Deduction Modulo

Claude Kirchner

LORIA & INRIA
Nancy
France

Joint work with
Eric Deplagne, Gilles Dowek and Thérèse Hardin

Deduction versus computation

COMPUTATION

COMPUTATION

Calculi were there before mathematics

COMPUTATION

Calculi were there before mathematics 

Calculi were there before mathematics 

At the beginning of the 20th century

some believed that everything can be computed...

Calculi were there before mathematics 

At the beginning of the 20th century

some believed that everything can be computed...

... but

Deduction

Deduction

undecidable problems exists

Deduction

undecidable problems exists

and deduction becomes the dominant view point...

Deduction

undecidable problems exists

and deduction becomes the dominant view point...

... but

Deduction

undecidable problems exists

and deduction becomes the dominant view point...

... but

do we want to deduce $2 + 2 = 4$ from basic Peano axioms?

Deduction

undecidable problems exists

and deduction becomes the dominant view point...

... but

do we want to deduce $2 + 2 = 4$ from basic Peano axioms?

do we want to compute $2 + 2$ into 4?

Deduction

undecidable problems exists

and deduction becomes the dominant view point...

... but

do we want to deduce $2 + 2 = 4$ from basic Peano axioms?

do we want to compute $2 + 2$ into 4?

do you believe the last check you got at the supermarket?

Deduction versus computation

Deduction versus computation

Let's try to get the best of both concepts

Deduction versus computation

Let's try to get the best of both concepts

How can that be formalized...

Deduction versus computation

Let's try to get the best of both concepts

How can that be formalized...

... and used?

Roadmap

- Introduction to deduction modulo
- Proof search for deduction modulo
- Deduction modulo for higher-order logic
- Deduction modulo as a programming paradigm
- Deduction modulo and induction
- Conclusions

What is deduction modulo?

- specific presentation of first-order logic
- that works $modulo$ a congruence on term and propositions
- and that makes a clear distinction between *computation* and *deduction*

Reasoning Modulo

$$\frac{A \Rightarrow B \quad A}{B}$$

Reasoning Modulo

$$\frac{A \Rightarrow B \quad A}{B}$$

$$\frac{A' \Rightarrow B \quad A}{B} \text{ if } A \equiv A'$$

Reasoning Modulo

$$\frac{A \Rightarrow B \quad A}{B}$$

$$\frac{A' \Rightarrow B \quad A}{B} \text{ if } A \equiv A'$$

$$\frac{C \quad A}{B} \text{ if } C \equiv A \Rightarrow B$$

Reasoning versus Computing...

Do we want to *prove* that $2+2 = 4$

or

Do we want to *check* it via a computation: $2 + 2 \rightarrow 4$?

... leads to Deduction modulo Computation

$Computations$ can be considered as $much\ simpler$ than $deduction$

$\rightarrow$ Poincaré principle

Indeed abstractly we want to reason modulo a congruence that could be defined by computation rules, or by other means.

– For example 2+2 and 3+1 and 4 and 4+0 are all congruent modulo the definition of addition,

– $(x+y)+z,\ \ y+(x+z),\ \ (y+z)+x, \ldots$ are congruent modulo the associativity and commutativity of $+$.

The 2+2 example

Look at a proof of $\exists y\ 2+y=4$:

$$\cfrac{\cfrac{\overline{4=4\vdash 2+2=4}\ \text{axiom}}{\forall x\ x=x\vdash 2+2=4}\ (x, x=x, 4)\ \forall\text{-l}}{\forall x\ x=x\vdash \exists y\ 2+y=4}\ (y, 2+y=4, 2)\ \exists\text{-r}$$

The 2+2 example

Look at a proof of $\exists y\ 2 + y = 4$:

$$\dfrac{\dfrac{\overline{4 = 4 \vdash 2 + 2 = 4}\ \text{axiom}}{\forall x\ x = x \vdash 2 + 2 = 4}\ (x, x = x, 4)\ \forall\text{-l}}{\forall x\ x = x \vdash \exists y\ 2 + y = 4}\ (y, 2 + y = 4, 2)\ \exists\text{-r}$$

The computational argument 2+2 $=$ 4 is left out of the proof.

The 2+2 example

Look at a proof of $\exists y\ 2+y=4$:

$$\cfrac{\cfrac{\overline{4=4 \vdash 2+2=4}\ \text{axiom}}{\forall x\ x=x \vdash 2+2=4}\ (x, x=x, 4)\ \forall\text{-l}}{\forall x\ x=x \vdash \exists y\ 2+y=4}\ (y, 2+y=4, 2)\ \exists\text{-r}$$

The computational argument 2+2 = 4 is left out of the proof.

In this case, the congruence is defined by a rewriting system defined on *terms*

$$\begin{array}{rcl} 0+y & \rightarrow & y \\ s(x)+y & \rightarrow & s(x+y) \end{array}$$

A proof of $2+2=4$ using this definition of addition can of course be done and consists in the equational proof:

$$2+2 = s(s(0)) + s(s(0)) = s(s(0) + s(s(0))) = s(s(0 + s(s(0)))) = s(s(s(s(0)))) = 4$$

Congruences on terms

Andrews 71: equivalence modulo $\beta\eta$

operationally handled by $\beta\eta$-reduction

mechanization of deduction requires HO-unification

Plotkin 72: $\forall x\ \forall y\ \forall z\ ((x + y) + z = x + (y + z))$

operationally handled by rewriting

mechanization of deduction requires Associative-unification

Peterson& Stickel 81, **Huet 80**: E-completion

requires E-unification

Stickel 85: Theory resolution

requires deciding validity of certain formulas

Meseguer 89: logic $=$ deduction rule $+$ congruence

JouannaudKirchner, Dershowitz, Bachmair, Vigneron, . . .

We need more!

We need more!

$$x * y = 0 \;\Leftrightarrow\; x = 0 \vee y = 0$$

We need more!

$$x * y = 0 \;\Leftrightarrow\; x = 0 \vee y = 0$$

$$x * y = x * z \;\Leftrightarrow\; y = z \vee x = 0$$

We need more!

$$x * y = 0 \;\Leftrightarrow\; x = 0 \vee y = 0$$

$$x * y = x * z \;\Leftrightarrow\; y = z \vee x = 0$$

$$x \in \{y, z\} \Leftrightarrow (x = y) \vee (x = z)$$

We need more!

$$x * y = 0 \;\Leftrightarrow\; x = 0 \vee y = 0$$

$$x * y = x * z \;\Leftrightarrow\; y = z \vee x = 0$$

$$x \in \{y, z\} \Leftrightarrow (x = y) \vee (x = z)$$

$$x \in \mathcal{P}(y) \Leftrightarrow \forall z\ (z \in x \Rightarrow z \in y)$$

A congruence on propositions

It is useful to identify not only

Terms $$x + y \equiv y + x$$

but also

Propositions $$x \in \mathcal{P}(y) \equiv \forall z \ (z \in x \Rightarrow z \in y)$$

Definition of the congruence

Conditional class rewrite systems

- $\mathcal{R}$ a set of conditional rewrite rules rewriting:
  - Atomic Proposition into Proposition
    $$x * y = 0 \rightarrow_{\mathcal{R}} x = 0 \vee y = 0$$
  - Term into Term
    $$x + 0 \rightarrow_{\mathcal{R}} x$$
- $\mathcal{E}$ a set of conditional equational axioms equating:
  - Atomic Proposition to Atomic Proposition
    $$(x = y) =_{\mathcal{E}} (y = x)$$
  - Term to Term
    $$(x * (y * z)) =_{\mathcal{E}} ((x * y) * z)$$
    $$(x * y) =_{\mathcal{E}} (y * x)$$
- $\mathcal{RE}$ a conditional class rewrite system $\rightarrow^{\Gamma}_{\mathcal{RE}}$

Sequent Calculus Modulo

$$\frac{\Gamma, P \vdash_{\mathcal{RE}} \Delta \quad \Gamma \vdash_{\mathcal{RE}} Q, \Delta}{\Gamma \vdash_{\mathcal{RE}} \Delta}\textsf{cut} \quad \text{if } P =^{\Gamma}_{\mathcal{RE}} Q$$

$$\frac{\Gamma, Q_1, Q_2 \vdash_{\mathcal{RE}} \Delta}{\Gamma, P \vdash_{\mathcal{RE}} \Delta}\textsf{contr-l} \quad \text{if } P =^{\Gamma}_{\mathcal{RE}} Q_1 =^{\Gamma}_{\mathcal{RE}} Q_2$$

Compatibility
A duality between computation and deduction

A set of axioms $\mathcal{T}$ (used for deduction)
and
a conditional class rewrite system $\mathcal{RE}$ (used for computation)

are *compatible* when:

- for all propositions P and Q
  $P =_{\mathcal{RE}} Q$ implies $\mathcal{T} \vdash P \Leftrightarrow Q$
- for every proposition P in $\mathcal{T}$, we have $\vdash_{\mathcal{RE}} P$.

Canonical theory

We associate to $\mathcal{RE}$ the canonical theory $T_{\mathcal{RE}}$:

- proposition conditional rule or equation
  $(p \rightarrow q \text{ if } c)$ or $(p \approx q \text{ if } c) \in \mathcal{RE}$
  $\forall \bar{x}(c \Rightarrow (p \Leftrightarrow q)) \in T_{\mathcal{RE}}$

- term conditional rule or equation
  $(g \rightarrow d \text{ if } c)$ or $(g \approx d \text{ if } c) \in \mathcal{RE}$
  $\forall \bar{x}(c \Rightarrow (g \approx d)) \in T_{\mathcal{RE}}$

- [DHK] $T_{\mathcal{RE}}$ and $\mathcal{RE}$ are compatible

Deduction versus deduction modulo
or
The Poincaré Principle

If the theory $\mathcal{T}$ and the conditional class rewrite system $\mathcal{RE}$ are compatible then we have [DHK]:

$$\Gamma, \mathcal{T} \vdash \Delta \text{ if and only if } \Gamma \vdash_{\mathcal{RE}} \Delta$$

Deduction modulo is modular

Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be two theories, compatible respectively with the conditional class rewrite systems $\mathcal{RE}_{\mathcal{T}_1}$ and $\mathcal{RE}_{\mathcal{T}_2}$

$$\begin{array}{ccccc} & & \Gamma \vdash_{\mathcal{RE}_{\mathcal{T}_1 \cup \mathcal{T}_2}} \Delta & & \\ & & \Updownarrow & & \\ \mathcal{T}_2, \Gamma \vdash_{\mathcal{RE}_{\mathcal{T}_1}} \Delta & \Leftrightarrow & \mathcal{T}_1, \mathcal{T}_2, \Gamma \vdash \Delta & \Leftrightarrow & \mathcal{T}_1, \Gamma \vdash_{\mathcal{RE}_{\mathcal{T}_2}} \Delta \\ & & \Updownarrow & & \\ & & \Gamma \vdash_{\mathcal{RE}_{\mathcal{T}_1} \cup \mathcal{RE}_{\mathcal{T}_2}} \Delta & & \end{array}$$

Let's use it!

Seeking for proofs

Example: Integral domains

$$\forall x, y \quad x * y = 0 \Leftrightarrow x = 0 \vee y = 0$$

$$\mathcal{R} : x * y = 0 \rightarrow x = 0 \vee y = 0$$

How to prove: (A) $\exists z (a * a = z \Rightarrow a = z)$?

$$\dfrac{\dfrac{\dfrac{a = 0 \vdash_{\mathcal{RE}} a = 0 \quad a = 0 \vdash_{\mathcal{RE}} a = 0}{\boxed{\begin{array}{c} a = 0 \vee a = 0 \vdash_{\mathcal{RE}} a = 0 \\ =_{\mathcal{RE}} \\ a * a = 0 \vdash_{\mathcal{RE}} a = 0 \end{array}}} \vee\text{-l}}{\vdash_{\mathcal{RE}} a * a = 0 \Rightarrow a = 0} \Rightarrow\text{-r}}{\vdash_{\mathcal{RE}} \exists z (a * a = z \Rightarrow a = z)} \exists\text{-r}$$

Example: Integral domains, continuing

How to prove by resolution: (A) $\exists z(a * a = z \Rightarrow a = z)$?

The clausal form of $\neg A$ is:

$a * a = z \qquad \neg(a = z)$

And there is NO resolution step possible, why?. . .

Example: Integral domains, continuing

How to prove by resolution: (A) $\exists z(a * a = z \Rightarrow a = z)$?

The clausal form of $\neg A$ is:
$a * a = z \qquad \neg(a = z)$
And there is NO resolution step possible, why?. . .

Because with the standard resolution rule, there is no way to use the now "built-in" information that $\forall x, y \quad x * y = 0 \Leftrightarrow x = 0 \vee y = 0$

Of course we can put everything in a big bag and apply resolution blindly, but the main idea of deduction modulo is different and we would like to use rewrite rules as such.

Theorem Proving Modulo

Mechanize the discovery of a proof of $A_1, \ldots, A_n \vdash B_1, \ldots, B_m$

ConvenientRepresentationOf$(A_1, \ldots, A_n \vdash B_1, \ldots, B_m)$
$\longmapsto$
$\ldots$
$\longmapsto$
Evidence of the proof

Extension of the resolution method
[Robinson 1965], [Stickel 1985], [Kirchner^{2}Rusinowitch 1990], ...

Clauses with constraints

Given an equational theory $\mathcal{E}$

$t =^?_{\mathcal{E}} t'$: equation modulo $\mathcal{E}$ i.e. pair of terms or atomic propositions
σ is a $\mathcal{E}$-solution of $t =^?_{\mathcal{E}} t'$ when $\sigma(t) =_{\mathcal{E}} \sigma(t')$

A constrained clause is a pair $C\,[E]$ such that:

- C is a clause
- E is a set of equations.

It schematizes $\{\sigma(C) | \sigma \in \text{Solution}(E)\}$

▶ Allows for a clear separation between the "built-in" theory and the rest of the deduction system,

▶ Permits to postpone the problem of constraint *solving*,
  - undecidable unification problems in theories of main interest (HOL)
  - Solving could be much more complex than just testing satisfiability

▶ Additional benefits: sharing, modularity, clever strategies, ...

Example of explicit constraint handling goodies

Example of explicit constraint handling goodies

1. To make visible every symbolic computation step: unification, orientation, typing.

Example of explicit constraint handling goodies

1. To make visible every symbolic computation step: unification, orientation, typing.

2. To schematize (infinitely) many objects.
   $(x * y)/y \to x$ [$y \neq 0$]
   $f(g(x)) \to g(x)$ [$x \in \{g^n(f(a)), n \geq 0\}$]
   $f(x)$ [$x + a =^?_A a + x$]

Example of explicit constraint handling goodies

1. To make visible every symbolic computation step:
   unification, orientation, typing.

2. To schematize (infinitely) many objects.
   $(x * y)/y \to x \quad [y \neq 0]$
   $f(g(x)) \to g(x) \quad [x \in \{g^n(f(a)), n \geq 0\}]$
   $f(x) \quad [x + a =^?_A a + x]$

3. To take into account structure sharing.
   $f(x, x, x, x) \quad [x = \text{bigterm}]$

Example of explicit constraint handling goodies

1. To make visible every symbolic computation step:
   unification, orientation, typing.

2. To schematize (infinitely) many objects.
   $(x * y)/y \rightarrow x \quad [y \neq 0]$
   $f(g(x)) \rightarrow g(x) \quad [x \in \{g^n(f(a)), n \geq 0\}]$
   $f(x) \quad [x + a =^?_A a + x]$

3. To take into account structure sharing.
   $f(x, x, x, x) \quad [x = \text{bigterm}]$

4. To take advantage of constraint accumulation:
   $f(x, x, x, x) =^? f(u, v, w, z)$ has 34 359 607 481 minimal AC-solutions.

5. $$\begin{array}{ll} f(x,x,x,x) & =^? f(u,v,w,z) \\ f(u,u,u,u) & =^? f(x,v,w,z) \\ f(v,v,v,v) & =^? f(u,x,w,z) \\ f(w,w,w,w) & =^? f(u,v,x,z) \\ f(z,z,z,z) & =^? f(u,v,w,x) \\ g(y,y,y,y) & =^? g(z,p,r,z) \\ g(z,z,z,z) & =^? g(y,p,r,s) \\ g(p,p,p,p) & =^? g(z,y,r,s) \\ g(r,r,r,r) & =^? g(z,p,y,s) \\ g(s,s,s,s) & =^? g(z,p,r,y) \end{array}$$

has one minimal AC-solution.

Extended Narrowing and Resolution

$$ExtendedResolution \quad \frac{\{P_1,\ldots,P_n,Q_1,\ldots,Q_m\}\,[\mathcal{C}_1] \quad \{\neg R_1,\ldots,\neg R_p,S_1,\ldots,S_q\}\,[\mathcal{C}_2]}{\{Q_1,\ldots,Q_m,S_1,\ldots,S_q\}\,[\mathcal{C}_1\cup\mathcal{C}_2\cup\{P_1 =^?_{\mathcal{E}} \ldots =^?_{\mathcal{E}} P_n =^?_{\mathcal{E}} R_1 \ldots =^?_{\mathcal{E}} R_p\}]}$$

$$ExtendedNarrowing \quad \frac{U\,[\mathcal{C}]}{U'\,[\mathcal{C}\cup\{U_{|\omega} =^?_{\mathcal{E}} l\}]} \text{ if } l\rightarrow r\in\mathcal{R},\ U_{|\omega} \text{ atom. prop. and } U'\in c\ell(\{U[r]_\omega\})$$

Exemple: Back to integral domains

$\mathcal{R} = \{x * y = 0 \rightarrow x = 0 \vee y = 0\}$

How to prove (A): $\exists z(a * a = z \Rightarrow a = z)$?

The clausal form of $\neg A$ is: $a * a = z \qquad \neg(a = z)$

We can narrow $a * a = z$ using $\mathcal{R}$ and we get:

$$c\ell(x = 0 \vee y = 0\,[\![(a * a = z) =^? (x * y = 0)]\!]) \qquad \neg(a = z)$$

$$x = 0,\ y = 0\,[\![(a * a = z) =^? (x * y = 0)]\!] \qquad \neg(a = z)$$

Applying resolution we have furthermore:

$$y = 0\,[\![(x = 0) =^? (a = z) \wedge (a * a = z) =^? (x * y = 0)]\!]$$

And once again:

$$\Box\,[\![(y = 0) =^? (a = z) \wedge (x = 0) =^? (a = z) \wedge (a * a = z) =^? (x * y = 0)]\!]$$

And check satisfiability of the constraint:

$$y =^? a \ \wedge\ 0 =^? z \ \wedge\ x =^? a \ \wedge\ 0 =^? z \ \wedge\ a =^? x \ \wedge\ a =^? y \ \wedge\ z =^? 0$$

QED

Theorem Proving Modulo: Main Results

Let $\mathcal{RE}$ be a conluent class rewrite system

Then, for all propositions $A_1, \ldots, A_n, B_1, \ldots, B_m$:

- If
$$c\ell(\{\{A_1\}, \ldots, \{A_n\}, \{\neg B_1\}, \ldots, \{\neg B_m\}\})\,[\emptyset] \;\rightsquigarrow_{\mathcal{RE}}\; \Box\,[E]$$
where E is an $\mathcal{E}$-unifiable set of equations then
$$A_1, \ldots, A_n \vdash_{\mathcal{RE}} B_1, \ldots, B_m$$
is derivable.

- If
$$A_1, \ldots, A_n \vdash_{\mathcal{RE}} B_1, \ldots, B_m$$
has a cut free proof then
$$c\ell(\{\{A_1\}, \ldots, \{A_n\}, \{\neg B_1\}, \ldots, \{\neg B_m\}\})\,[\emptyset] \;\rightsquigarrow_{\mathcal{RE}}\; \Box\,[E]$$
where E is an $\mathcal{E}$-unifiable set of equations.

Corollary

When the cut rule is redundant in the sequent calculus modulo $\mathcal{RE}$ then:

$$\mathcal{T}_{\mathcal{RE}}, A_1, \ldots, A_n \vdash B_1, \ldots, B_m$$
$$\Leftrightarrow$$
$$A_1, \ldots, A_n \vdash_{\mathcal{RE}} B_1, \ldots, B_m$$
$$\Leftrightarrow$$
$$c\ell(\{\{A_1\}, \ldots, \{A_n\}, \{\neg B_1\}, \ldots, \{\neg B_m\}\})\,\texttt{[}\emptyset\texttt{]} \;\leadsto\; \Box\,\texttt{[}E\texttt{]}$$

where E is an $\mathcal{E}$-unifiable set of equations.

Corollary

When the cut rule is redundant in the sequent calculus modulo $\mathcal{RE}$ then:

$$\mathcal{T}_{\mathcal{RE}}, A_1, \ldots, A_n \vdash B_1, \ldots, B_m$$
$$\Leftrightarrow$$
$$A_1, \ldots, A_n \vdash_{\mathcal{RE}} B_1, \ldots, B_m$$
$$\Leftrightarrow$$
$$c\ell(\{\{A_1\}, \ldots, \{A_n\}, \{\neg B_1\}, \ldots, \{\neg B_m\}\})\,[\emptyset] \;\leadsto\; \Box\,[E]$$

where E is an $\mathcal{E}$-unifiable set of equations.

Syntactically based (elaborated) proof [JAR2003-04]

When is the cut rule redundant in the sequent calculus modulo $\mathcal{RE}$?

This has been studied in [Dowek & Werner, 98]

Sufficient conditions is to have $\mathcal{RE}$ confluent, terminating and either:

– without quantifier introduction

– without negative occurrence of literals in right-hand sides of rules

But also theories like first order presentation of HOL either based on combinatory logic or on theories of explicit substitutions.

Example: A simple theorem on sets

Prove: $A : \forall x(x \cap x = x)$

The clausal form of the negation of A is: $\neg a \cap a = a$

The following definitions of predicates:

$$\begin{array}{lcl} x = y & \Leftrightarrow & x \subseteq y \wedge y \subseteq x \\ x \subseteq y & \Leftrightarrow & \forall z(z \in x \Rightarrow z \in y) \\ z \in x \cap y & \Leftrightarrow & z \in x \wedge z \in y \end{array}$$

are used in an oriented way:

$$\begin{array}{lcl} x = y & \longrightarrow & x \subseteq y \wedge y \subseteq x \\ x \subseteq y & \longrightarrow & \forall z(z \in x \Rightarrow z \in y) \\ z \in x \cap y & \longrightarrow & z \in x \wedge z \in y \end{array}$$

We are in a trivial case where the Narrowing rule is restricted to rewriting. Its repeated application yields the set of ground clauses:

$b \in a, c \in a$ $\qquad$ $\neg(b \in a), c \in a$ $\qquad$ $b \in a, \neg(c \in a)$ $\qquad$ $\neg(b \in a), \neg(c \in a)$

which could be trivially resolved into the empty clause.

Example: Sets again

Just adding the following definitions of union ($\cup$) and powerset ($\mathcal{P}$):

$$\begin{aligned} x \in \mathcal{P}(y) &\longrightarrow x \subseteq y \\ z \in x \cup y &\longrightarrow z \in x \vee z \in y \end{aligned}$$

For the proof of: $$\mathcal{P}(x \cap y) = \mathcal{P}(x) \cap \mathcal{P}(y)$$

following [Plaisted and Zhu, 99], no proof were found by Otter after 1000s and 27656 generated clauses and it tooks 11s to their prover (OSHL) to prove it.

- Confirms that "removing" computational proofs makes a big difference.

Applications of the proof search modulo

- To automated first-order theorem proving
  - Implementation of TPM
  - Extensions (equational, ...)
- To higher-order theorem proving: HOL-$\lambda\sigma$ [DHK-RTA99]

Application of deduction modulo

Deduction Modulo as

- a programming paradigm: ELAN
- a proof representation paradigm: CoC modulo

Application to higher-order resolution

An important theory modulo: $HOL_{\lambda\sigma}$

- syntax:
  - $\lambda\sigma$-terms with constants for the logical symbols
  - unique predicate ε
- congruence: class rewrite system $\lambda\sigma\mathcal{L}$
  - rewrite rules of $\lambda\sigma$-calculus
  - logical rules $\mathcal{L}$
- cut elimination holds in $HOL_{\lambda\sigma}$

The link with Higher Order Logic: $HOL_{\lambda\sigma}$ and HOL_{λ}

Pre-cooking: translation by F of a λ-term into a $\lambda\sigma$-term

$HOL_{\lambda\sigma}$ is intentionally equivalent to HOL_{λ}:

$$p_1, \ldots, p_n \vdash_{HOL_\lambda} q_1, ..., q_m$$
$$\Leftrightarrow$$
$$\varepsilon(p_{1F}), \ldots, \varepsilon(p_{nF}) \vdash_{HOL_{\lambda\sigma}} \varepsilon(q_{1F}), ..., \varepsilon(q_{nF})$$

Higher order resolution (i.e. resolution and splitting in HOL_{λ})

is equivalent to

First order resolution and narrowing modulo in $HOL_{\lambda\sigma}$

[DHK-RTA99]

ELAN: THE DEDUCTION MODULO PARADIGM AT WORK

A short look at ELAN

rewrite rewrite rewrite rewrite rewrite rewrite rewrite rewrite

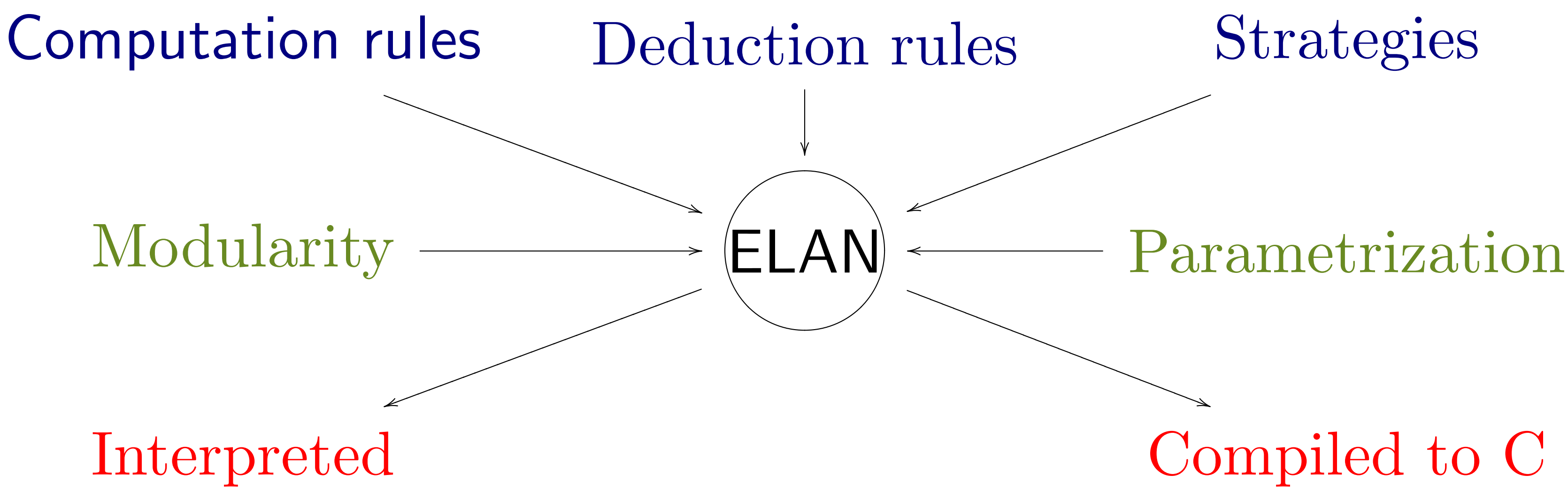
Logic Programming, Theorem Proving, Constraint Solving are instances of the same deduction schema:

Apply rewrite rules (may be modulo) on formulas with some strategy, until getting specific forms

- Rewrite blindly: implements computations
- Rewrite wisely: implements deduction

ELAN= computation rules + (deduction rules + strategies)

ELAN: **Language concepts and features**



Deductions and computations in ELAN

- Rules for computations:
  unique normal form required
  leftmost innermost strategy fixed

- Rules for deductions:
  no confluence nor termination required
  application strategy required

- Derivation tree exploration:
  strategies to express choices

Example 1: Pure computation

```
module fib_builtin
import global builtinInt;
  end

operators global
        fib(@)  : (builtinInt) builtinInt ;
  end

rules for builtinInt
        n : builtinInt ;
global
    [] fib(0) => 1 end
    [] fib(1) => 1 end
    [] fib(n) => fib(n - 1) + fib(n - 2) if greater_builtinInt(n,1) end
  end
end
```

fib(33) = 5702887   11405773 rewrite steps in 0.695 s   16.411.184 rewrite/s

Digital 500/500, 128Mo

Example 2: Implement a deduction mechanism for the propositional sequent calculus

$$\frac{H, P \vdash Q}{H \vdash \neg P, Q}\text{neg-r}$$

```
rules for Seq
        P, Q, R : Pred;   H : Pred;   S1, S2 : Seq;
global
  [neg-r]   H |- ^P : Q   => S1
                             where S1 := (dedstrat) H : P |- Q
  end

...

  [axio]   P : H |- P : R  =>  $
                               if neq_Pred(P, EmptyP)
```

The true code

Built (for later use) the proof term :

```
[neg-r]  H |- ^P : Q   =>  [#neg-r,H |- ^P : Q] <> S1
                          where S1 := (dedstrat) H : P |- Q
end
```

Strategies

```
strategies  for Seq
implicit

        [] SetRules  =>  dc one(
                                axio
                                ,neg-r ,disj-r
                                ,imp-r ,neg-l ,conj-l
                                ,disj-l ,conj-r ,imp-l)

strategies  for Seq
implicit
        [] dedstrat  =>  dc one( Start );
                         repeat*( SetRules )
```

The resulting proof term

```
[dedstrat](A |=> B |- ^(B) |=> ^(A))
```

evaluates to:

```
#infer[#impd]<(A#to B)#vdash(#neg(B)#to#neg(A))>
<#infer[#negd]<(A#to B),#neg(B)#vdash#neg(A)>
<#infer[#negg]<A,(A#to B),#neg(B)#vdash EmptyP>
<#infer[#impg]<A,(A#to B)#vdash B>
<#infer[#axiom]<A,B#vdash B><#mbox<>>&
#infer[#axiom]<A#vdash A,B><#mbox<>>>>>>
end
```

Deduction Modulo is a realistic programming paradigm

On typical applications, ELAN applies per second:

- 10 millions of non-labeled rules
- 1 million of labeled rules
- 100 000 of AC rules

On real size applications:

- Jobs shop 10x10: 2 billions rewrite applied in 2 hours

More on elan

Just get it at

`www.loria.fr/ELAN`

and use it to deduce modulo.

ELAN implements the deduction modulo paradigm

but

what about its semantics?

The rewriting calculus

Induction and deduction modulo

skip

Example 1

- signature

$$\begin{array}{rcrcl} 0 & : & & \longrightarrow & Nat \\ s\_ & : & Nat & \longrightarrow & Nat \\ \\ \_+\_ & : & Nat \times Nat & \longrightarrow & Nat \end{array}$$

Example 1

- signature

$$
\begin{array}{rcrcl}
0 & : & & \rightarrow & Nat \\
s\_ & : & Nat & \rightarrow & Nat \\
\\
\_+\_ & : & Nat \times Nat & \rightarrow & Nat
\end{array}
$$

- rules

$$
\begin{array}{rcl}
x + 0 & \rightarrow & x \\
x + s(y) & \rightarrow & s(x + y)
\end{array}
$$

Example 1

- goal: $0 + x = x$

Example 1

- goal: $0 + x = x$
- base case:

$$0 + 0 = 0 \quad \rightarrow \quad 0 = 0$$

Example 1

- goal: $0 + x = x$
- base case:

$$0 + 0 = 0 \quad \rightarrow \quad 0 = 0$$

- step case:

$$\begin{array}{rcl} 0 + s(y) = s(y) & \rightarrow & s(0 + y) = s(y) \\ & \rightarrow & s(y) = s(y) \end{array}$$

Example 2

- signature

$$
\begin{array}{rcccl}
0 & : & & \rightarrow & Nat \\
s\_ & : & Nat & \rightarrow & Nat \\
 & & & & \\
true & : & & \rightarrow & Bool \\
false & : & & \rightarrow & Bool \\
 & & & & \\
even\_ & : & Nat & \rightarrow & Bool \\
odd\_ & : & Nat & \rightarrow & Bool \\
 & & & & \\
pair\_ & : & Nat & \rightarrow & Bool
\end{array}
$$

Example 2

- rules

$$
\begin{array}{rcl}
even(0) & \rightarrow & true \\
even(s(x)) & \rightarrow & odd(x) \\
\\
odd(0) & \rightarrow & false \\
odd(s(x)) & \rightarrow & even(x) \\
\\
pair(0) & \rightarrow & true \\
pair(s(0)) & \rightarrow & false \\
pair(s(s(x))) & \rightarrow & pair(x)
\end{array}
$$

Example 2

- goal: $even(x) = pair(x)$

Example 2

- goal: $even(x) = pair(x)$
- base case 1:

$$\begin{aligned} even(0) = pair(0) &\rightarrow even(0) = true \\ &\rightarrow true = true \end{aligned}$$

Example 2

- goal: $even(x) = pair(x)$
- base case 1:

$$\begin{array}{rcr} even(0) = pair(0) & \rightarrow & even(0) = true \\ & \rightarrow & true = true \end{array}$$

- base case 2:

$$\begin{array}{rcr} even(s(0)) = pair(s(0)) & \rightarrow & even(s(0)) = false \\ & \rightarrow & odd(0) = false \\ & \rightarrow & false = false \end{array}$$

Example 2

- step case:

$$\begin{array}{rcl} even(s(s(x))) = pair(s(s(x))) & \longrightarrow & even(s(s(x))) = pair(x) \\ & \longrightarrow & even(s(s(x))) = even(x) \\ & \longrightarrow & odd(s(x)) = even(x) \\ & \longrightarrow & even(x) = even(x) \end{array}$$

Example 3

- signature

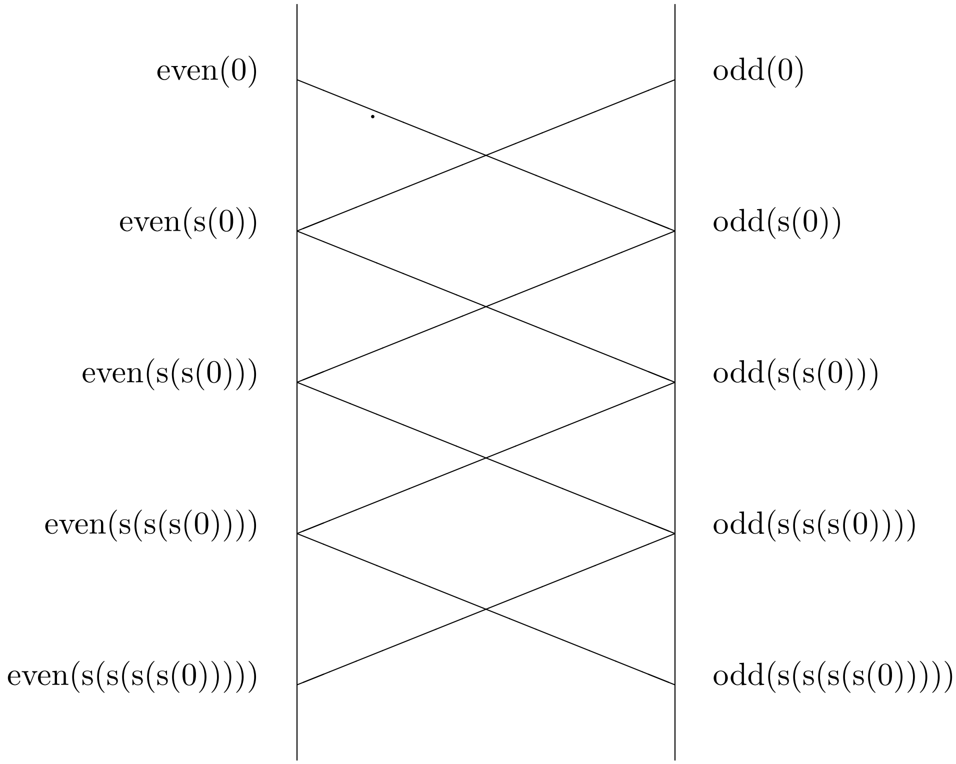
$$
\begin{array}{rcccl}
0 & : & & \rightarrow & Nat \\
s\_ & : & Nat & \rightarrow & Nat \\
\\
true & : & & \rightarrow & Bool \\
false & : & & \rightarrow & Bool \\
\\
even\_ & : & Nat & \rightarrow & Bool \\
odd\_ & : & Nat & \rightarrow & Bool \\
\\
neg\_ & : & Bool & \rightarrow & Bool
\end{array}
$$

Example 3

- rules

$$
\begin{array}{rcl}
even(0) & \rightarrow & true \\
even(s(x)) & \rightarrow & odd(x) \\
 & & \\
odd(0) & \rightarrow & false \\
odd(s(x)) & \rightarrow & even(x) \\
 & & \\
neg(true) & \rightarrow & false \\
neg(false) & \rightarrow & true
\end{array}
$$

Mutual recursivity



Example 3

- goal: $even(x) = neg(odd(x))$

Example 3

- goal: $even(x) = neg(odd(x))$
- base case 1:

$$\begin{aligned} even(0) = neg(odd(0)) &\rightarrow & even(0) = neg(false) \\ &\rightarrow & even(0) = true \\ &\rightarrow & true = true \end{aligned}$$

Example 3

- goal: $even(x) = neg(odd(x))$
- base case 1:

$$\begin{aligned} even(0) = neg(odd(0)) &\rightarrow & even(0) = neg(false) \\ &\rightarrow & even(0) = true \\ &\rightarrow & true = true \end{aligned}$$

- base case 2:

$$\begin{aligned} even(s(0)) = neg(odd(s(0))) &\rightarrow & even(s(0)) = neg(even(0)) \\ &\rightarrow & even(s(0)) = neg(true) \\ &\rightarrow & even(s(0)) = false \\ &\rightarrow & odd(0) = false \\ &\rightarrow & false = false \end{aligned}$$

Example 3

- step case:

$$
\begin{array}{rcl}
even(s(s(x))) = neg(odd(s(s(x)))) & \rightarrow & even(s(s(x))) = neg(even(s(x))) \\
 & \rightarrow & even(s(s(x))) = neg(odd(x)) \\
 & \rightarrow & even(s(s(x))) = even(x) \\
 & \rightarrow & odd(s(x)) = even(x) \\
 & \rightarrow & even(x) = even(x)
\end{array}
$$

On these examples, INDUCTION is implemented by computation

Indeed induction is (mainly) performed by rewriting

This method is implemented for example in SPIKE or RRL

On these examples, INDUCTION is implemented by computation

Indeed induction is (mainly) performed by rewriting

This method is implemented for example in SPIKE or RRL

First aim of this talk: How and why does it work?

Explicit versus implicit induction

Explicit versus implicit induction

- Explicit induction: proof assistants (Coq, . . .)

Explicit versus implicit induction

- Explicit induction: proof assistants (Coq, . . .)
- Implicit induction by first-order rewriting (Spike, RRL)

Explicit versus implicit induction

- Explicit induction: proof assistants (Coq, . . .)
- Implicit induction by first-order rewriting (Spike, RRL)
- Link between the two approaches?

The noetherian induction axiom

R is a well-founded relation on τ

$$
\begin{array}{l}
NoethInd(R,\tau) \\
= \\
\forall P \\
(\\
\quad \forall x((x \in \tau \wedge \forall \underline{y}((\underline{y} \in \tau \wedge \alpha(\alpha(R,x),\underline{y})) \Rightarrow P(\underline{y}))) \Rightarrow P(x)) \\
\quad\; \Rightarrow \\
\quad \forall x(x \in \tau \Rightarrow P(x)) \\
)
\end{array}
$$

Inductive reasoning

Inductive consequence

$$\forall R \forall \tau (Noeth(R, \tau) \Rightarrow NoethInd(R, \tau)), Noeth(R, \tau), Th_u \vdash Q$$

Inductive reasoning

Inductive consequence

$$\forall R \forall \tau (Noeth(R,\tau) \Rightarrow NoethInd(R,\tau)), Noeth(R,\tau), Th_u \vdash Q$$

Inductive property

$$\mathcal{H}erbrand(Th_u) \models Q$$

Inductive reasoning

Inductive consequence

$$\forall R \forall \tau (Noeth(R, \tau) \Rightarrow NoethInd(R, \tau)), Noeth(R, \tau), Th_u \vdash Q$$

Inductive property

$$\mathcal{H}erbrand(Th_u) \models Q$$

Inductive consequence $\Rightarrow$ Inductive property

expressing $NoethInd(R, \tau)$: Step 1

Standard expression of $NoethInd(R, \tau)$ in HOL_λ

$$
\begin{array}{l}
\dot{\forall}\lambda P \\
(\\
\quad \dot{\forall}\lambda x((x \in \tau \mathbin{\dot{\wedge}} \dot{\forall}\lambda \underline{y}((\underline{y} \in \tau \mathbin{\dot{\wedge}} R(x, \underline{y})) \mathbin{\dot{\Rightarrow}} P(\underline{y}))) \mathbin{\dot{\Rightarrow}} P(x)) \\
\quad \dot{\Rightarrow} \\
\quad \dot{\forall}\lambda x(x \in \tau \mathbin{\dot{\Rightarrow}} P(x)) \\
)
\end{array}
$$

expressing $NoethInd(R, \tau)$: Step 2

Expression of $NoethInd(R, \tau)$ in $HOL_{\lambda\sigma}$

$$
\begin{array}{l}
\forall P \\
(\\
\quad \forall x((\varepsilon(x \in \tau) \wedge \forall \underline{y}((\varepsilon(\underline{y} \in \tau) \wedge \varepsilon(\alpha(\alpha(R, x), \underline{y}))) \Rightarrow \varepsilon(\alpha(P, \underline{y})))) \Rightarrow \varepsilon(\alpha(P, x))) \\
\quad\; \Rightarrow \\
\quad \forall x \, (\varepsilon(x \in \tau) \Rightarrow \varepsilon(\alpha(P, x))) \\
)
\end{array}
$$

expressing $NoethInd(R, \tau)$: Step $2'$

Just forget $\varepsilon' s$ and $\alpha' s$

$$
\begin{array}{l}
\forall \tau \forall P \\
(\\
\quad \forall x((x \in \tau \wedge \forall \underline{y}((\underline{y} \in \tau \wedge R(x, \underline{y})) \Rightarrow P(\underline{y}))) \Rightarrow P(x)) \\
\quad\quad \Rightarrow \\
\quad \forall x \, (x \in \tau \Rightarrow P(x)) \\
)
\end{array}
$$

But now, this is a *first order* proposition

Inducing in $HOL_{\lambda\sigma}$

$$\forall R \forall \tau (Noeth(R, \tau) \Rightarrow NoethInd(R, \tau)), Th_u$$
$$\vdash \forall x (x \in \tau \Rightarrow Q(x))$$

Inducing in $HOL_{\lambda\sigma}$

$$\forall R \forall \tau (Noeth(R, \tau) \Rightarrow NoethInd(R, \tau)), Th_u$$
$$\vdash \forall x (x \in \tau \Rightarrow Q(x))$$
$$\Leftarrow$$
$$\forall R \forall \tau (Noeth(R, \tau) \Rightarrow NoethInd(R, \tau)), Th_u,$$
$$X \in \tau \vdash Q(X)$$

Inducing in $HOL_{\lambda\sigma}$

$$\forall R \forall \tau (Noeth(R,\tau) \Rightarrow NoethInd(R,\tau)), Th_u$$
$$\vdash \forall x (x \in \tau \Rightarrow Q(x))$$
$$\Leftarrow$$
$$\forall R \forall \tau (Noeth(R,\tau) \Rightarrow NoethInd(R,\tau)), Th_u,$$
$$X \in \tau \vdash Q(X)$$
$$\Leftrightarrow$$
$$\forall R \forall \tau (Noeth(R,\tau) \Rightarrow NoethInd(R,\tau)), Th_u,$$
$$X \in \tau, \forall \underline{y}\, ((\underline{y} \in \tau \wedge R(X,\underline{y}) \Rightarrow Q(\underline{y})) \vdash Q(X)$$

Inducing in $HOL_{\lambda\sigma}$

$$
\begin{gathered}
\forall R \forall \tau (Noeth(R,\tau) \Rightarrow NoethInd(R,\tau)), Th_u \\
\vdash \forall x (x \in \tau \Rightarrow Q(x)) \\
\Leftarrow \\
\forall R \forall \tau (Noeth(R,\tau) \Rightarrow NoethInd(R,\tau)), Th_u, \\
X \in \tau \vdash Q(X) \\
\Leftrightarrow \\
\forall R \forall \tau (Noeth(R,\tau) \Rightarrow NoethInd(R,\tau)), Th_u, \\
X \in \tau, \forall \underline{y} \, ((\underline{y} \in \tau \wedge R(X,\underline{y}) \Rightarrow Q(\underline{y})) \vdash Q(X)
\end{gathered}
$$

Let us look at the case where $Q(X)$ is an equality of the form $t_1(X) = t_2(X)$

Internalizing the induction hypothesis

$$\mathcal{T}h = \forall R \forall \tau (Noeth(R, \tau) \Rightarrow NoethInd(R, \tau)), Th_{\approx}, Th_{=}$$

Internalizing the induction hypothesis

$$\mathcal{T}h = \forall R \forall \tau (Noeth(R,\tau) \Rightarrow NoethInd(R,\tau)), Th_{\approx}, Th_{=}$$

$$\mathcal{T}h, Th_u, X \in \tau \vdash t_1(X) \approx t_2(X)$$

Internalizing the induction hypothesis

$$\mathcal{T}h = \forall R \forall \tau (Noeth(R, \tau) \Rightarrow NoethInd(R, \tau)), Th_{\approx}, Th_{=}$$

$$\mathcal{T}h, Th_u, X \in \tau \vdash t_1(X) \approx t_2(X)$$

$$\Leftrightarrow$$

$$\mathcal{T}h, Th_u, X \in \tau, \forall \underline{y}\, ((\underline{y} \in \tau \wedge R(X, \underline{y})) \Rightarrow t_1(\underline{y}) \approx t_2(\underline{y})) \vdash t_1(X) \approx t_2(X)$$

Internalizing the induction hypothesis

$$\mathcal{T}h = \forall R \forall \tau (Noeth(R,\tau) \Rightarrow NoethInd(R,\tau)), Th_{\approx}, Th_{=}$$

$$\mathcal{T}h, Th_u, X \in \tau \vdash t_1(X) \approx t_2(X)$$

$$\Leftrightarrow$$

$$\mathcal{T}h, Th_u, X \in \tau, \forall \underline{y}\,((\underline{y} \in \tau \wedge R(X,\underline{y})) \Rightarrow t_1(\underline{y}) \approx t_2(\underline{y})) \vdash t_1(X) \approx t_2(X)$$

$$\Leftrightarrow$$

$$\mathcal{T}h, Th_u, X \in \tau \vdash_{t_1(\underline{y}) \approx t_2(\underline{y}) \text{ if } \underline{y} \in \tau \wedge R(X,\underline{y})} t_1(X) \approx t_2(X)$$

Internalization theorem when $Q(x) = t_1(x) \approx t_2(x)$

To prove by Noetherian induction that the property

$$\forall x \, (x \in \tau \Rightarrow t_1(x) \approx t_2(x))$$

holds in Th_u, it is enough to prove that

$$\mathcal{T}h, Th_u, X \in \tau \vdash_{\mathcal{RE}_{Ind(Q)}(X), \lambda\sigma\mathcal{L}, \mathcal{RE}_u} Q'(X)$$

where

$$\mathcal{RE}_{Ind(Q)}(X) = t_1(\underline{y}) \approx t_2(\underline{y}) \text{ if } \underline{y} \in \tau \wedge R(X, \underline{y})$$

$$Q(X) =^{\mathcal{T}h, Th_u, X \in \tau}_{\mathcal{RE}_{Ind(Q)}(X) \cup \mathcal{RE}_u} Q'(X)$$

Notice that we can add induction levels using the modularity of deduction modulo.

Internalization theorem when $Q(x) = t_1(x) \approx t_2(x)$

To prove by Noetherian induction that the property

$$\forall x \, (x \in \tau \Rightarrow t_1(x) \approx t_2(x))$$

holds in Th_u, it is enough to prove that

$$\mathcal{T}h, Th_u, X \in \tau \vdash_{\mathcal{RE}_{Ind(Q)}(X), \lambda\sigma\mathcal{L}, \mathcal{RE}_u} Q'(X)$$

where

$$\mathcal{RE}_{Ind(Q)}(X) = t_1(\underline{y}) \approx t_2(\underline{y}) \text{ if } \underline{y} \in \tau \wedge R(X, \underline{y})$$

$$Q(X) =^{\mathcal{T}h, Th_u, X \in \tau}_{\mathcal{RE}_{Ind(Q)}(X) \cup \mathcal{RE}_u} Q'(X)$$

Notice that we can add induction levels using the modularity of deduction modulo.

How to continue the proof?

How to continue the proof? Using the $X \in \tau$ hypothesis

- Expand the $X \in \tau$ hypothesis
  explicitely with an induction scheme

- This can be done implicitly by various means:
  - ⋆ test sets (Spike)
  - ⋆ covering sets (RRL)
  - ⋆ narrowing
  - ⋆ ...

How to continue the proof? Getting rid of the conditions

$$\mathcal{RE}_{Ind(Q)}(X) = t_1(\underline{y}) \approx t_2(\underline{y}) \text{ if } \underline{y} \in \tau \wedge R(X, \underline{y})$$

- $\underline{y} \in \tau$ is always satisfied if
  - ⋆ we start with a well-typed goal
  - ⋆ the way we expand $X \in \tau$ preserves types

  in practice it is always the case in many-sorted theories

- $R(X, \underline{y})$ is always satisfied as soon as
  - ⋆ the goal Q has been reduced
  - ⋆ the induction hypothesis is used on a subterm of Q

Condition $R(X, \underline{y})$

- IF the relation R is the simplification ordering ordering orienting $\mathcal{RE}_u$
- AFTER reducing Q by a rule $l \to r \in \mathcal{RE}_u$
- ANY possible application of $\mathcal{RE}_{Ind(Q)}$ will have its $R(X, \underline{y})$ condition verified.

Example: definitions

- definition for sort Nat

$$Th^{sort}_{Nat} = \left\{ \begin{array}{l} \forall x \, (x \in Nat \Leftrightarrow \\ (x = 0 \vee \exists y \, (y \in Nat \wedge x = s(y)) \\ \vee \exists y \, \exists z \, (y \in Nat \wedge z \in Nat \wedge x = y + z))) \end{array} \right.$$

- definition for addition

$$Th^{def+}_{Nat} = \left\{ \begin{array}{l} \forall x \, (x \in Nat \Rightarrow x + 0 \approx x) \\ \forall x \forall y \, ((x \in Nat \wedge y \in Nat) \Rightarrow x + s(y) \approx s(x + y)) \end{array} \right.$$

Example: internalizing the user theory

We want to prove the proposition $\forall x(x \in Nat \Rightarrow 0 + x \approx x)$

$$\mathcal{T}h, Th^{sort}_{Nat}, Th^{def+}_{Nat} \vdash \forall x(x \in Nat \Rightarrow 0 + x \approx x)$$

is equivalent to:

$$\mathcal{T}h, Th^{sort}_{Nat} \vdash_{\mathcal{RE}_{Th^{def+}_{Nat}}} \forall x(x \in Nat \Rightarrow 0 + x \approx x)$$

where

$$\mathcal{RE}_{Th^{def+}_{Nat}} = \left\{ \begin{array}{rcll} x + 0 & \approx & x & \text{if} \quad x \in Nat \\ x + s(y) & \approx & s(x+y) & \text{if} \quad x \in Nat \wedge y \in Nat \end{array} \right.$$

$\mathcal{RE}_{Th^{def+}_{Nat}}$ can be oriented

$$\overrightarrow{\mathcal{RE}_{Th^{def+}_{Nat}}} = \left\{ \begin{array}{rcll} x + 0 & \rightarrow & x & \text{if} \quad x \in Nat \\ x + s(y) & \rightarrow & s(x+y) & \text{if} \quad x \in Nat \wedge y \in Nat \end{array} \right.$$

Example: using induction

$$\mathcal{T}h, Th_{Nat}^{sort} \vdash_{\mathcal{RE}_{Th_{Nat}^{def+}}} \forall x (x \in Nat \Rightarrow 0 + x \approx x)$$

is implied by:

$$\mathcal{T}h, Th_{Nat}^{sort}, X \in Nat \vdash_{\mathcal{RE}_{Th_{Nat}^{def+}} \cup \mathcal{RE}_{Ind(0+x\approx x)}(X)} 0 + X \approx X$$

where

$\mathcal{RE}_{Ind(0+x\approx x)}(X) = \{0 + \underline{y} \approx \underline{y}$ if $\underline{y} \in Nat \wedge \underline{y}\langle X\}$

Example: using the $X \in Nat$ hypothesis

Using the induction scheme $\forall x\,(x \in Nat \Leftrightarrow (x = 0 \vee \exists y\,(y \in Nat \wedge x = s(y))))$

$$\mathcal{T}h, Th^{sort}_{Nat}, X \in Nat \vdash_{\mathcal{RE}_{Th^{def+}_{Nat}} \cup \mathcal{RE}_{Ind(0+x\approx x)}(X)} 0 + X \approx X$$

is implied by

$$\mathcal{T}h, Th^{sort}_{Nat} \vdash_{\mathcal{RE}_{Th^{def+}_{Nat}} \cup \mathcal{RE}_{Ind(0+x\approx x)}(0)} 0 + 0 \approx 0$$

and

$$\mathcal{T}h, Th^{sort}_{Nat}, Y \in Nat \vdash_{\mathcal{RE}_{Th^{def+}_{Nat}} \cup \mathcal{RE}_{Ind(0+x\approx x)}(s(Y))} 0 + s(Y) \approx s(Y)$$

Example: finishing the proof

to prove:

$$\mathcal{T}h, Th_{Nat}^{sort} \vdash_{\mathcal{RE}_{Th_{Nat}^{def+}} \cup \mathcal{RE}_{Ind(0+x\approx x)}(0)} 0 + 0 \approx 0$$

is equivalent to prove

$$\mathcal{T}h, Th_{Nat}^{sort} \vdash_{\mathcal{RE}_{Th_{Nat}^{def+}} \cup \mathcal{RE}_{Ind(0+x\approx x)}(0)} 0 \approx 0$$

which is trivial using $Th_{\approx}$ and to prove

$$\mathcal{T}h, Th_{Nat}^{sort}, Y \in Nat \vdash_{\mathcal{RE}_{Th_{Nat}^{def+}} \cup \mathcal{RE}_{Ind(0+x\approx x)}(s(Y))} 0 + s(Y) \approx s(Y)$$

is equivalent to prove

$$\mathcal{T}h, Th_{Nat}^{sort}, Y \in Nat \vdash_{\mathcal{RE}_{Th_{Nat}^{def+}} \cup \mathcal{RE}_{Ind(0+x\approx x)}(s(Y))} s(Y) \approx s(Y)$$

which is also trivial using $Th_{\approx}$

To sum-up

- Deduction modulo is “just” a specific presentation of predicate logic
- In deduction modulo, the congruence identifies not only terms but also propositions
- Deduction modulo is a powerful paradigm for
  - ⋆ programming
  - ⋆ proof search
  - ⋆ proof representation)

Conclusion

Conclusion

- Using deduction modulo, we can use induction in an explicit or implicit way

Conclusion

- Using deduction modulo, we can use induction in an explicit or implicit way
- Formal setting for the combination of explicit and implicit induction:
  Coq + ELAN

Conclusion

- Using deduction modulo, we can use induction in an explicit or implicit way
- Formal setting for the combination of explicit and implicit induction:
  Coq + ELAN
- Formal setting for the combination of assisted and automated theorem proving

Conclusion

- Using deduction modulo, we can use induction in an explicit or implicit way
- Formal setting for the combination of explicit and implicit induction:
  Coq + ELAN
- Formal setting for the combination of assisted and automated theorem proving
- The combination of computation and deduction is the *deduction modulo* paradigm shows to be the right way to understand techniques already known and to combine them with others.

Conclusion

- Using deduction modulo, we can use induction in an explicit or implicit way
- Formal setting for the combination of explicit and implicit induction:
  Coq $+$ ELAN
- Formal setting for the combination of assisted and automated theorem proving
- The combination of computation and deduction is the *deduction modulo* paradigm shows to be the right way to understand techniques already known and to combine them with others.
- What about application to modal logics?

Proof by consistency

Two main ideas:

Proof by consistency

Two main ideas:

- perform deduction at the level of the congruence: Power of ATP

Proof by consistency

Two main ideas:

- perform deduction at the level of the congruence: Power of ATP
  Saturation methods like completion, ...

Proof by consistency

Two main ideas:

- perform deduction at the level of the congruence: Power of ATP
  Saturation methods like completion, ...
- use an axiomatization of the domain

Proof by consistency

Two main ideas:

- perform deduction at the level of the congruence: Power of ATP
  Saturation methods like completion, ...
- use an axiomatization of the domain
  in case of peano

$$Th_P = \begin{cases} \forall x \in Nat \;\; s(x) \neq 0 \\ \forall x, y \in Nat \;\; s(x) = s(y) \Rightarrow x = y \end{cases}$$

Example: $double(x) = 0 \Rightarrow x = 0$

We want to prove:
$..,Th_d, Th_P \vdash \forall x(x \in Nat \Rightarrow (double(x) \approx 0 \Rightarrow x \approx 0))$

Example: $double(x) = 0 \Rightarrow x = 0$

We want to prove:

$.., Th_d, Th_P \vdash \forall x(x \in Nat \Rightarrow (double(x) \approx 0 \Rightarrow x \approx 0))$

$.., Th_d, Th_P, \forall y(y\langle X \Rightarrow (d(y) = 0 \Rightarrow y = 0)) \vdash (d(X) \approx 0 \Rightarrow X \approx 0)$

Example: $double(x) = 0 \Rightarrow x = 0$

We want to prove:
$.., Th_d, Th_P \vdash \forall x(x \in Nat \Rightarrow (double(x) \approx 0 \Rightarrow x \approx 0))$
$.., Th_d, Th_P, \forall y(y \langle X \Rightarrow (d(y) = 0 \Rightarrow y = 0)) \vdash (d(X) \approx 0 \Rightarrow X \approx 0)$
$X = 0$: trivial after application of definition

Example: $double(x) = 0 \Rightarrow x = 0$

We want to prove:

$.., Th_d, Th_P \vdash \forall x(x \in Nat \Rightarrow (double(x) \approx 0 \Rightarrow x \approx 0))$

$.., Th_d, Th_P, \forall y(y\langle X \Rightarrow (d(y) = 0 \Rightarrow y = 0)) \vdash (d(X) \approx 0 \Rightarrow X \approx 0)$

$X = 0$: trivial after application of definition

$X = s(Z)$

$\ldots, Th_d, Th_P, \vdash_{\mathcal{RE}_{Th_d}, (y\langle s(Z) \wedge d(y)=0) \Rightarrow y=0} (d(s(Z)) \approx 0 \Rightarrow s(Z) \approx 0)$

Example: $double(x) = 0 \Rightarrow x = 0$

We want to prove:

$.., Th_d, Th_P \vdash \forall x(x \in Nat \Rightarrow (double(x) \approx 0 \Rightarrow x \approx 0))$

$.., Th_d, Th_P, \forall y(y\langle X \Rightarrow (d(y) = 0 \Rightarrow y = 0)) \vdash (d(X) \approx 0 \Rightarrow X \approx 0)$

$X = 0$: trivial after application of definition

$X = s(Z)$

$\ldots, Th_d, Th_P, \vdash_{\mathcal{RE}_{Th_d}, (y\langle s(Z) \wedge d(y)=0) \Rightarrow y=0} (d(s(Z)) \approx 0 \Rightarrow s(Z) \approx 0)$

$\ldots, Th_d, Th_P, \vdash_{\mathcal{RE}_{Th_d}, y\langle s(Z) \wedge d(y)=0 \Rightarrow y=0} (ss(d(Z)) \approx 0 \Rightarrow s(Z) \approx 0)$

Example: $double(x) = 0 \Rightarrow x = 0$

We want to prove:
$.., Th_d, Th_P \vdash \forall x(x \in Nat \Rightarrow (double(x) \approx 0 \Rightarrow x \approx 0))$
$.., Th_d, Th_P, \forall y(y\langle X \Rightarrow (d(y) = 0 \Rightarrow y = 0)) \vdash (d(X) \approx 0 \Rightarrow X \approx 0)$
$X = 0$: trivial after application of definition
$X = s(Z)$
$\ldots, Th_d, Th_P, \vdash_{\mathcal{RE}_{Th_d}, (y\langle s(Z) \wedge d(y)=0) \Rightarrow y=0} (d(s(Z)) \approx 0 \Rightarrow s(Z) \approx 0)$
$\ldots, Th_d, Th_P, \vdash_{\mathcal{RE}_{Th_d}, y\langle s(Z) \wedge d(y)=0 \Rightarrow y=0} (ss(d(Z)) \approx 0 \Rightarrow s(Z) \approx 0)$
in this context: $(ss(d(Z)) \approx 0 \Rightarrow s(Z) \approx 0) \equiv \neg(ss(d(Z)) \approx 0), s(Z) \approx 0$
and we conclude with the use of Th_P

Example: $double(x) = 0 \Rightarrow x = 0$

We want to prove:
$.., Th_d, Th_P \vdash \forall x(x \in Nat \Rightarrow (double(x) \approx 0 \Rightarrow x \approx 0))$
$.., Th_d, Th_P, \forall y(y \langle X \Rightarrow (d(y) = 0 \Rightarrow y = 0)) \vdash (d(X) \approx 0 \Rightarrow X \approx 0)$
$X = 0$: trivial after application of definition
$X = s(Z)$
$\ldots, Th_d, Th_P, \vdash_{\mathcal{RE}_{Th_d},(y\langle s(Z) \wedge d(y)=0) \Rightarrow y=0} (d(s(Z)) \approx 0 \Rightarrow s(Z) \approx 0)$
$\ldots, Th_d, Th_P, \vdash_{\mathcal{RE}_{Th_d}, y\langle s(Z) \wedge d(y)=0 \Rightarrow y=0} (ss(d(Z)) \approx 0 \Rightarrow s(Z) \approx 0)$
in this context: $(ss(d(Z)) \approx 0 \Rightarrow s(Z) \approx 0) \equiv \neg(ss(d(Z)) \approx 0), s(Z) \approx 0$
and we conclude with the use of Th_P
—no application of the induction hypothesis!—

$$\frac{}{\Gamma, P \vdash_{\mathcal{RE}} Q}\text{axiom} \quad \text{if } P =^{\Gamma}_{\mathcal{RE}} Q$$

$$\frac{\Gamma, P \vdash_{\mathcal{RE}} \Delta \quad \Gamma \vdash_{\mathcal{RE}} Q, \Delta}{\Gamma \vdash_{\mathcal{RE}} \Delta}\text{cut} \quad \text{if } P =^{\Gamma}_{\mathcal{RE}} Q$$

$$\frac{\Gamma, Q_1, Q_2 \vdash_{\mathcal{RE}} \Delta}{\Gamma, P \vdash_{\mathcal{RE}} \Delta}\text{contr-l} \quad \text{if } P =^{\Gamma}_{\mathcal{RE}} Q_1 =^{\Gamma}_{\mathcal{RE}} Q_2$$

$$\frac{\Gamma \vdash_{\mathcal{RE}} Q_1, Q_2, \Delta}{\Gamma \vdash_{\mathcal{RE}} P, \Delta}\text{contr-r} \quad \text{if } P =^{\Gamma}_{\mathcal{RE}} Q_1 =^{\Gamma}_{\mathcal{RE}} Q_2$$

$$\frac{\Gamma \vdash_{\mathcal{RE}} \Delta}{\Gamma, P \vdash_{\mathcal{RE}} \Delta}\text{weak-l}$$

$$\frac{\Gamma \vdash_{\mathcal{RE}} \Delta}{\Gamma \vdash_{\mathcal{RE}} P, \Delta}\text{weak-r}$$

$$\frac{\Gamma, P, Q \vdash_{\mathcal{RE}} \Delta}{\Gamma, R \vdash_{\mathcal{RE}} \Delta}\wedge\text{-l} \quad \text{if } R =^{\Gamma}_{\mathcal{RE}} (P \wedge Q)$$

$$\frac{\Gamma \vdash_{\mathcal{RE}} P, \Delta \quad \Gamma \vdash_{\mathcal{RE}} Q, \Delta}{\Gamma \vdash_{\mathcal{RE}} R, \Delta}\wedge\text{-r} \quad \text{if } R =^{\Gamma}_{\mathcal{RE}} (P \wedge Q)$$

$$\frac{\Gamma, P \vdash_{\mathcal{RE}} \Delta \quad \Gamma, Q \vdash_{\mathcal{RE}} \Delta}{\Gamma, R \vdash_{\mathcal{RE}} \Delta}\vee\text{-l} \quad \text{if } R =^{\Gamma}_{\mathcal{RE}} (P \vee Q)$$

$$\frac{\Gamma \vdash_{\mathcal{RE}} P, Q, \Delta}{\Gamma \vdash_{\mathcal{RE}} R, \Delta}\vee\text{-r} \quad \text{if } R =^{\Gamma}_{\mathcal{RE}} (P \vee Q)$$

TOPS

$$\frac{\Gamma \vdash_{\mathcal{RE}} P, \Delta \quad \Gamma, Q \vdash_{\mathcal{RE}} \Delta}{\Gamma, R \vdash_{\mathcal{RE}} \Delta}\Rightarrow\text{-l} \quad \text{if } R =^{\Gamma}_{\mathcal{RE}} (P \Rightarrow Q)$$

$$\frac{\Gamma, P \vdash_{\mathcal{RE}} Q, \Delta}{\Gamma \vdash_{\mathcal{RE}} R, \Delta}\Rightarrow\text{-r} \quad \text{if } R =^{\Gamma}_{\mathcal{RE}} (P \Rightarrow Q)$$

$$\frac{\Gamma \vdash_{\mathcal{RE}} P, \Delta}{\Gamma, R \vdash_{\mathcal{RE}} \Delta}\neg\text{-l} \quad \text{if } R =^{\Gamma}_{\mathcal{RE}} \neg P$$

$$\frac{\Gamma, P \vdash_{\mathcal{RE}} \Delta}{\Gamma \vdash_{\mathcal{RE}} R, \Delta}\neg\text{-r} \quad \text{if } R =^{\Gamma}_{\mathcal{RE}} \neg P$$

$$\frac{}{\Gamma, P \vdash_{\mathcal{RE}} \Delta}\bot\text{-l} \quad \text{if } P =^{\Gamma}_{\mathcal{RE}} \bot$$

$$\frac{\Gamma, Q\{t/x\} \vdash_{\mathcal{RE}} \Delta}{\Gamma, P \vdash_{\mathcal{RE}} \Delta}(Q, x, t)\ \forall\text{-l} \quad \text{if } P =^{\Gamma}_{\mathcal{RE}} \forall x\ Q$$

$$\frac{\Gamma \vdash_{\mathcal{RE}} Q\{y/x\}, \Delta}{\Gamma \vdash_{\mathcal{RE}} P, \Delta}(Q, x, y)\ \forall\text{-r} \quad \text{if } \begin{cases} P =^{\Gamma}_{\mathcal{RE}} \forall x\ Q \\ y \text{ fresh variable} \end{cases}$$

$$\frac{\Gamma, Q\{y/x\} \vdash_{\mathcal{RE}} \Delta}{\Gamma, P \vdash_{\mathcal{RE}} \Delta}(Q, x, y)\ \exists\text{-l} \quad \text{if } \begin{cases} P =^{\Gamma}_{\mathcal{RE}} \exists x\ Q \\ y \text{ fresh variable} \end{cases}$$

$$\frac{\Gamma \vdash_{\mathcal{RE}} Q\{t/x\}, \Delta}{\Gamma \vdash_{\mathcal{RE}} P, \Delta}(Q, x, t)\ \exists\text{-r} \quad \text{if } P =^{\Gamma}_{\mathcal{RE}} \exists x\ Q$$

$HOL_{\lambda\sigma}$: syntax

1_A^Γ	constant of sort	$A.\Gamma \vdash A$
$\alpha_{A\to B,A}^\Gamma$	binary function of rank	$(\Gamma \vdash A \to B, \Gamma \vdash A)\Gamma \vdash B$
$\lambda_{A,B}^\Gamma$	unary function of rank	$(A.\Gamma \vdash B)\Gamma \vdash A \to B$
$[\,]_A^{\Gamma,\Gamma'}$	binary function of rank	$(\Gamma' \vdash A, \Gamma \vdash \Gamma')\Gamma \vdash A$
id^Γ	constant of sort	$\Gamma \vdash \Gamma$
$\uparrow_A^\Gamma$	constant of sort	$A.\Gamma \vdash \Gamma$
$\cdot_A^{\Gamma,\Gamma'}$	binary function of rank	$(\Gamma \vdash A, \Gamma \vdash \Gamma')\Gamma \vdash A.\Gamma'$
$\circ^{\Gamma,\Gamma',\Gamma''}$	binary function of rank	$(\Gamma \vdash \Gamma'', \Gamma'' \vdash \Gamma')\Gamma \vdash \Gamma'$
$\dot{\Rightarrow}$	constant of sort	$\vdash o \to o \to o$
$\dot{\wedge}$	constant of sort	$\vdash o \to o \to o$
$\dot{\vee}$	constant of sort	$\vdash o \to o \to o$
$\dot{\neg}$	constant of sort	$\vdash o \to o$
$\dot{\bot}$	constant of sort	$\vdash o$
$\dot{\forall}_A$	constant of sort	$\vdash (A \to o) \to o$
$\dot{\exists}_A$	constant of sort	$\vdash (A \to o) \to o$
ε	of rank	$(\vdash o)$

TOPS

The rewrite rules of $\lambda\sigma$-calculus

TOPS

$Beta$	$(\lambda a)b$	$\longrightarrow$	$a[b.id]$
Eta	$\lambda(a\ 1)$	$\longrightarrow$	b
			if $a =_\sigma b[\uparrow]$
App	$(a\ b)[s]$	$\longrightarrow$	$(a[s]\ b[s])$
$VarCons$	$1[a.s]$	$\longrightarrow$	a
Id	$a[id]$	$\longrightarrow$	a
Abs	$(\lambda a)[s]$	$\longrightarrow$	$\lambda(a[1.(s \circ \uparrow)])$
$Clos$	$(a[s])[t]$	$\longrightarrow$	$a[s \circ t]$
IdL	$id \circ s$	$\longrightarrow$	s
$ShiftCons$	$\uparrow \circ (a.s)$	$\longrightarrow$	s
$AssEnv$	$(s_1 \circ s_2) \circ s_3$	$\longrightarrow$	$s_1 \circ (s_2 \circ s_3)$
$MapEnv$	$(a.s) \circ t$	$\longrightarrow$	$a[t].(s \circ t)$
IdR	$s \circ id$	$\longrightarrow$	s
$VarShift$	$1. \uparrow$	$\longrightarrow$	id
$Scons$	$1[s].(\uparrow \circ s)$	$\longrightarrow$	s

The $\mathcal{L}$-rewrite rules

$$
\begin{array}{rcl}
\varepsilon(\dot{\Rightarrow}\ x\ y) & \longrightarrow & \varepsilon(x) \Rightarrow \varepsilon(y) \\
\varepsilon(\dot{\wedge}\ x\ y) & \longrightarrow & \varepsilon(x) \wedge \varepsilon(y) \\
\varepsilon(\dot{\vee}\ x\ y) & \longrightarrow & \varepsilon(x) \vee \varepsilon(y) \\
\varepsilon(\dot{\neg}\ x) & \longrightarrow & \neg\varepsilon(x) \\
\varepsilon(\dot{\bot}) & \longrightarrow & \bot \\
\varepsilon(\dot{\forall}_T\ x) & \longrightarrow & \forall y\ \varepsilon(x\ y) \\
\varepsilon(\dot{\exists}_T\ x) & \longrightarrow & \exists y\ \varepsilon(x\ y)
\end{array}
$$

TOPS

Pre-cooking

Translation of a λ-term into a $\lambda\sigma$-term

- $F((\lambda x.a), l) = \lambda(F(a, x.l))$,
- $F((a\ b), l) = F(a, l)F(b, l)$,
- $F(x, l) = 1[\uparrow^{k-1}]$, if x is the k-th variable of l
- $F(x, l) = x[\uparrow^{n}]$ where n is the length of l if x is a variable not occurring in l or a constant.

TOPS