Mechanizing Undecidable Properties of Rewriting Systems using the Rocq Prover

Lecture 3: Undecidability of Turing Machine Halting

Andrej Dudenhefner

ISR 2026 – 15th International School on Rewriting

2026-07-12

Course Roadmap

1. Formalizing string rewriting (SR) in Rocq
2. Synthetic computability
3. **Undecidability of Turing machine halting**
   - Turing machine halting (SBTM_HALT)
   - Synthetic undecidability
   - Undecidability of SBTM_HALT
4. Turing machine halting $\preceq$ SR

Simple Binary Turing Machines

The Machine Model

Minimalist by design: one tape, binary alphabet `bool`, blanks are `false`.

```
Definition mv (d: direction) (t: list bool * bool * list bool) :=
  match d with
  | go_left  => match t with
      | (l :: ls, a, rs) => (ls, l, a :: rs)
      | (nil,     a, rs) => (nil, false, a :: rs)  (* blank in *)
      end
  | go_right => ...  (* symmetric *)
  end.
```

- Tape: `(ls, a, rs)`, left part (reversed), head symbol, right part
- Implicit infinite blanks on both ends, a fresh `false` appears at an edge
- No separate "halting state": halting is *absence* of a transition

States and Transitions

```
Record SBTM := Build_SBTM {
  num_states : nat;
  trans : Vector.t (
    (option (Fin.t num_states * bool * direction)) *   (* on true  *)
    (option (Fin.t num_states * bool * direction)))    (* on false *)
    num_states }.
```

States are `Fin.t num_states`: a *finite*, statically sized type.

```
Definition trans' M : state M * bool -> option (state M * bool * direction) :=
  fun '(q, a) => (if a then fst else snd) (Vector.nth (trans M) q).
```

- `trans' M (q, a) = Some (q', a', d)`: write `a'`, move `d`, go to `q'`
- `trans' M (q, a) = None`: **no move**, the machine halts

Stepping and Halting

```
Definition step (M: SBTM) : config M -> option (config M) :=
  fun '(q, (ls, a, rs)) =>
    match trans' M (q, a) with
    | None => None
    | Some (q', a', d) => Some (q', mv d (ls, a', rs))
    end.

Definition steps (M: SBTM) (k: nat) (x: config M) : option (config M) :=
  Nat.iter k (obind (step M)) (Some x).
```

- `step` returns `None` exactly when `trans'` does
- `steps M k x`: iterate k times, propagating `None` via `obind`
- once `None`, always `None` (`steps_plus`)

The halting problem

```
SBTM_HALT (existT _ M x) := exists k, steps M k x = None
```

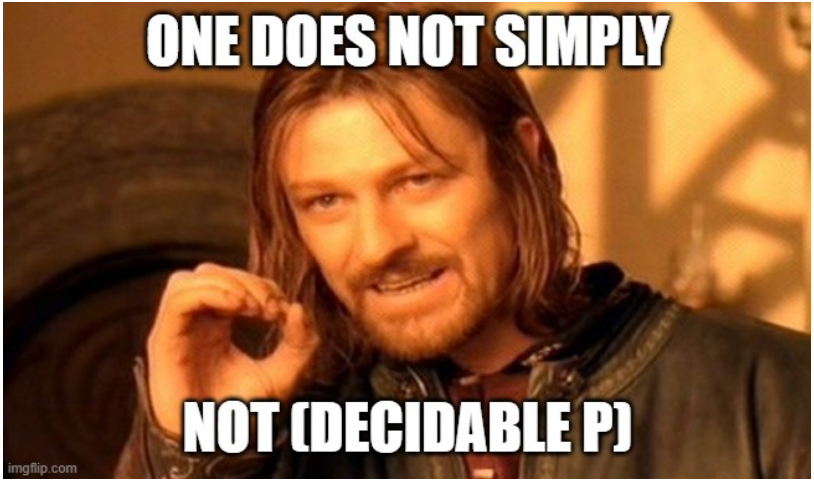
Synthetic Undecidability

The Definition of undecidable

```
Definition undecidable {X} (p : X -> Prop) := not (decidable p).
```

The Definition of `undecidable`

```
Definition undecidable {X} (p : X -> Prop) := not (decidable p).
```



Why is `not (decidable p)` Problematic?

Rocq's type theory is **consistent with classical axioms**,
e.g. functional forms of choice + excluded middle, under which *every* predicate is decidable.

- `SBTM_HALT`: halting of simple binary Turing machines
- `not (decidable SBTM_HALT)` is *not provable* and would contradict such axioms
- We prove *conditional* statements:
  - Decidability of p implies something we believe is not provable

The chosen statement: `enumerable (complement SBTM_HALT)`

- `SBTM_HALT` *is* enumerable (run all machines for all step counts).
- If its complement were enumerable too, halting would be decidable (Post theorem).
- Meta-level: this is exactly the classical unsolvability of the halting problem.

Why enumerable (complement SBTM_HALT) Suffices

```
Definition undecidable {X} (p : X -> Prop) :=
  decidable p -> enumerable (complement SBTM_HALT).
```

Read: *If p were decidable, then the complement of TM halting would be enumerable.*
Chain of consequences if some `undecidable p` had a decider:

$$\texttt{decidable p} \Rightarrow \texttt{enumerable (complement SBTM\_HALT)}$$

Combined with enumerability of `SBTM_HALT` and Post theorem:

$$\Rightarrow \texttt{decidable SBTM\_HALT}$$

- Outside the system: a Rocq term deciding halting would yield an actual algorithm. Absurd by classical computability.
- Inside the system: "co-enumerability of halting" is an axiom-free proxy for falsity.

The Workhorse Lemma

```
Lemma undecidability_from_reducibility {X p Y q} :
  undecidable p  ->  p ⪯ q  ->  undecidable q.
Proof.
  intros H [f Hf] [d Hd]. eapply H.
  exists (fun x => d (f x)).
  intros x. rewrite Hf. eapply Hd.
Qed.
```

Intuition: a decider d for q plus a reduction f gives the decider $d \circ f$ for p.

This is the entire engine of the course. Every undecidability result will be:

1. build f (a Rocq function)
2. certify `forall x, p x <-> q (f x)` (the hard part!)
3. `apply undecidability_from_reducibility`

Undecidability Proofs

This is how every `_undec.v` proof looks.

```
Theorem reduction : SBTM_HALT ⪯ SR.
Proof.
  (* ... *)
Qed.

Lemma SR_undec : undecidable SR.
Proof.
  apply (undecidability_from_reducibility SBTM_HALT_undec).
  exact reduction.
Qed.
```

Reductions **compose** (`reduces_transitive`). The library of undecidability proofs[1] is a growing DAG of problems, rooted in TM halting.

[1] `https://github.com/uds-psl/coq-library-undecidability`

Undecidability of SBTM_HALT

The Root of the DAG

Every undecidable problem in the library reduces to `SBTM_HALT`.

Recall the actual definition

```
undecidable p := decidable p -> enumerable (complement SBTM_HALT)
```

So `undecidable SBTM_HALT` unfolds to

```
decidable SBTM_HALT ⇒ enumerable (complement SBTM_HALT).
```

Theorem (Post theorem)

A problem p is decidable, iff p is enumerable and the complement of p is enumerable

Corollary

If `SBTM_HALT` is decidable, then `complement SBTM_HALT` is enumerable.

Summary

Summary

- `SBTM_HALT` is the *root* of the undecidability DAG
- Direct proof of undecidability of `SBTM_HALT` by Post theorem
- We *do not assume* that there is no decider for `SBTM_HALT`
- We *propagate* the unprovable consequence `enumerable (complement SBTM_HALT)`
- `undecidable p` really means `decidable p -> enumerable (complement SBTM_HALT)`, an axiom-free proxy for falsity

Exercise session (now):

1. `SBTM` facts
2. Undecidability facts
3. Post theorem