0,0]} ?_2 < 2?_1 \to \sigma_0 \qquad \vdash_0^{[0,0]} ?_1 > 3 \to \sigma_1 \qquad \vdash_0^{[0,0]} ?_1 < 6 \to \sigma_2}{\vdash_0^{[0,0]} (?y < 2?x) \wedge (?x > 3) \wedge (?x < 6) \ \to \sigma_0 \wedge \sigma_1 \wedge \sigma_2 = \sigma}}{\vdash_0^{[0]} \exists y.((y < 2?x) \wedge (?x > 3) \wedge (?x < 6))}}{\vdash_0^{[]} \exists x.\exists y.((y < 2x) \wedge (x > 3) \wedge (x < 6))}$$

$\sigma_0 = (?_2 \in]-\infty, 2?_1[)$, $\sigma_1 = (?_1 \in]3, +\infty[)$ and $\sigma_2 = (?_1 \in]-\infty, 6[)$
$\sigma = (?_1 \in \{4, 5\}, ?_2 \in]-\infty, 2?_1[)$

Example

$$\dfrac{\dfrac{\dfrac{\vdash_0^{[0,0]} ?_2 < 2?_1 \to \sigma_0 \qquad \vdash_0^{[0,0]} ?_1 > 3 \to \sigma_1 \qquad \vdash_0^{[0,0]} ?_1 < 6 \to \sigma_2}{\vdash_0^{[0,0]} (?y < 2?x) \wedge (?x > 3) \wedge (?x < 6) \ \to \sigma_0 \wedge \sigma_1 \wedge \sigma_2 = \sigma}}{\vdash_0^{[0]} \exists y. ((y < 2?x) \wedge (?x > 3) \wedge (?x < 6)) \to \sigma_\downarrow}}{\vdash_0^{[]} \exists x. \exists y. ((y < 2x) \wedge (x > 3) \wedge (x < 6)) \to (\sigma_\downarrow)_\downarrow}$$

$\sigma_0 = (?_2 \in]-\infty, 2?_1[)$, $\sigma_1 = (?_1 \in]3, +\infty[)$ and $\sigma_2 = (?_1 \in]-\infty, 6[)$
$\sigma = (?_1 \in \{4, 5\}, ?_2 \in]-\infty, 2?_1[)$
$\sigma_\downarrow = (?_1 \in \{4, 5\}), (\sigma_\downarrow)_\downarrow = \emptyset$

Constraint-refining system: lift constraint structures

The constraint structure is refined with a (family of) lift operator(s):
$\sigma \mapsto \sigma^{\uparrow}$ on Ψ_I

Constraint-refining system: lift constraint structures

The constraint structure is refined with a (family of) lift operator(s):
$\sigma \mapsto \sigma^{\uparrow}$ on Ψ_l

$$\frac{}{\sigma \rightarrow\vdash^l_n \Gamma \rightarrow \sigma'}\ \sigma \rightarrow\models^l \Gamma_{lit} \rightarrow \sigma'$$

$$\frac{\sigma \rightarrow\vdash^l_n \Gamma, A_i \rightarrow \sigma_0 \qquad \sigma_0 \rightarrow\vdash^l_n \Gamma, A_{1-i} \rightarrow \sigma'}{\sigma \rightarrow\vdash^l_n \Gamma, A_0 \wedge A_1 \rightarrow \sigma'}\ i \in \{0,1\}$$

A word on satisfiability

We want to prove the soundness and completeness of the constraint-producing (resp. refining) system w.r.t. the system without delayed instantiation.

In particular, we want the minimal properties on the constraint structure giving us these equivalences.

A tool: compatibility relations between instantiations and constraints.

A word on satisfiability

- $T_n = \{$ground terms whose eigenvariables are below $n\}$
  It is extented to domains: an instantiation on domain I is an element of T_I
- Compatibility relation between $\rho \in T_I$ and $\sigma \in \Psi_I$: $\rho \epsilon \sigma$ such that
  - $(t :: \rho)\, \epsilon \sigma \Rightarrow \rho \epsilon \sigma_{\downarrow}$
  - $\rho \epsilon \sigma \wedge \sigma' \Leftrightarrow \rho \epsilon \sigma$ and $\rho \epsilon \sigma'$
- σ is satisfiable if we can find ρ such that $\rho \epsilon \sigma$

Implementation

- OCaml module for constraint structures in PSYCHE
- A top constraint (always satisfiable) is required to start the proof-search
- Backtracking implies the production of a stream at the leaves
- Only the empty theory (pure first order) has been implemented in this framework

Conclusion

- Constraint structures allow delayed instantiations
- Sufficient (minimal) axiomatisation to prove soundness and completeness
- Backtracking and streams open the doors to subtle strategies in proof-search
- Still has to be tested