Mechanizing Undecidable Properties of Rewriting Systems using the Rocq Prover

Lecture 2: Synthetic Computability

Andrej Dudenhefner

ISR 2026 – 15th International School on Rewriting

2026-07-12

Course Roadmap

1. Formalizing string rewriting ($\mathtt{SR}$) in Rocq
2. **Synthetic computability**
   - Traditional decidability
   - Synthetic decidability and enumerability
   - Many-one reductions
3. Undecidability of Turing machine halting
4. Turing machine halting $\preceq \mathtt{SR}$

Synthetic Computability

The Problem with "Undecidable" in a Proof Assistant

Classically: *no Turing machine decides P.*

To state this in Rocq we would need to:

- Formalize Turing machines
- Formalize what it means for a TM to "compute" a Rocq predicate
- Drag this machinery through *every* proof

Synthetic Approach [Forster et al. 2020]

Use **Rocq functions** as the model of computation.
Every constructively definable `f : X -> bool` *is* computable.

Decidability

```
Definition reflects (b : bool) (p : Prop) :=
  p <-> b = true.

Definition decider {X} (f : X -> bool) (P : X -> Prop) :=
  forall x, reflects (f x) (P x).

Definition decidable {X} (P : X -> Prop) :=
  exists f : X -> bool, decider f P.
```

- `decidable P`: *some* Rocq function pointwise reflects P
- no machines, no encodings
- f is total and computable by virtue of existing

Enumerability

```
Definition enumerator {X} (f : nat -> option X) (P : X -> Prop) :=
  forall x, P x <-> exists n, f n = Some x.

Definition enumerable {X} (P : X -> Prop) :=
  exists f, enumerator f P.

Definition complement {X} (P : X -> Prop) :=
  fun x => not (P x).
```

Classically familiar facts hold synthetically

- `decidable P` $\Rightarrow$ `enumerable P` (on enumerable types)
- Turing machine halting is enumerable
- String rewriting is enumerable

Many-One Reductions

```
Definition reduction {X Y} (f : X -> Y) (P : X -> Prop) (Q : Y -> Prop) :=
  forall x, P x <-> Q (f x).

Definition reduces {X Y} (P : X -> Prop) (Q : Y -> Prop) :=
  exists f : X -> Y, reduction f P Q.

Notation "P ⪯ Q" := (reduces P Q).
```

- `f`: instance translation, again automatically computable
- correctness: an *iff*, for every instance
- $\preceq$ is a **preorder** (reflexive, transitive)

Facts that Hold Classically *and* Synthetically

Statement	**Exercise**
`decidable P` $\Rightarrow$ `decidable (complement P)`	1.2
`decidable P` $\Rightarrow$ `decidable Q` $\Rightarrow$ `decidable (P` $\wedge$ `Q)`	1.3
$P \preceq P$	2.1
$P \preceq Q \Rightarrow Q \preceq R \Rightarrow P \preceq R$	2.2
$P \preceq Q \Rightarrow$ `decidable Q` $\Rightarrow$ `decidable P`	2.3
$P \preceq Q \Rightarrow$ `complement P` $\preceq$ `complement Q`	2.4

Needs Markov's principle (not provable in axiom-free Rocq)

`enumerable P` $\wedge$ `enumerable (complement P)` $\Rightarrow$ `decidable P`

Summary

Summary

- synthetic computability: `decidable`, `enumerable`
- many-one reduction $\preceq$

Exercise session (now):

1. decidability of simple predicates
2. enumerability of simple predicates
3. decidability facts
4. $\preceq$ facts

Bibliography I

 Forster, Yannick et al. (2020). "A Coq library of undecidable problems." In: *CoqPL 2020 The Sixth International Workshop on Coq for Programming Languages*.