FIRST-ORDER LOGIC: REASONING ABOUT EQUALITY

Course “Computational Logic”

Wolfgang Schreiner
Research Institute for Symbolic Computation (RISC)
Wolfgang.Schreiner@risc.jku.at

JOHANNES KEPLER UNIVERSITY LINZ

Equality

So far, the binary predicate symbol "=" has played no special role; however, due to its central role in mathematics, it deserves particular attention.

- Standard: First-Order Logic with Equality
  - Most important logic in general practice.
  - First-order logic where "=" has the fixed interpretation "equality".
    - Normal model: a structure where $=$ is interpreted as "equality".
    - Simple approach: add explicit equality axioms to every proving problem.
    - More comprehensive: extend first-order proof calculus by rules for equality.
- Alternative: Equational Logic
  - A restricted subset of predicate logic.
  - The only predicate is "=" (other predicates simulated as functions into Bool).
    - Implement special (semi-)decision procedure for this logic.

We will now sketch these alternatives in turn.

Equality Axioms

Equality is the equivalence relation that is a congruence for every predicate/function.

$$\forall x.\ x = x \tag{1}$$

$$\forall x, y.\ x = y \Rightarrow y = x \tag{2}$$

$$\forall x, y, z.\ x = y \wedge y = z \Rightarrow x = z \tag{3}$$

$$\forall x_1, \ldots, x_n, y_1, \ldots, y_n.\ x_1 = y_1 \wedge \ldots \wedge x_n = y_n \Rightarrow f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n) \tag{4}$$

$$\forall x_1, \ldots, x_n, y_1, \ldots, y_n.\ x_1 = y_1 \wedge \ldots \wedge x_n = y_n \Rightarrow p(x_1, \ldots, x_n) \Leftrightarrow p(y_1, \ldots, y_n) \tag{5}$$

- Axioms (1-3): $=$ is reflexive, symmetric, transitive, i.e., $=$ an equivalence relation.
- Axiom schemes (4-5): $=$ is a function/predicate congruence.
  - One instance of the schemes for every function symbol f and every predicate symbol p.
- Theorem: Let Δ be a set of formulas and $eq(\Delta)$ be the equivalence relation axioms together with the instances of the congruence schemes for every function/predicate in Δ. Then Δ is satisfiable by a normal model (valid in all normal models) if and only if $\Delta \cup eq(\Delta)$ is satisfiable (valid).
  - Proof sketch: Any model of $\Delta \cup eq(\Delta)$ can be lifted to a normal model of Δ by partitioning the domain into equivalence classes according to the interpretation of $=$.

Implementation in OCaml

```
let function_congruence (f,n) = ... ;;
let predicate_congruence (p,n) = ... ;;

let equivalence_axioms =
  [<<forall x. x = x>>; <<forall x y z. x = y /\ x = z ==> y = z>>];;

let equalitize fm =
  let allpreds = predicates fm in
  if not (mem ("=",2) allpreds) then fm else
  let preds = subtract allpreds ["=",2] and funcs = functions fm in
  let axioms = itlist (union ** function_congruence) funcs
                      (itlist (union ** predicate_congruence) preds
                              equivalence_axioms) in
  Imp(end_itlist mk_and axioms,fm);;
```

Implementation in OCaml

```
# let ewd = equalitize
 <<(forall x. f(x) ==> g(x)) /\ (exists x. f(x)) /\ (forall x y. g(x) /\ g(y) ==> x = y)
   ==> forall y. g(y) ==> f(y)>>;;
val ewd : fol formula =
  <<(forall x. x = x) /\ (forall x y z. x = y /\ x = z ==> y = z) /\
    (forall x1 y1. x1 = y1 ==> f(x1) ==> f(y1)) /\
    (forall x1 y1. x1 = y1 ==> g(x1) ==> g(y1)) ==>
    (forall x. f(x) ==> g(x)) /\
    (exists x. f(x)) /\ (forall x y. g(x) /\ g(y) ==> x = y) ==>
    (forall y. g(y) ==> f(y))>>

# splittab ewd ;;
Searching with depth limit 0
...
Searching with depth limit 9
- : int list = [9]
```

Simple approach but not very effective in more complex examples.

Sequent Calculus and Equality

We may extend the sequent calculus by the “core” of the equality axioms.

$$\frac{\Gamma, x = y \Rightarrow F[x] \Leftrightarrow F[y] \vdash \Delta}{\Gamma \vdash \Delta}\ \text{(SUBST)} \qquad \frac{\Gamma, t = t \vdash \Delta}{\Gamma \vdash \Delta}\ \text{(REFL)}$$

- Rule (SUBST) represents Leibnitz’s law (the principle of substitutivity):
  - Formula $F[y]$ is identical to $F[x]$ except that *any* (not necessarily *all*) free occurrences of x may be replaced by y (which must remain free in F).
- Rule (SUBST) is equivalent to the more special congruence rules:

$$\frac{\Gamma, t_1 = u_1 \wedge \ldots \wedge t_n = u_n \Rightarrow f(t_1, \ldots, t_n) = f(u_1, \ldots, u_n) \vdash \Delta}{\Gamma \vdash \Delta}\ \text{(CONGF)}$$

$$\frac{\Gamma, t_1 = u_1 \wedge \ldots \wedge t_n = u_n \Rightarrow p(t_1, \ldots, t_n) \Leftrightarrow p(u_1, \ldots, u_n) \vdash \Delta}{\Gamma \vdash \Delta}\ \text{(CONGP)}$$

- From rules (SUBST) and (REFL), also symmetry and transitivity can be derived.

The extended calculus is sound and complete (with respect to <u>normal</u> models) but very inefficient to implement automatically.

First-Order Tableaux and Equality

The method of firder-order tableaux extended by the following rules:

$$\frac{\begin{array}{c} t = u \\ F[t] \end{array}}{F[u]} \qquad \frac{}{t = t}$$

- Replacement: If a branch contains the equality $t = u$ and the formula $F[t]$ with an occurrence of term t that is not in the scope of any quantifier, the branch can be extended by $F[u]$ which is a duplicate of $F[t]$ except that the occurrence of t in $F[t]$ has been replaced by term u in $F[u]$.
- Reflexivity: We may add to any branch the equality $t = t$ for an arbitrary term t.

The extended calculus is sound and complete: if a closed tableau can be derived, its root formula is not satisfiable by any <u>normal</u> model, and vice versa.

Example

Proof of $\forall x.\ \forall y.\ \forall z.\ x = y \land y = z \Rightarrow x = z$:

1.	$\neg\forall x.\ \forall y.\ \forall z.\ x = y \land y = z \Rightarrow x = z$	
2.	$\neg\forall y.\ \forall z.\ c = y \land c = z \Rightarrow c = z$	(1)
3.	$\neg\forall z.\ c = d \land d = z \Rightarrow c = z$	(2)
4.	$\neg(c = d \land d = e \Rightarrow c = e)$	(3)
5.	$c = d \land d = e$	(4)
6.	$\neg(c = e)$	(4)
7.	$c = d$	(5)
8.	$d = e$	(5)
9.	$c = e$	(7,8)
	(6,9)	

Proof of $\forall x.\ \forall y.\ x = y \Rightarrow y = x$:

1.	$\neg\forall x.\ \forall y.\ x = y \Rightarrow y = x$	
2.	$\neg\forall y.\ c = y \Rightarrow y = c$	(1)
3.	$\neg(c = d \Rightarrow d = c)$	(2)
4.	$c = d$	(3)
5.	$\neg(d = c)$	(3)
6.	$\neg(d = d)$	(4,5)
7.	$d = d$	
	(6,7)	

Free-Variable Tableaux and Equality

The method of free-variable tableaux extended by the following rules:

$$\frac{\begin{array}{c} t = u \\ F[t'] \end{array}}{F[u]} \qquad \frac{}{x = x} \qquad \frac{}{f(x_1, \ldots, x_n) = f(x_1, \ldots, x_n)}$$

- MGU Replacement: if $t = u$ and $F[t']$ occur in the same branch of tableau T and σ is a most general unifier of t and t', then we may replace tableau T by $T'\sigma$ where T' is identical to T except that $F[u]$ has been added to the branch.
- Reflexivity: We may add to every branch the equality $x = x$ where x is a fresh variable.
- Function Reflexivity: We may add to every branch the equality $f(x_1, \ldots, x_n) = f(x_1, \ldots, x_n)$ where f is an n-ary function symbol and $x_1, \ldots, x_n$ are fresh variables.

The extended calculus is sound and complete: if a closed tableau can be derived, its root formula is not satisfiable by any normal model, and vice versa.

Example

Proof of $\forall x.\ \exists y.\ (y = f(x) \land \forall z.\ (z = f(x) \Rightarrow y = z))$:

1.	$\neg\forall x.\ \exists y.\ (y = f(x) \land \forall z.\ (z = f(x) \Rightarrow y = z))$	
2.	$\neg\exists y.\ (y = f(c) \land \forall z.\ (z = f(c) \Rightarrow y = z))$	(1)
3.	$\neg(y_1 = f(c) \land \forall z.\ (z = f(c) \Rightarrow y_1 = z))$	(2)

Left branch:

4.	$\neg(y_1 = f(c))$	(3)
5.	$y_2 = y_2$	
	(4,5)	

Right branch:

4.	$\neg\forall z.\ (z = f(c) \Rightarrow f(c) = z)$	(3)
5.	$\neg(d = f(c) \Rightarrow f(c) = d)$	(4)
6.	$d = f(c)$	(5)
7.	$\neg(f(c) = d)$	(5)
8.	$\neg(f(c) = f(c))$	(6,7)
9.	$y_3 = y_3$	
	(8,9)	

Tableau closed with $\sigma = [y_1 \mapsto f(c), y_2 \mapsto f(c), y_3 \mapsto f(c)]$.

Paramodulation

An extension of first-order resolution by a treatment of equality (George Robinson and Lawrence Wos, 1969).

$$\frac{C \cup \{L[t]\} \in F \qquad D \cup \{s = u\} \in F \qquad \sigma \text{ is mgu of t and s} \\ C \cup \{L[t]\} \text{ and } D \cup \{s = u\} \text{ have no common variables} \qquad F \cup \{C\sigma \cup D\sigma \cup \{L[u]\sigma\}\} \vdash}{F \vdash} \text{(PARA)}$$

- The paramodulation rule (PARA):
  - Literal $L[t]$ with an occurrence of term t that is replaced by term u in $L[u]$.
  - Clause $C\sigma \cup D\sigma \cup \{L[u]\sigma\}$ is the paramodulant of $C \cup \{L[t]\}$ and $D \cup \{s = u\}$.
- The paramodulation calculus consists of rules (AX), (RES), (REN), (FACT), (PARA).
  - Soundness: if $F \cup feq(F) \vdash$ can be derived, F is not satisfiable by a normal model.
  - Completeness: if F is not satisf. by a normal model, $F \cup feq(F) \vdash$ can be derived.
    - $feq(F)$ consists of the reflexivity axiom $x = x$ and one function reflexivity axiom $f(x_1, \ldots, x_n) = f(x_1, \ldots, x_n)$ for every n-ary function symbol f in F.
    - In most proofs, function reflexivity axioms are not needed; thus many implementations only use the reflexity axiom.

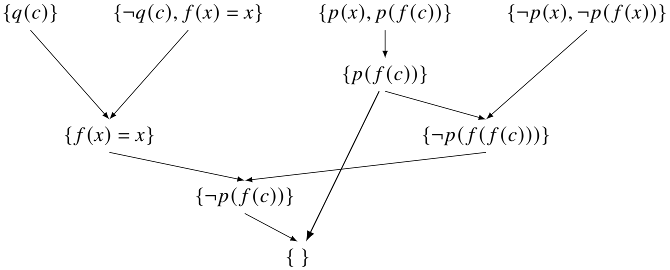
A much more restricted form of the application of equalities.

Example

We show the unsatisfiability of

$$\{\{q(c)\}, \{\neg q(c), f(x) = x\}, \{p(x), p(f(c))\}, \{\neg p(x), \neg p(f(x))\}\}$$

by the following refutation (here reflexivity is not needed):



3 resolution steps, 1 paramodulation step, 1 factorization step.

Paramodulation in OCaml

```
let rec overlapl (l,r) fm rfn = (* Find paramodulations with l = r inside a literal fm. *)
  match fm with
    Atom(R(f,args)) -> listcases (overlaps (l,r))
                              (fun i a -> rfn i (Atom(R(f,a)))) args []
  | Not(p) -> overlapl (l,r) p (fun i p -> rfn i (Not(p)))
  | _ -> failwith "overlapl: not a literal";;

(* Now find paramodulations within a clause. *)
let overlapc (l,r) cl rfn acc = listcases (overlapl (l,r)) rfn cl acc;;

(* Overall paramodulation of ocl by equations in pcl. *)
let paramodulate pcl ocl =
  itlist (fun eq -> let pcl' = subtract pcl [eq] in
                    let (l,r) = dest_eq eq
                    and rfn i ocl' = image (subst i) (pcl' @ ocl') in
                    overlapc (l,r) ocl rfn ** overlapc (r,l) ocl rfn)
         (filter is_eq pcl) [];;
```

Paramodulation in OCaml

```
let para_clauses cls1 cls2 =
  let cls1' = rename "x" cls1 and cls2' = rename "y" cls2 in
  paramodulate cls1' cls2' @ paramodulate cls2' cls1';;

let rec paraloop (used,unused) = (* Incorporation into resolution loop. *)
  match unused with
    [] -> failwith "No proof found"
  | cls::ros ->
        print_string(string_of_int(length used) ^ " used; "^
                     string_of_int(length unused) ^ " unused.");
        print_newline();
        let used' = insert cls used in
        let news =
          itlist (@) (mapfilter (resolve_clauses cls) used')
            (itlist (@) (mapfilter (para_clauses cls) used') []) in
        if mem [] news then true else
        paraloop(used',itlist (incorporate cls) news ros);;
```

Paramodulation in OCaml

```
let pure_paramodulation fm =
  paraloop([],[mk_eq (Var "x") (Var "x")]::simpcnf(specialize(pnf fm)));;

let paramodulation fm =
  let fm1 = askolemize(Not(generalize fm)) in
  map (pure_paramodulation ** list_conj) (simpdnf fm1);;

# paramodulation
 <<(forall x. f(f(x)) = f(x)) /\ (forall x. exists y. f(y) = x)
   ==> forall x. f(x) = x>>;;
0 used; 4 unused.
...
10 used; 108 unused.
11 used; 125 unused.
- : bool list = [true]
```

The naive application of paramodulation leads to huge proof search spaces; in practice, strong restrictions and sophisticated strategies are implemented.

The Superposition Calculus

A spezialization of resolution/paramodulation that leads to smaller search spaces (Leo Bachmair and Harald Ganzinger, 1991).

$$\frac{C \cup \{l = r\} \in F \quad \sigma \text{ is mgu of } l \text{ and } r \quad F \cup \{C\sigma\} \vdash}{F \vdash} \text{(ER)}$$

$$\frac{C \cup \{l_1 = r_1, l_2 = r_2\} \in F \quad \sigma \text{ is mgu of } l_1 \text{ and } l_2 \quad F \cup \{C\sigma \cup \{(l_1 = r_1)\sigma, \neg(r_1 = r_2)\sigma\}\} \vdash}{F \vdash} \text{(EF)}$$

$$\frac{\begin{array}{c} C \cup \{l_1 = r_1\} \in F \qquad D \cup \{l_2[l_1'] = r_2\} \in F \qquad l_1' \text{ is not a variable} \qquad \sigma \text{ is mgu of } l_1 \text{ and } l_1' \\ C \cup \{l_1 = r_1\} \text{ and } D \cup \{l_2[l_1'] = r_2\} \text{ have no common variables} \qquad F \cup \{C\sigma \cup D\sigma \cup \{(l_2[r_1] = r_2)\sigma\}\} \vdash \end{array}}{F \vdash} \text{(SUP)}$$

$$\frac{\begin{array}{c} C \cup \{l_1 = r_1\} \in F \qquad D \cup \{\neg(l_2[l_1'] = r_2)\} \in F \qquad l_1' \text{ is not a variable} \qquad \sigma \text{ is mgu of } l_1 \text{ and } l_1' \\ C \cup \{l_1 = r_1\} \text{ and } D \cup \{l_2[l_1'] = r_2\} \text{ have no common variables} \qquad F \cup \{C\sigma \cup D\sigma \cup \{\neg(l_2[r_1] = r_2)\sigma\}\} \vdash \end{array}}{F \vdash} \text{(SUP)}$$

- Actually constrained forms of above (SUP) rules.
  - Term orderings ensure that equations are only applied in one direction.
  - Still sound and complete with respect to normal models.

Only predicate is =; other predicates are modeled as functions into Bool.

Equational Logic

Let Δ be a set of equations of form $t = u$ which are implicitly universally quantified.

$$\frac{(s = t) \in \Delta}{\Delta \vdash s = t}\ \text{(AXIOM)} \qquad \frac{\Delta \vdash s = t}{\Delta \vdash (s = t)[u/x]}\ \text{(INST)}$$

$$\frac{}{\Delta \vdash t = t}\ \text{(REFL)} \qquad \frac{\Delta \vdash u = t}{\Delta \vdash t = u}\ \text{(SYM)} \qquad \frac{\Delta \vdash t = s \quad \Delta \vdash s = u}{\Delta \vdash t = u}\ \text{(TRANS)}$$

$$\frac{\Delta \vdash t_1 = u_1 \quad \ldots \quad \Delta \vdash t_n = u_n}{\Delta \vdash f(t_1, \ldots, t_n) = f(u_1, \ldots, u_n)}\ \text{(CONG)}$$

- Judgement $\Delta \vdash t = u$
  - Interpreted as “every normal model of Δ satisfies $t = u$”.
  - Equivalent to: $\Delta \models t = u$ holds in first-order logic with equality.
- Birkhoff’s Theorem (Garrett Birkhoff, 1935):
  - If $\Delta \vdash s = t$ is derivable by above inference rules (the “Birkhoff rules”), then every normal model of Δ satisfies $t = u$, and vice versa.

Birkhoff’s rules denote a sound and complete inference calculus for equational logic; like first-order logic, however, equational logic is undecidable.

Equational Proving

- Let set Δ consist of the following equations:

$$g(x, c) = x \tag{1}$$

$$g(x, f(y)) = f(g(x, y)) \tag{2}$$

$$h(x, c) = c \tag{3}$$

$$h(x, f(y)) = g(x, h(x, y)) \tag{4}$$

- How to prove $\Delta \models h(f(f(c)), f(f(c))) = g(h(f(c), f(c)), f(f(f(c))))$?

$$\begin{aligned}
\underline{h(f(f(c)), f(f(c)))} &\overset{(4)}{=} g(f(f(c)), h(f(f(c)), f(c))) \overset{(4)}{=} g(f(f(c)), g(f(f(c)), h(f(f(c)), c))) \\
&\overset{(3)}{=} g(f(f(c)), g(f(f(c)), c)) \overset{(1)}{=} g(f(f(c)), f(f(c))) \overset{(2)}{=} f(g(f(f(c)), f(c))) \\
&\overset{(2)}{=} f(f(g(f(f(c)), c))) \overset{(1)}{=} \underline{f(f(f(f(c))))}
\end{aligned}$$

$$\begin{aligned}
\underline{g(h(f(c), f(c)), f(f(f(c))))} &\overset{(4)}{=} g(g(f(c), h(f(c), c)), f(f(f(c)))) \overset{(3)}{=} g(g(f(c), c), f(f(f(c)))) \\
&\overset{(1)}{=} g(f(c), f(f(f(c)))) \overset{(2)}{=} f(g(f(c), f(f(c)))) \overset{(2)}{=} f(f(g(f(c), f(c)))) \\
&\overset{(2)}{=} f(f(f(g(f(c), c)))) \overset{(1)}{=} \underline{f(f(f(f(c))))}
\end{aligned}$$

By a sequence of equality substitutions in the left term and a sequence of equality substitutions in the right term the same term can be derived; thus the left term and the right term are equal.

Equational Proving

We have just performed a strategy of "simplifying calculations".

- Set Δ described some arithmetic axioms:

$$x + 0 = x \tag{1}$$

$$x + (y') = (x + y)' \tag{2}$$

$$x \cdot 0 = 0 \tag{3}$$

$$x \cdot (y') = x + (x \cdot y) \tag{4}$$

- We have proved $\Delta \models (0'') \cdot (0'') = ((0') \cdot (0')) + (0''')$ (i.e., $2 \cdot 2 = 1 + 3$):

$$\begin{aligned}
\underline{(0'') \cdot (0'')} &\overset{(4)}{=} (0'') + ((0'') \cdot (0')) \overset{(4)}{=} (0'') + ((0'') + ((0'') \cdot 0)) \\
&\overset{(3)}{=} (0'') + ((0'') + 0) \overset{(1)}{=} (0'') + (0'') \overset{(2)}{=} ((0'') + (0'))' \\
&\overset{(2)}{=} ((0'') + 0)'' \overset{(1)}{=} \underline{0''''} \\
\underline{((0') \cdot (0')) + (0''')} &\overset{(4)}{=} ((0') + ((0') \cdot 0)) + (0''') \overset{(3)}{=} ((0') + 0) + (0''') \\
&\overset{(1)}{=} (0') + (0''') \overset{(2)}{=} ((0') + (0''))' \overset{(2)}{=} ((0') + (0'))'' \\
&\overset{(2)}{=} ((0') + 0)''' \overset{(1)}{=} \underline{0''''}
\end{aligned}$$

When can this strategy be performed?

Term Rewriting

Consider the elements of Δ not as equations but as (left-to-right) rewrite rules.

- Abstract reduction system $(S, \to)$: a set S and a binary relation $\to$ on S.
  - $x \leftrightarrow y$: $x \to y$ or $y \to x$.
  - $x \to^* y$ and $x \leftrightarrow^* y$: the reflexive transitive closure of $\to$ and $\leftrightarrow$.
- Term rewriting system: an abstract reduction system induced by Δ.
  - S is the set of terms and $\to$ is the "term rewriting relation" generated by Δ when considering every equation $t = u$ as a (left-to-right) rewrite rule.
- Theorem: Let $\to$ be the term rewriting relation induced by Δ. Then we have $\Delta \models t = u$ if and only if $t \leftrightarrow^* u$.
  - Proof sketch: If $\Delta \models t = u$, by Birkhoff's theorem $\Delta \vdash t = u$ is derivable. One can show by induction on the Birkhoff rules that this implies $t \leftrightarrow^* u$. Conversely, by the semantics of substitution $t \to u$ implies $\Delta \models t = u$; from this one can show by induction that also $t \leftrightarrow^* u$ implies $\Delta \models t = u$.

To show $\Delta \models t = u$ it suffices to show $t \leftrightarrow^* u$.

Term Rewriting as a Decision Strategy

Some fundamental notions and properties of an abstract reduction system $(S, \to)$.

- Element $x \in S$ is a normal form: there is no $y \in S$ such that $x \to y$.
- $\to$ is terminating (Noetherian): there are no infinite reduction sequences $x_0 \to x_1 \to \cdots$, i.e., every reduction sequence ends with a normal form $x_n \in S$.
- $\to$ has the Church-Rosser property: if $x \leftrightarrow^* y$, then $x \to^* z$ and $y \to^* z$ for some $z \in S$.
  - Lemma: If $\to$ has the Church-Rosser property, then for every $x \in S$ there exists *at most* one normal form $x' \in S$ such that $x \to^* x'$.
- $\to$ is canonical: $\to$ is terminating and also has the Church rosser property.
  - Lemma: If $\to$ is canonical, then for every $x \in S$ there exists *exactly one* normal form $x' \in S$ such that $x \to^* x'$.
- Theorem (Trevor Evans, 1951): If $\to$ is canonical and $x \to^* x'$ and $y \to^* y'$ with normal forms $x' \in S$ and $y' \in S$, then $x \leftrightarrow^* y$ holds if and only if $x' = y'$ does.

If Δ induces a canonical term rewriting system, we can decide $\Delta \models t = u$ by rewriting terms t and u to normal forms t' and u' and comparing t' with u'.

Term Rewriting in OCaml

```
let rec rewrite1 eqs t = (* Rewriting at the top level with first of list of equations. *)
  match eqs with
    Atom(R("=",[l;r]))::oeqs ->
     (try tsubst (term_match undefined [l,t]) r
      with Failure _ -> rewrite1 oeqs t)
  | _ -> failwith "rewrite1";;

let rec rewrite eqs tm = (* Rewriting repeatedly and at depth (top-down). *)
  try rewrite eqs (rewrite1 eqs tm) with Failure _ ->
  match tm with
    Var x -> tm
  | Fn(f,args) -> let tm' = Fn(f,map (rewrite eqs) args) in
                  if tm' = tm then tm else rewrite eqs tm';;

# rewrite [<<0 + x = x>>; <<S(x) + y = S(x + y)>>;
         <<0 * x = 0>>; <<S(x) * y = y + x * y>>]
        <<|S(S(S(0))) * S(S(0)) + S(S(S(S(0))))|>>;;
- : term = <<|S(S(S(S(S(S(S(S(S(S(0))))))))))|>>
```

Non-Canonical Term Rewriting

- Not Terminating:

$$x + y = y + x \tag{1}$$

$$c + d \longrightarrow d + c \longrightarrow c + d \longrightarrow \cdots$$

- No Church-Rosser Property:

$$x \cdot (y + z) = x \cdot y + x \cdot z \tag{1}$$

$$(x + y) \cdot z = x \cdot z + y \cdot z \tag{2}$$

$$\begin{aligned}(a + b) \cdot (c + d) &\stackrel{(2)}{\longrightarrow} a \cdot (c + d) + b \cdot (c + d) \\ &\stackrel{(1)}{\longrightarrow} (a \cdot c + a \cdot d) + b \cdot (c + d) \stackrel{(1)}{\longrightarrow} (a \cdot c + a \cdot d) + (b \cdot c + b \cdot d) \\ (a + b) \cdot (c + d) &\stackrel{(1)}{\longrightarrow} (a + b) \cdot c + (a + b) \cdot d \\ &\stackrel{(2)}{\longrightarrow} (a \cdot c + b \cdot c) + (a + b) \cdot d \stackrel{(2)}{\longrightarrow} (a \cdot c + b \cdot c) + (a \cdot d + b \cdot d)\end{aligned}$$

If a term rewriting system is not canonical, rewriting fails as a decision strategy.

Ensuring Termination

- It is generally undecidable whether a term rewriting system is terminating.
  - Term rewriting systems can perform arbitrary computations.
  - The problem whether computing machines halt is undecidable (Alan Turing, 1937).
- But we can prove that a particular term rewriting system is terminating.
  - Determine a suitable termination ordering, i.e., a well-founded relation on terms that is decreased by the application of every rewrite rule.
  - One such termination ordering is the lexicographic path order $t > u$ defined as follows:
    - $t > u$, if u is a proper subterm of t.
    - $f(t_1, \ldots, t_n) > t$, if $t_i > t$ for some i.
    - $f(t_1, \ldots, t_n) > f(u_1, \ldots, u_n)$ if $t_i > u_i$ for some i and $t_j = u_j$ for all $j < i$.
    - $f(t_1, \ldots, t_n) > g(u_1, \ldots, u_m)$, if $f > g$ for some ordering of function/constant symbols.

    In the last two rules we additionally require $f(t_1, \ldots, t_n) > u_i$ for every i.
- Example: consider the lexicographic path order for '$\cdot$' > '+' > '$'$' > '0'.
  - $x + 0 > x$ because x is a proper subterm of $x + 0$.
  - $x + (y') > (x + y)'$ because '+' > '$'$' and $x + (y') > x + y$ (why?).
  - $x \cdot 0 > 0$ because 0 is a proper subterm of $x \cdot 0$.
  - $x \cdot (y') > x + (x \cdot y)$ because '$\cdot$' > '+' and $x \cdot (y') > x$ and $x \cdot (y') > x \cdot y$ (why?).

Thus the previously stated arithmetic term rewriting system is terminating.

Ensuring the Church-Rosser Property

- Does the following term rewriting system have the Church-Rosser Property?

$$(x \cdot y) \cdot z = x \cdot (y \cdot z) \quad (1)$$

$$1 \cdot x = x \quad (2)$$

$$i(x) \cdot x = 1 \quad (3)$$

- We can rewrite term $(1 \cdot x) \cdot y$ in two different ways:

$$(1 \cdot x) \cdot y \stackrel{(1)}{\rightarrow} 1 \cdot (x \cdot y)$$

$$(1 \cdot x) \cdot y \stackrel{(2)}{\rightarrow} x \cdot y$$

- This does not violate the property, because both results have the same normal form:

$$1 \cdot (x \cdot y) \stackrel{(2)}{\rightarrow} x \cdot y$$

- But we can also rewrite term $(i(x) \cdot x) \cdot y$ in two different ways:

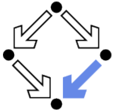
$$(i(x) \cdot x) \cdot y \stackrel{(1)}{\rightarrow} i(x) \cdot (x \cdot y)$$

$$(i(x) \cdot x) \cdot y \stackrel{(3)}{\rightarrow} 1 \cdot y \stackrel{(2)}{\rightarrow} y$$

- Thus we have derived two different normal forms which violates the Church-Rosser property.

This may spark the idea of how to decide the Church-Rosser property.

Ensuring the Church-Rosser Property



- Reduction relation $\rightarrow$ is locally confluent if the following property holds: if $x \rightarrow y_1$ and $x \rightarrow y_2$, then $y_1 \rightarrow^* z$ and $y_2 \rightarrow^* z$ for some $z \in S$.
  - Newman's Lemma: If a reduction relation $\rightarrow$ is both terminating and locally confluent, it is confluent (and thus has the Church-Rosser property).
- Thus, given a set Δ of rewrite rules whose reduction relation $\rightarrow$ is terminating, the following algorithm decides whether $\rightarrow$ has the Church-Rosser property:
  - Consider every pair $l_1 = r_1$ and $l_2 = r_2$ of rewrite rules (both rules may be the same).
  - Rename the variables in these rules such that variables in l_1 and l_2 are disjoint.
  - Determine every critical pair of these rules, i.e., terms $r_1\sigma$ and $l_1[r_2]\sigma$ such that:
    - l_2' is a non-variable term such that σ is the most general unifier of l_2 and l_2' and
    - l_1 contains an occurrence of l_2' and $l_1[r_2]$ is l_1 with that occurrence replaced by r_2.
  - The reduction reduction system has the Church-Rosser property if and only if every critical pair y_1 and y_2 can be rewritten by $\rightarrow$ to a common normal form z.

The decision of the Church-Rosser property is reduced to critical pair computation.

Computing Critical Pairs

- Example: equations $x_1 + 0 = x_1$ and $x_2 + 0 = x_2$ (the first equation renamed).
  - $x_1 + 0$ and $x_2 + 0$ have mgu $[x_1 \mapsto x_2]$ which yields the trivial critical pair x_2 and x_2.
  - The arithmetic system has only trivial critical pairs and thus is Church-Rosser.
  - We only need to consider the overlap of a rule with itself at a proper subterm of the left side.
- Example: $\Delta := \{f(g(f(x))) \xrightarrow{r} g(x)\}$
  - Rule instances $f(g(f(x_1))) \xrightarrow{r_1} g(x_1), f(g(f(x_2))) \xrightarrow{r_2} g(x_2)$
  - Unify $f(x_1)$ and $f(g(f(x_2)))$ with mgu $\sigma = [x_1 \mapsto g(f(x_2))]$.
  - Reduction $f(g(f(g(f(x_2))))) \xrightarrow{r_1} g(g(f(x_2)))$ with normal form $g(g(f(x_2)))$.
  - Reduction $f(g(f(g(f(x_2))))) \xrightarrow{r_2} f(g(g(x_2)))$ with normal form $f(g(g(x_2)))$.
  - Critical pair $g(g(f(x_2)))$ and $f(g(g(x_2)))$ with different normal forms.
  - Δ does not have the Church-Rosser property.

Critical Pairs in OCaml

```
let renamepair (fm1,fm2) = ... ;;
let rec listcases fn rfn lis acc = (* Rewrite with l = r inside tm to give a critical pair. *)
  match lis with
    [] -> acc
  | h::t -> fn h (fun i h' -> rfn i (h'::t)) @ listcases fn (fun i t' -> rfn i (h::t')) t acc;;
let rec overlaps (l,r) tm rfn =
  match tm with
    Fn(f,args) -> listcases (overlaps (l,r)) (fun i a -> rfn i (Fn(f,a))) args
                        (try [rfn (fullunify [l,tm]) r] with Failure _ -> [])
  | Var x -> [];;

let crit1 (Atom(R("=",[l1;r1]))) (Atom(R("=",[l2;r2]))) =
  overlaps (l1,r1) l2 (fun i t -> subst i (mk_eq t r2));;
let critical_pairs fma fmb = (* Generate all critical pairs between two equations. *)
  let fm1,fm2 = renamepair (fma,fmb) in
  if fma = fmb then crit1 fm1 fm2
  else union (crit1 fm1 fm2) (crit1 fm2 fm1);;

# let eq = <<f(f(x)) = g(x)>> in critical_pairs eq eq;;
- : fol formula list = [<<f(g(x0)) = g(f(x0))>>; <<g(x1) = g(x1)>>]
```

Knuth-Bendix Completion

A semi-algorithm to derive a canonical term rewriting system (Donald Knuth and Peter Bendix, 1970).

```
procedure COMPLETE(Δ)    ▷ if the procedure terminates, it returns a canonical system equivalent to Δ
    Δ1 ← Δ
    repeat                                                              ▷ may not terminate
        Δ0 ← Δ1
        for every critical pair (t, u) in Δ0 do
            reduce t and u to normal forms t′ and u′ according to Δ0   ▷ may not terminate
            if t′ ≠ u′ then
                choose l = r ∈ {t = u, u = t}
                Δ1 ← Δ1 ∪ {l = r}
            end if
        end for
    until Δ1 = Δ0
    return Δ1
end procedure
```

There are numerous improvements to increase the practical applicability.

Knuth Bendix Completion

- Example: $\Delta := \{f(g(f(x))) \xrightarrow{r} g(x)\}$
  - Rule instances $f(g(f(x_1))) \xrightarrow{r_1} g(x_1), f(g(f(x_2))) \xrightarrow{r_2} g(x_2)$
    - Unify $f(x_1)$ and $f(g(f(x_2)))$ with mgu $\sigma = [x_1 \mapsto g(f(x_2))]$.
    - Reduction $f(g(f(g(f(x_2))))) \xrightarrow{r_1} g(g(f(x_2)))$ with normal form $g(g(f(x_2)))$.
    - Reduction $f(g(f(g(f(x_2))))) \xrightarrow{r_2} f(g(g(x_2)))$ with normal form $f(g(g(x_2)))$.
    - Critical pair $g(g(f(x_2)))$ and $f(g(g(x_2)))$ with different normal forms.
    - $\Delta' := \{f(g(f(x))) \xrightarrow{r} g(x), g(g(f(x))) \xrightarrow{s} f(g(g(x)))\}$
  - Rule instances $g(g(f(x_1))) \xrightarrow{s_1} f(g(g(x_1)))$ and $g(g(f(x_2))) \xrightarrow{s_2} f(g(g(x_2)))$
    - Only trivial mgu $[x_1 \to x_2]$ and trivial critical pair.
  - Rule instances $f(g(f(x_1))) \xrightarrow{r_1} g(x_1)$ and $g(g(f(x_2))) \xrightarrow{s_1} f(g(g(x_2)))$
    - Unify $f(x_2)$ and $f(g(f(x_1)))$ with mgu $[x_2 \mapsto g(f(x_1))]$.
    - $g(g(f(g(f(x_1))))) \xrightarrow{r_1} g(g(g(x_1)))$ with normal form $g(g(g(x_1)))$.
    - $g(g(f(g(f(x_1))))) \xrightarrow{s_1} f(g(g(g(f(x_1))))) \xrightarrow{s_1} f(g(f(g(g(x_1))))) \xrightarrow{r_1} g(g(g(x_1)))$.
    - Critical pair $g(g(g(x_1)))$ and $f(g(f(g(g(x_1)))))$ has common normal form.
  - No more non-trivial rule overlaps.

Δ' has the Church-Rosser property.

The Case of Variable-Free Equations

Our goal is to derive $\Delta \vdash (t = u)$.

- Consider the special case of only variable-free equations in $\Delta \vdash (t = u)$.
  - Any occurrence of a symbol x in $t = u$ does not denote any more a “variable” (that is universally quantified in the equation) but a “constant” (whose value is the same in all equations in which x occurs).
- Then proofs need not apply the Birkhoff rule (INST).
- This makes the theory decidable.

We will next consider decision procedures for variable-free equational logic and other decidable theories.