A Lean 4 Toolbox for First-Order Term Rewriting: Critical Pairs, Newman Confluence, and Completion Orderings

Short Paper

Anonymous author
Anonymous affiliation

Abstract

We present a mechanized formalization of first-order term rewriting systems (TRSs) in Lean 4. Our development provides a complete confluence proof pipeline: we formalize first-order terms with substitutions and contexts, implement critical pair construction via Robinson unification, and prove that joinability of all critical pairs implies local confluence. We then mechanize Newman's lemma—that termination plus local confluence implies confluence—enabling a standard Knuth-Bendix-style confluence argument. To support termination proofs, we formalize reduction orderings including weight-based orderings (KBO-style), the lexicographic path ordering (LPO), and matrix interpretations, all closed under substitution and context application. We demonstrate the pipeline on concrete TRS examples, proving confluence via computed critical pairs and termination via reduction orderings. The development comprises approximately 5,000 lines of Lean 4 code across 19 modules, is fully executable, and contains no axioms or `sorry` placeholders. All proofs are mechanized; the artifact is reproducible via `lake build`.

2012 ACM Subject Classification Theory of computation → Equational logic and rewriting; Theory of computation → Automated reasoning

Keywords and phrases term rewriting, confluence, critical pairs, Newman's lemma, Knuth-Bendix completion, Lean 4, formalization

Digital Object Identifier 10.4230/LIPIcs.ITP.2026.

Supplementary Material Anonymous supplementary material

1 Introduction

Term rewriting is a foundational computational model with applications ranging from functional programming semantics to automated theorem proving. A term rewriting system (TRS) specifies computation via directed equations: terms are repeatedly simplified by matching a left-hand side and replacing it with the corresponding right-hand side. The *confluence* property—that all reduction sequences from a common source can be joined—is essential for ensuring deterministic computation and unique normal forms.

On paper, the standard confluence proof for terminating TRSs is well-understood: compute all critical pairs (overlaps between rule left-hand sides), verify they are joinable, invoke Newman's lemma to lift local confluence to full confluence. However, mechanizing this pipeline in a proof assistant requires careful attention to: (1) the syntax of first-order terms with positions, substitutions, and contexts; (2) a complete unification algorithm for critical pair construction; (3) the peak decomposition lemma relating local divergences to critical pairs; and (4) reduction orderings with the required closure properties for termination proofs.

We contribute a **Lean 4 library** providing this infrastructure as reusable, verified components:

- **Core TRS semantics**: First-order terms over arbitrary signatures, substitutions, positions, subterm extraction, and replacement (Sections 2.1–2.3).

- **Confluence pipeline**: Critical pair construction via Robinson unification, the critical pair theorem (joinable critical pairs imply local confluence), and Newman's lemma yielding full confluence for terminating TRSs (Sections 3–4).
- **Completion scaffolding**: A reduction ordering interface with verified implementations of weight-based orderings (KBO-style), the lexicographic path ordering (LPO), and matrix interpretations (Section 5).

1.0.0.1 Positioning.

Our contribution is not a new theorem but a clean, reusable mechanization in Lean 4. The library is designed to compose with other metatheory developments—for instance, our abstract rewriting framework already underlies verified Church-Rosser proofs for the lambda calculus and combinatory logic in the same repository.

2 TRS Formalization in Lean 4

2.1 Syntax

A *signature* specifies a type of function symbols with associated arities:

```
structure Signature where
  Sym : Type
  arity : Sym -> Nat
```

First-order terms are built from variables (natural numbers) and applications of function symbols to the appropriate number of arguments:

```
inductive Term (sig : Signature) where
  | var : Nat -> Term sig
  | app : (f : sig.Sym) -> (Fin (sig.arity f) -> Term sig) -> Term sig
```

We represent argument tuples as functions from finite indices, avoiding fixed list lengths while maintaining strong typing. *Substitutions* are functions from variables to terms:

```
abbrev Subst (sig : Signature) := Nat -> Term sig
```

The application of substitution σ to term t, written $t\sigma$ or `Term.subst sub t`, is defined by structural recursion. We prove standard properties including $t[\mathrm{id}] = t$ and $(t\sigma)\tau = t(\sigma \circ \tau)$.

2.2 Positions and Contexts

A *position* is a path from the root to a subterm, represented as a list of argument indices:

```
abbrev Pos := List Nat
```

We define `Term.subterm t p` returning the subterm at position p (if it exists), and `Term.replace t p u` returning the result of replacing the subterm at p with u. Key lemmas include:

- `replace_self`: Replacing a subterm with itself yields the original term.
- `subterm_append`: Subterm extraction composes along paths.
- `replace_subst`: Replacement commutes with substitution.
- `replace_comm_of_disjoint`: Disjoint replacements commute.

2.3 Rewriting

A *rewrite rule* pairs a left-hand side with a right-hand side:

```
structure Rule (sig : Signature) where
  lhs : Term sig
  rhs : Term sig
```

A *rewrite step* $s \to t$ applies a rule at some position under some substitution:

```
structure Step (rules : RuleSet sig) (s t : Term sig) : Prop where
  rule : Rule sig
  rule_mem : rules rule
  pos : Pos
  sub : Subst sig
  subterm_eq : Term.subterm s pos = some (Term.subst sub rule.lhs)
  replace_eq : Term.replace s pos (Term.subst sub rule.rhs) = some t
```

The *multi-step* relation $\to^*$ is the reflexive-transitive closure:

```
abbrev MultiStep (rules : RuleSet sig) := Rewriting.Star (Step rules)
```

This connects to our abstract rewriting framework (module `Rewriting.Basic`), which defines `Star`, `Joinable`, `LocalConfluent`, `Confluent`, and `Terminating` generically for any relation.

3 Critical Pairs and Local Confluence

3.1 Overlaps and Critical Pair Construction

An *overlap* occurs when two rule left-hand sides unify at a non-variable position. Given rules $\ell_1 \to r_1$ and $\ell_2 \to r_2$, if $\ell_1|_p$ (the subterm of ℓ_1 at position p) unifies with ℓ_2 via most general unifier θ, we obtain a *critical pair*:

$$(r_1\theta,\ (\ell_1\theta)[p \leftarrow r_2\theta])$$

We implement Robinson unification with an occurs-check and prove it sound and complete:

```
theorem unify_sound : unify eqs = some sub -> UnifiesList sub eqs
theorem unify_complete : Unifiable eqs -> exists sub, unify eqs = some sub
```

The critical pair construction is captured by:

```
structure Overlap (r1 r2 : Rule sig) (p : Pos) (sub1 sub2 : Subst sig) where
  subterm_eq : Term.subterm (Term.subst sub1 r1.lhs) p =
               some (Term.subst sub2 r2.lhs)
  nonvar : NonVar (Term.subst sub2 r2.lhs)

def mkCriticalPair (r1 r2 : Rule sig) (p : Pos) (sub1 sub2 : Subst sig) :
    Option (CriticalPair sig) :=
  match Term.replace (Term.subst sub1 r1.lhs) p (Term.subst sub2 r2.rhs) with
  | none => none
  | some t => some { left := Term.subst sub1 r1.rhs, right := t }
```

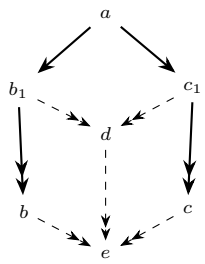

Figure 1 Newman's lemma proof: local confluence closes the top diamond; induction closes the rest.

3.2 Peak Decomposition

The key technical lemma is *peak decomposition*: every local peak (divergence $b \leftarrow s \rightarrow c$) either closes immediately in one step or corresponds to a critical pair.

```
theorem criticalPairsComplete_of_steps :
    forall s b c, Step rules s b -> Step rules s c ->
      (exists d, Step rules b d /\ Step rules c d) \/
      exists cp p, CriticalPairs rules cp /\ ...
```

The proof distinguishes three cases based on the relative positions of the two redexes:

1. **Nested**: One redex is inside the other, yielding a critical pair.
2. **Disjoint**: The redexes do not overlap; the peak closes in one step by commuting the replacements.

3.3 The Critical Pair Theorem

We prove both directions of the critical pair theorem:

```
theorem localConfluent_of_criticalPairsJoinable :
    CriticalPairsJoinable rules -> LocalConfluent rules

theorem criticalPairsJoinable_of_localConfluent :
    LocalConfluent rules -> CriticalPairsJoinable rules
```

The first (completeness) direction is the workhorse: given that all critical pairs join and a local peak $b \leftarrow s \rightarrow c$, we apply peak decomposition. If the peak closes directly, we are done. Otherwise, we obtain a critical pair (u, v) embedded in a context $C[\cdot]_p$ such that $b = C[u]_p$ and $c = C[v]_p$. By hypothesis, $u \downarrow v$; we then lift through the context using `star_context` to obtain $b \downarrow c$.

4 Newman's Lemma and Confluence

4.1 Newman's Lemma

Newman's lemma states that for terminating relations, local confluence implies full confluence (Figure 1).

We prove it in our abstract rewriting framework:

```
theorem confluent_of_terminating_localConfluent :
    Terminating r -> LocalConfluent r -> Confluent r
```

The proof proceeds by well-founded induction: given $a \to^* b$ and $a \to^* c$ with $a \to b_1 \to^* b$ and $a \to c_1 \to^* c$, local confluence joins b_1 and c_1 via d; the IH then closes the diagram.

4.2 Confluence for TRSs

Combining the critical pair theorem with Newman's lemma yields:

```
theorem confluent_of_terminating_criticalPairsJoinable :
   Terminating rules -> CriticalPairsJoinable rules -> Confluent rules
```

This is the standard Knuth-Bendix soundness criterion: compute all critical pairs, prove each is joinable, establish termination, and conclude confluence.

We package this as a *completion certificate*:

```
structure KnuthBendixComplete (rules : RuleSet sig) (ord : ReductionOrdering sig)
     where
  orient : forall r, rules r -> ord.lt r.rhs r.lhs
  criticalPairsJoinable : CriticalPairsJoinable rules

theorem confluent_of_knuthBendixComplete :
   KnuthBendixComplete rules ord -> Confluent rules
```

5 Reduction Orderings

A *reduction ordering* must be well-founded and closed under substitution and context application. If every rule $\ell \to r$ satisfies $r \prec \ell$, the TRS terminates. We implement weight-based orderings (KBO-style), the lexicographic path ordering (LPO) with verified well-foundedness, and 2×2 matrix interpretations. All are packaged as `ReductionOrdering` instances.

6 Worked Example

Consider rules $f(a) \to b$ and $f(b) \to a$ over signature $\{a, b, f\}$. Termination follows from size decrease; critical pairs are trivial (self-overlap yields (b, b)). The confluence certificate assembles both proofs:

```
theorem example_confluent : Confluent rules :=
  confluent_of_knuthBendixComplete
    { orient := rules_oriented, criticalPairsJoinable := ... }
```

The library includes additional examples with KBO, LPO, and matrix interpretations.

7 Related Work and Conclusion

7.0.0.1 Related Work.

Term rewriting theory originates with Newman [4] and Huet [2]; Knuth-Bendix completion [3] remains central. Mechanized developments include IsaFoR [5] (Isabelle) and CoLoR [1] (Coq). Our contribution is a Lean 4-native toolbox designed for composition with lambda calculus formalizations.

7.0.0.2 Conclusion.

We presented a Lean 4 library with: (1) first-order TRS syntax, (2) a verified critical pair theorem, (3) Newman's lemma, and (4) reduction orderings (weight, LPO, matrix). Future work includes Knuth-Bendix completion and higher-order rewriting.

7.0.0.3 Artifact.

The library is available as supplementary material; build via `lake build`. Key modules: `TRS.FirstOrder.{Syntax, CriticalPairs, Confluence, LPO}`. The development contains no axioms or `sorry`.

References

1 Frédéric Blanqui and Adam Koprowski. CoLoR: a Coq library on well-founded rewrite relations and its application to the automated verification of termination certificates. *Mathematical Structures in Computer Science*, 21(4):827–859, 2011. doi:10.1017/S0960129511000120.

2 Gérard Huet. Confluent reductions: Abstract properties and applications to term rewriting systems. *Journal of the ACM*, 27(4):797–821, 1980. doi:10.1145/322217.322230.

3 Donald E. Knuth and Peter B. Bendix. Simple word problems in universal algebras. In *Computational Problems in Abstract Algebra*, pages 263–297. Pergamon Press, 1970.

4 Maxwell H. A. Newman. On theories with a combinatorial definition of "equivalence". *Annals of Mathematics*, 43(2):223–243, 1942. doi:10.2307/1968867.

5 René Thiemann and Christian Sternagel. Certification of termination proofs using CeTA. In *Theorem Proving in Higher Order Logics, 22nd International Conference, TPHOLs 2009*, volume 5674 of *Lecture Notes in Computer Science*, pages 452–468. Springer, 2009. doi:10.1007/978-3-642-03359-9_31.