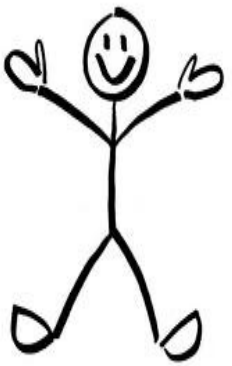
Reductions and Rice's Theorem

Announcements

- Welcome back! I hope you had a great break!
- Problem Set 8 out, due Friday, December 2 at 2:15 PM.
  - Stop by office hours with questions!
  - Email **cs103@cs.stanford.edu** with questions!
- Problem session tonight, 7-8PM, in 370-370.
  - Alternate time: 7-8PM tomorrow, Gates 100.

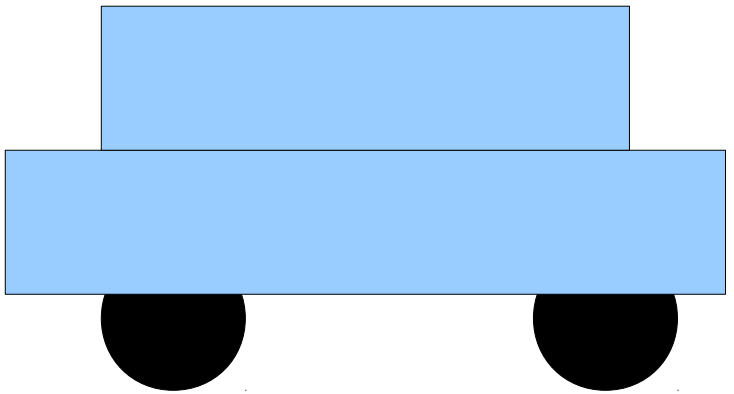
Reductions

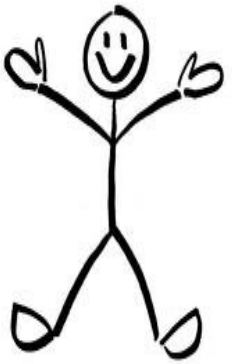
A Simple Reduction



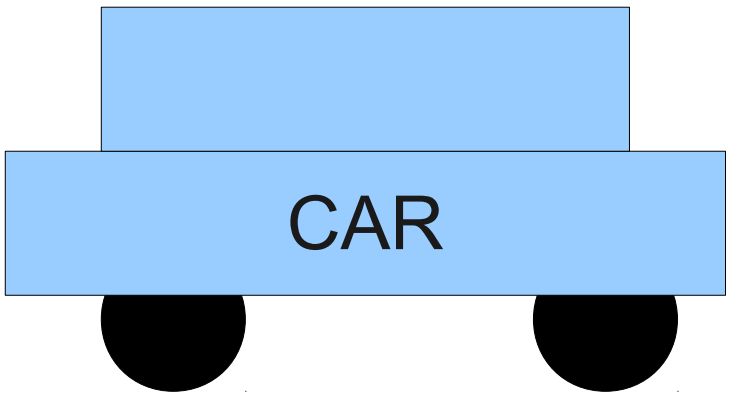
Thanks to David Dill for the idea

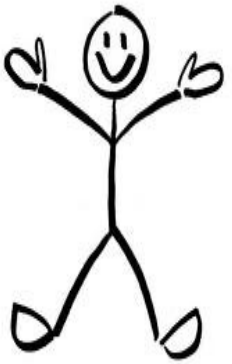
A Simple Reduction



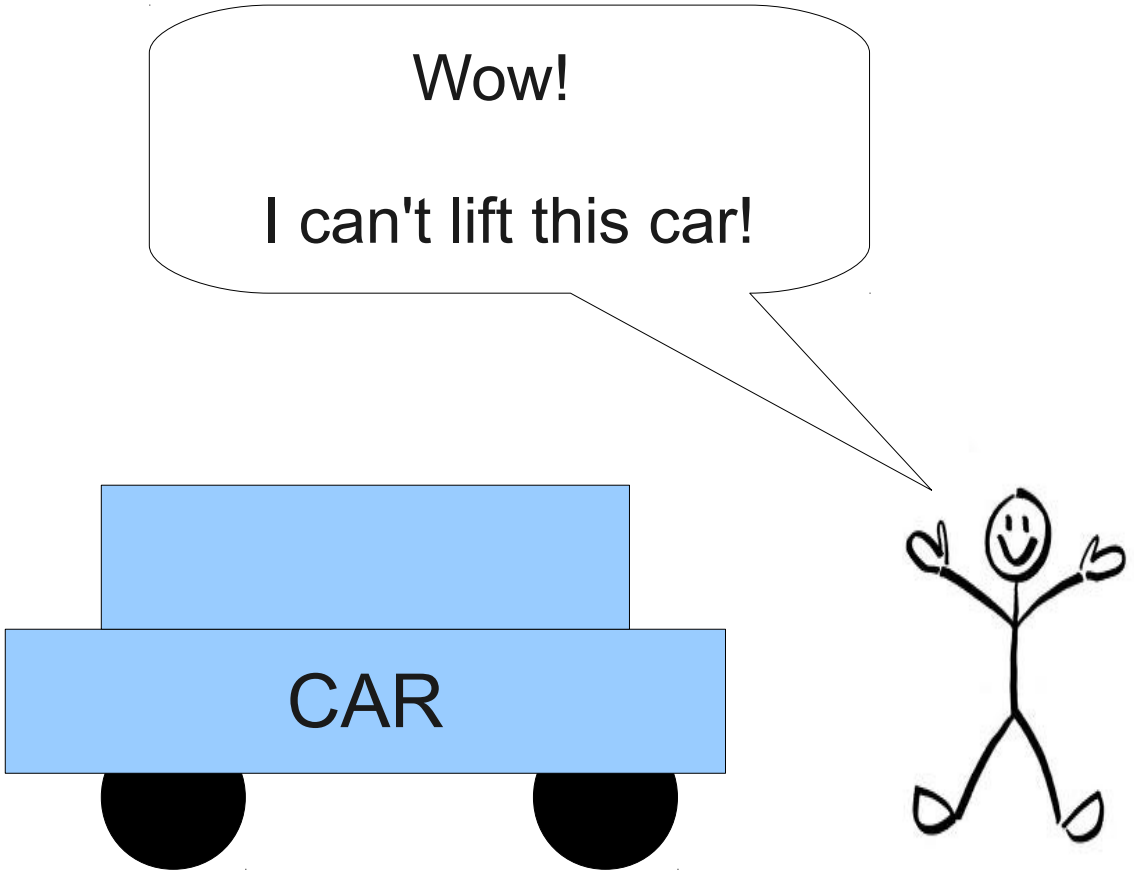


A Simple Reduction

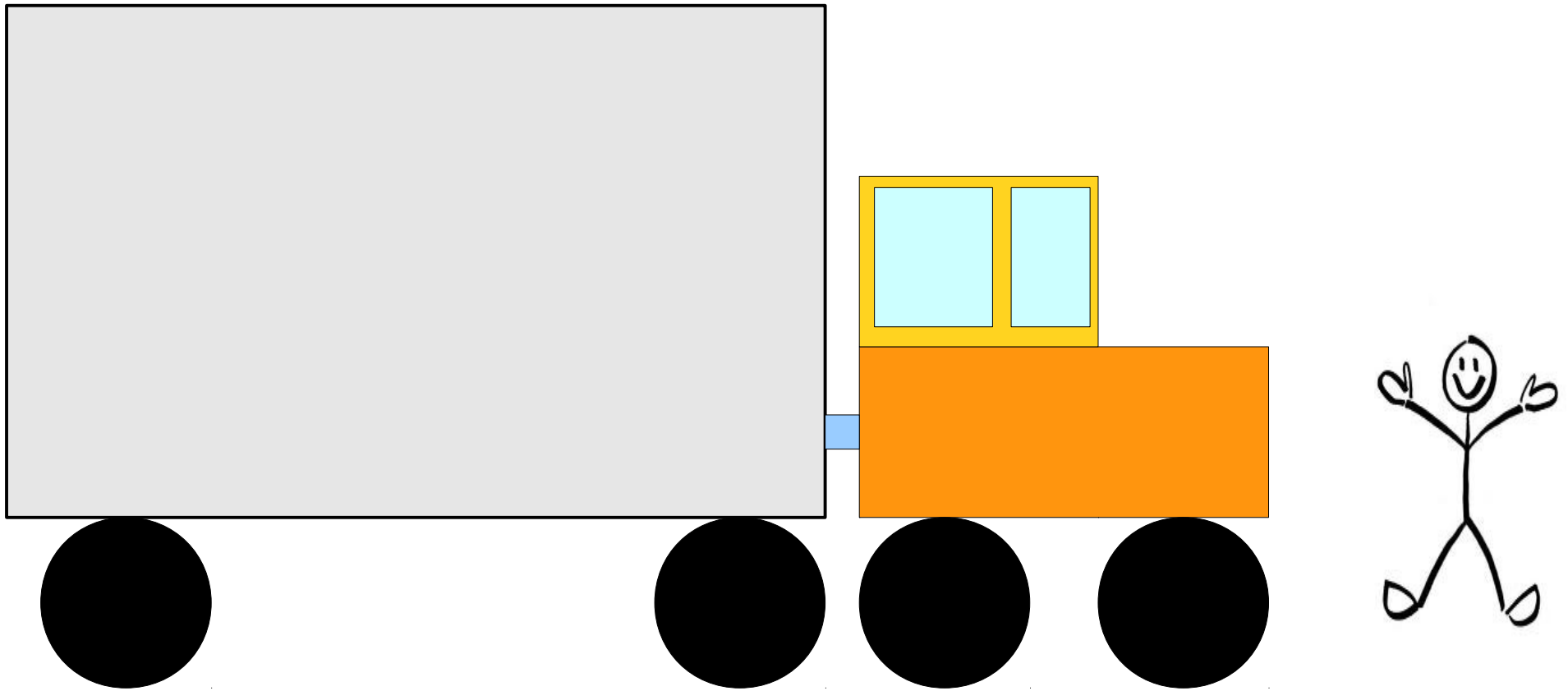




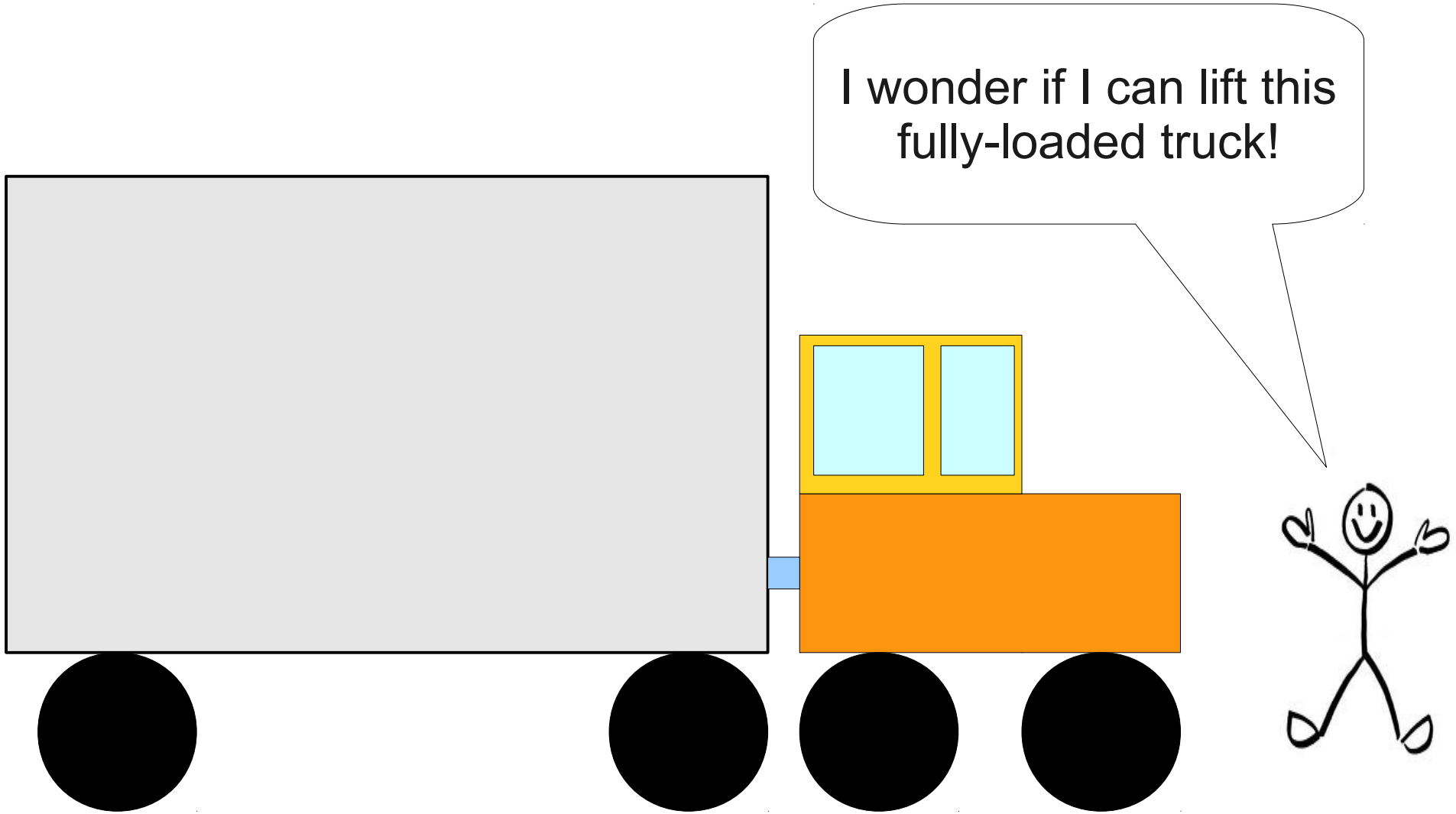
A Simple Reduction



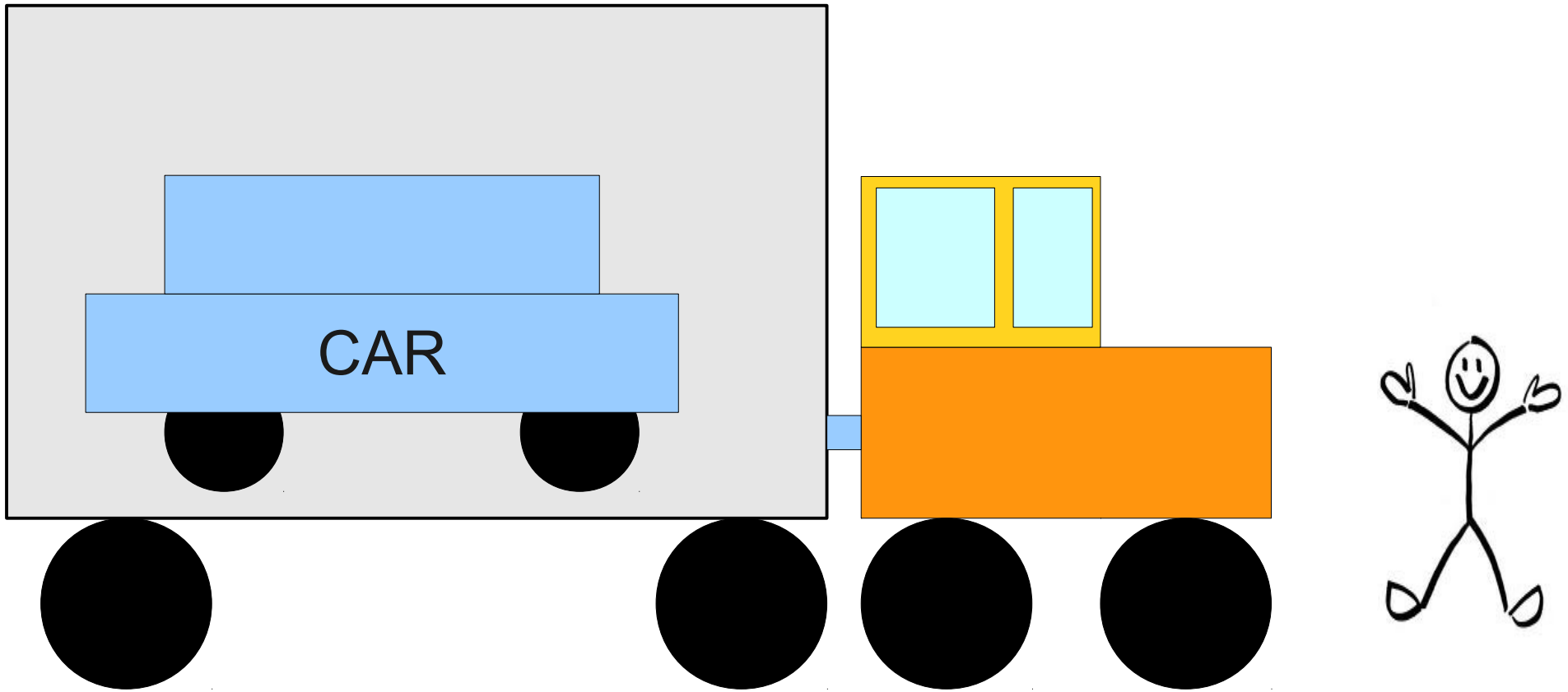
A Simple Reduction



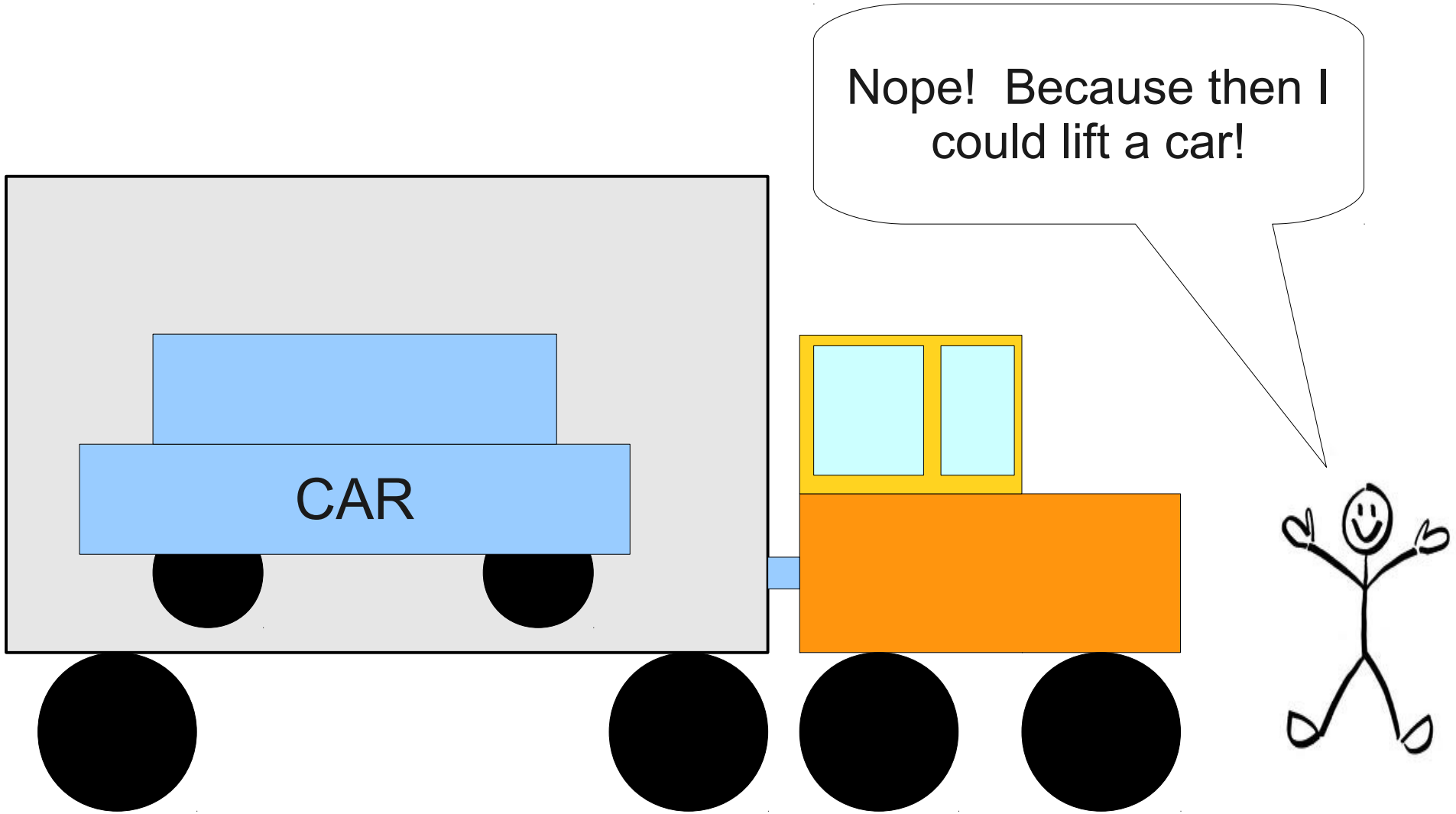
A Simple Reduction



A Simple Reduction



A Simple Reduction



Reductions

- A **reduction** is a way of solving a problem EASY with a problem HARDER.
- In this example, we could solve the problem of "lift the car" by *reducing* it to the harder problem of "lift the truck."

Reductions

- A **reduction** is a way of solving a problem EASY with a problem HARDER.
- In this example, we could solve the problem of "lift the car" by *reducing* it to the harder problem of "lift the truck."

Reductions

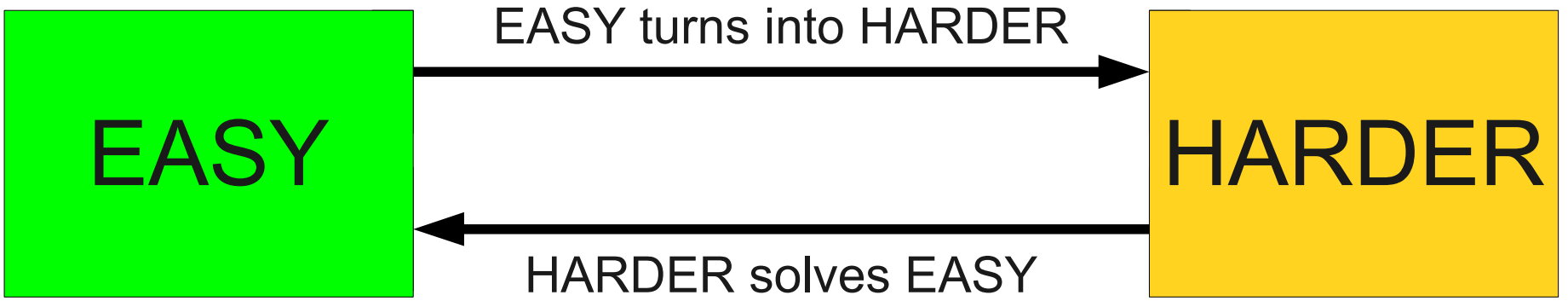
- A **reduction** is a way of solving a problem EASY with a problem HARDER.
- In this example, we could solve the problem of "lift the car" by *reducing* it to the harder problem of "lift the truck."

Reductions

- A **reduction** works by turning an instance of EASY into an instance of HARDER.
- In this example, we *reduce* the problem of lifting the car by putting it into a truck and lifting the truck.

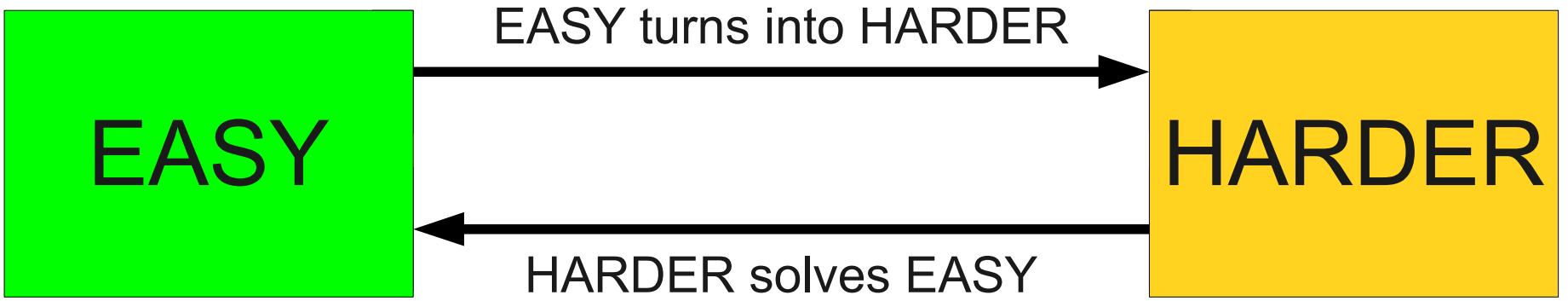
Reductions

- A **reduction** works by turning an instance of EASY into an instance of HARDER.
- In this example, we *reduce* the problem of lifting the car by putting it into a truck and lifting the truck.



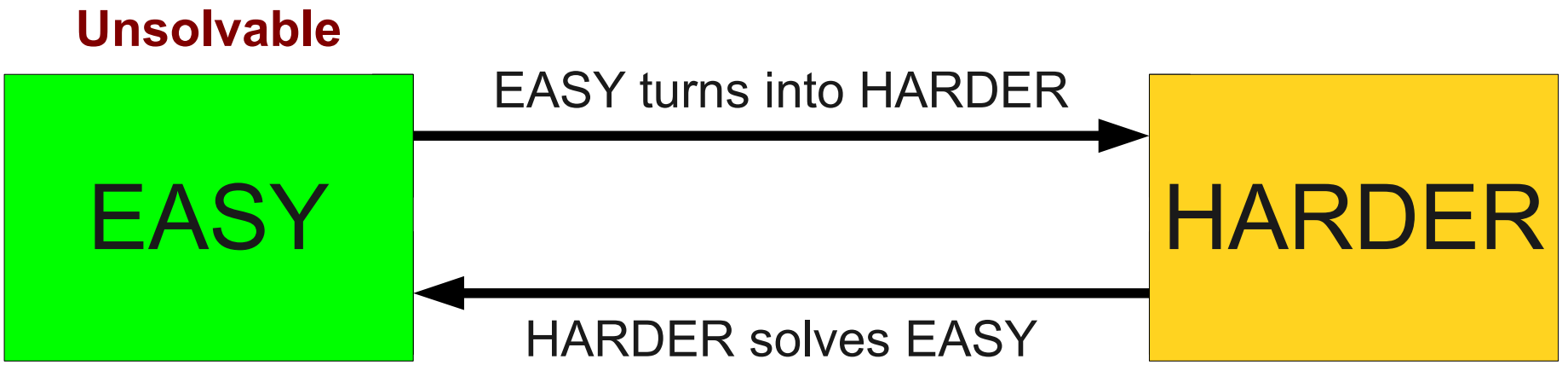
Reductions

- Suppose that we cannot solve EASY.
- If we can **reduce** EASY to HARDER, we cannot solve HARDER either.



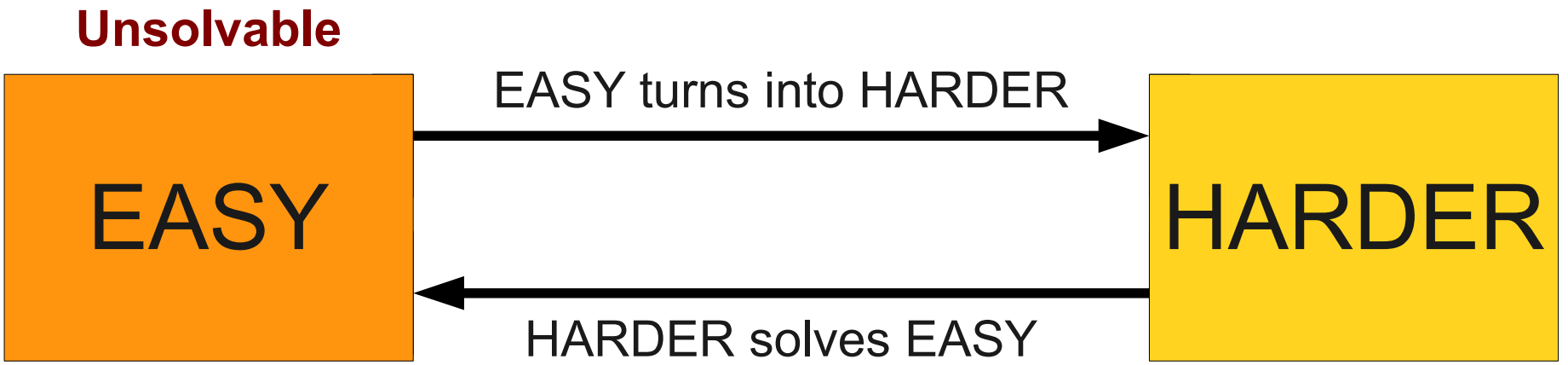
Reductions

- Suppose that we cannot solve EASY.
- If we can **reduce** EASY to HARDER, we cannot solve HARDER either.



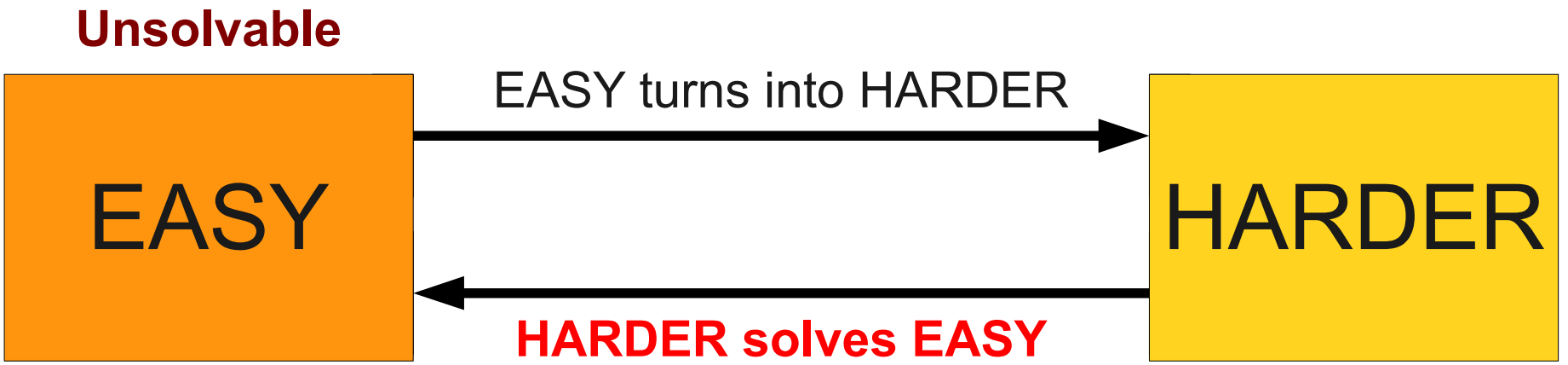
Reductions

- Suppose that we cannot solve EASY.
- If we can **reduce** EASY to HARDER, we cannot solve HARDER either.



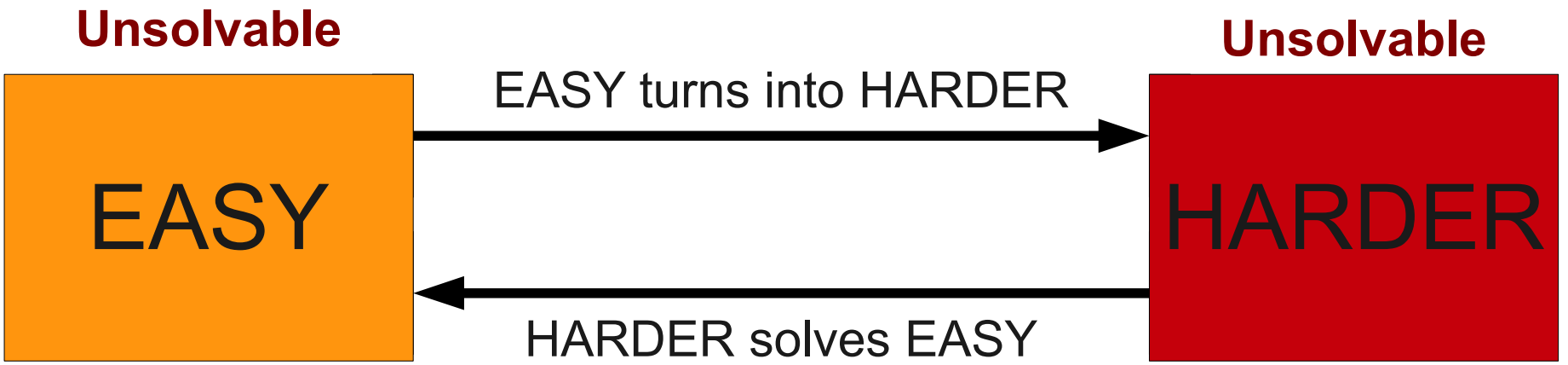
Reductions

- Suppose that we cannot solve EASY.
- If we can **reduce** EASY to HARDER, we cannot solve HARDER either.



Reductions

- Suppose that we cannot solve EASY.
- If we can **reduce** EASY to HARDER, we cannot solve HARDER either.



Reductions and Decidability

- Suppose we want to prove that some problem HARDER is undecidable.
- If we can **reduce** an undecidable problem HARD to HARDER, then we know that we cannot decide HARDER.

Reductions and Decidability

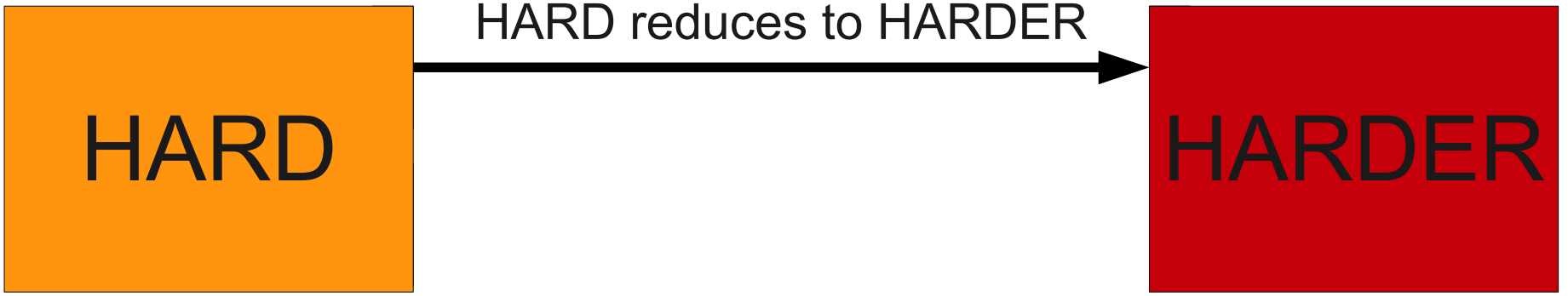
- Suppose we want to prove that some problem HARDER is undecidable.
- If we can **reduce** an undecidable problem HARD to HARDER, then we know that we cannot decide HARDER.

Reductions and Decidability

- Suppose we want to prove that some problem HARDER is undecidable.
- If we can **reduce** an undecidable problem HARD to HARDER, then we know that we cannot decide HARDER.

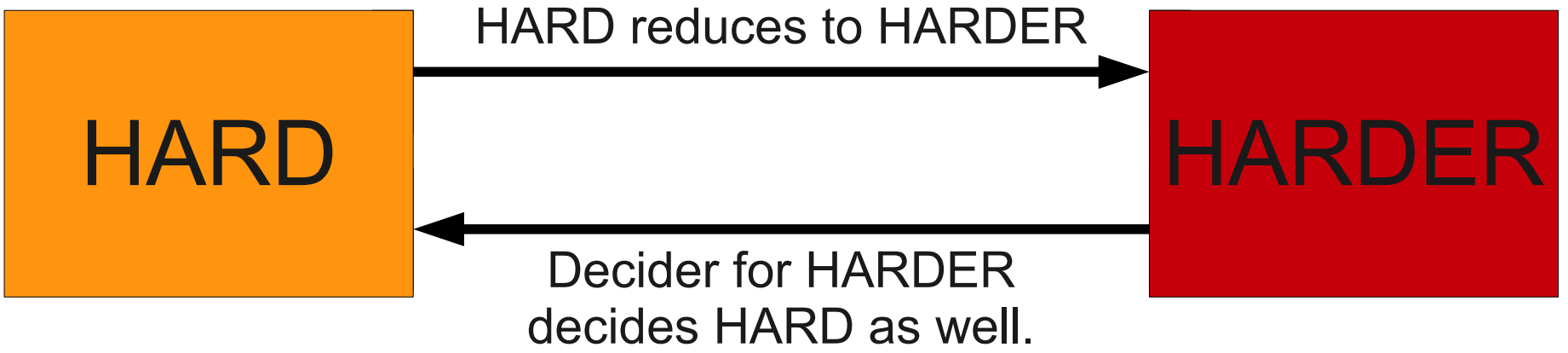
Reductions and Decidability

- Suppose we want to prove that some problem HARDER is undecidable.
- If we can **reduce** an undecidable problem HARD to HARDER, then we know that we cannot decide HARDER.



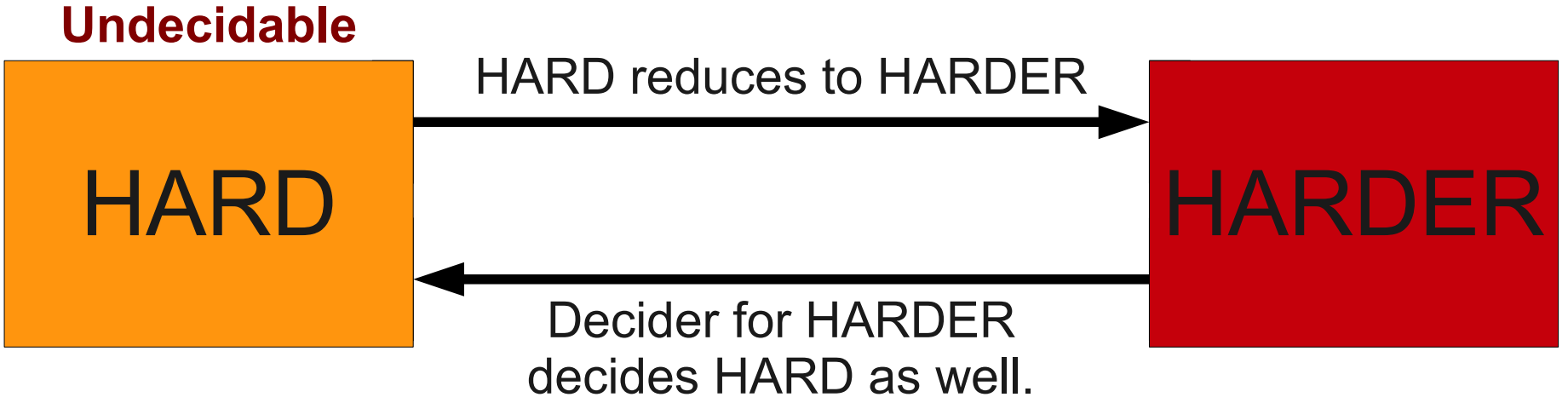
Reductions and Decidability

- Suppose we want to prove that some problem HARDER is undecidable.
- If we can **reduce** an undecidable problem HARD to HARDER, then we know that we cannot decide HARDER.



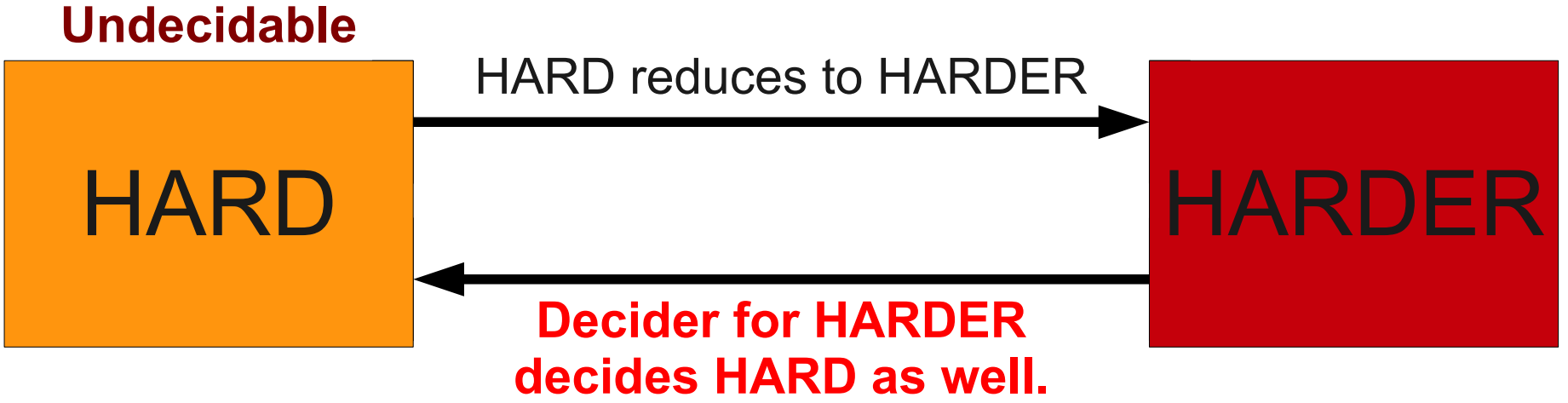
Reductions and Decidability

- Suppose we want to prove that some problem HARDER is undecidable.
- If we can **reduce** an undecidable problem HARD to HARDER, then we know that we cannot decide HARDER.



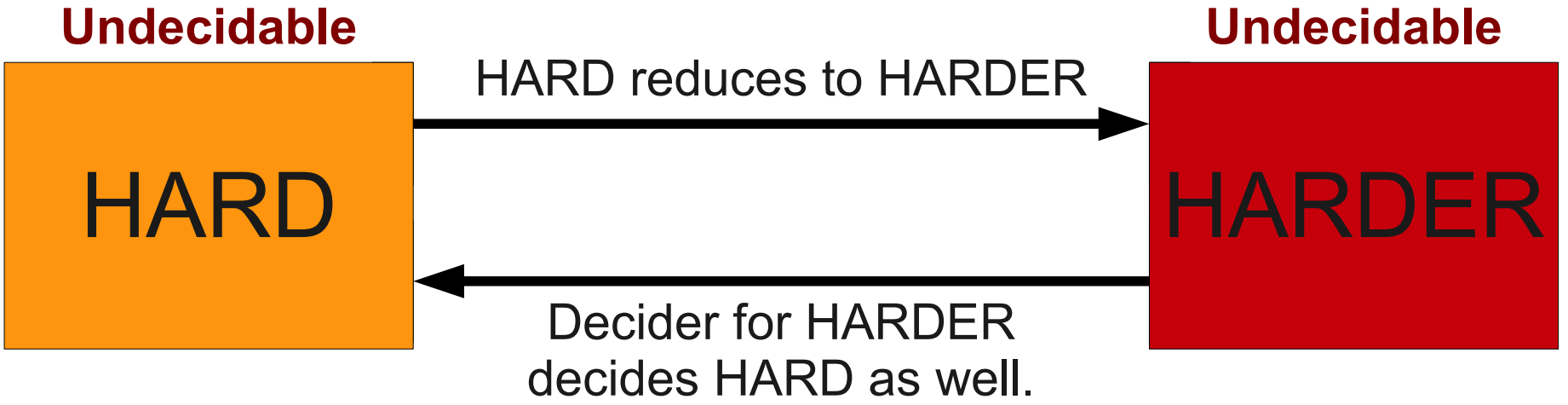
Reductions and Decidability

- Suppose we want to prove that some problem HARDER is undecidable.
- If we can **reduce** an undecidable problem HARD to HARDER, then we know that we cannot decide HARDER.



Reductions and Decidability

- Suppose we want to prove that some problem HARDER is undecidable.
- If we can **reduce** an undecidable problem HARD to HARDER, then we know that we cannot decide HARDER.



Reduction from the Halting Problem

- The language *HALT* is RE but not recursive:

$$HALT = \{ \langle M, w \rangle \mid M \text{ halts on } w \}$$

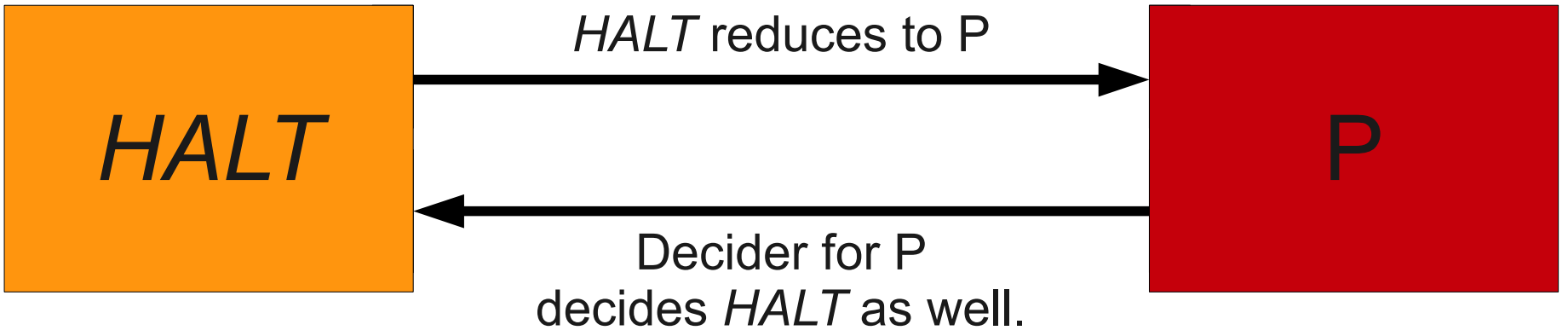
- If we can **reduce** *HALT* to some problem P, then that new problem cannot be decidable.

Reduction from the Halting Problem

- The language *HALT* is RE but not recursive:

$$HALT = \{ \langle M, w \rangle \mid M \text{ halts on } w \}$$

- If we can **reduce** *HALT* to some problem P, then that new problem cannot be decidable.

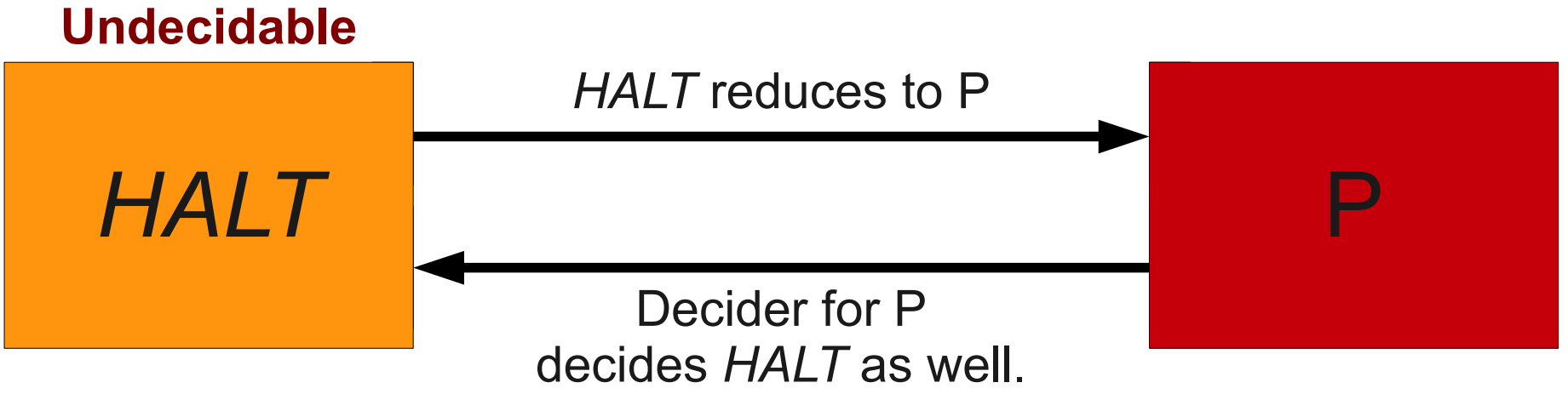


Reduction from the Halting Problem

- The language *HALT* is RE but not recursive:

$$HALT = \{ \langle M, w \rangle \mid M \text{ halts on } w \}$$

- If we can **reduce** *HALT* to some problem P, then that new problem cannot be decidable.

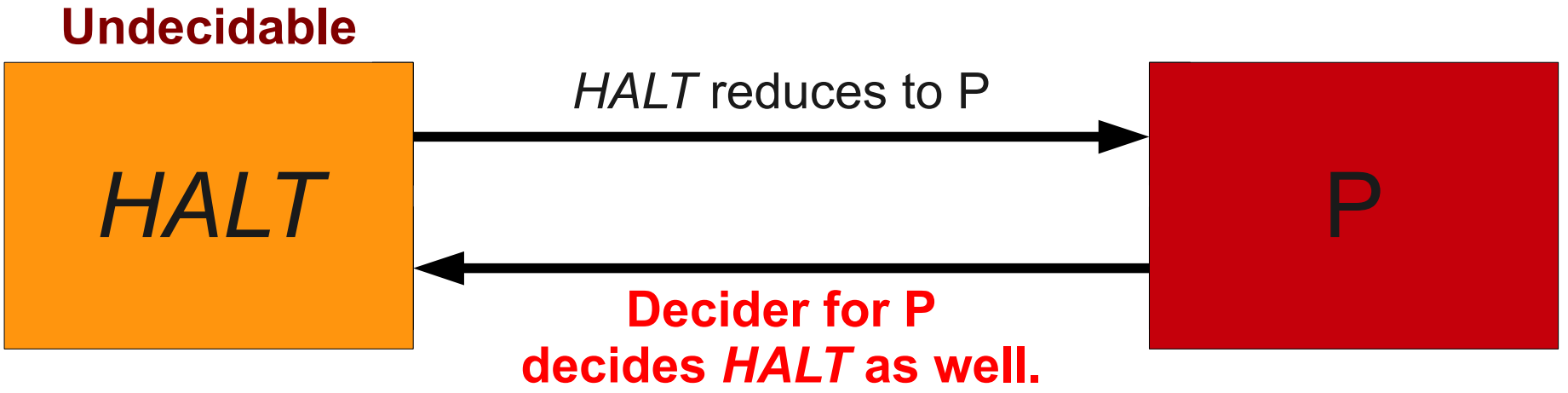


Reduction from the Halting Problem

- The language *HALT* is RE but not recursive:

$$HALT = \{ \langle M, w \rangle \mid M \text{ halts on } w \}$$

- If we can **reduce** *HALT* to some problem P, then that new problem cannot be decidable.

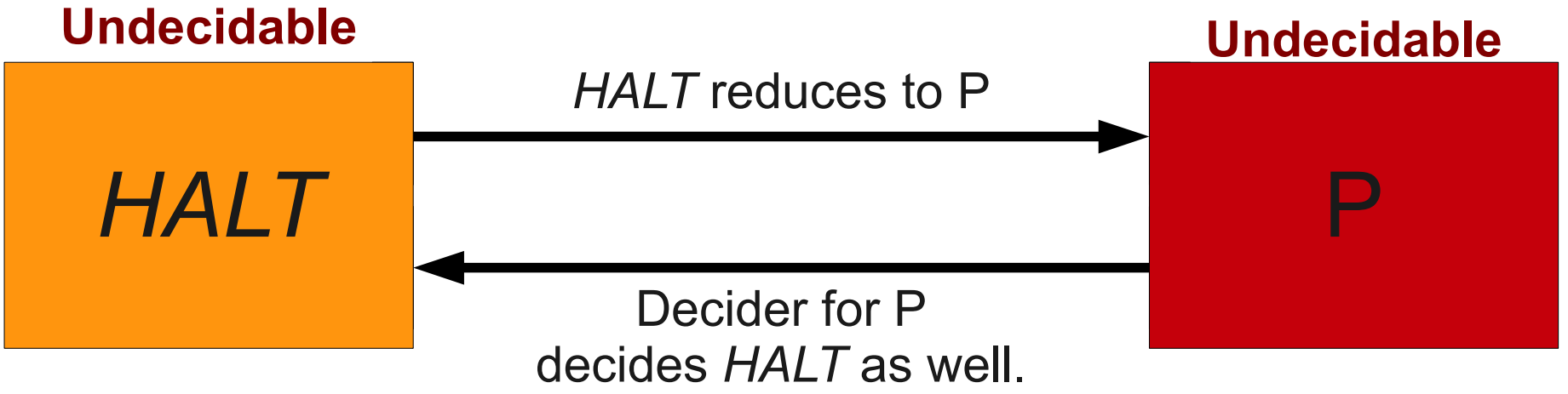


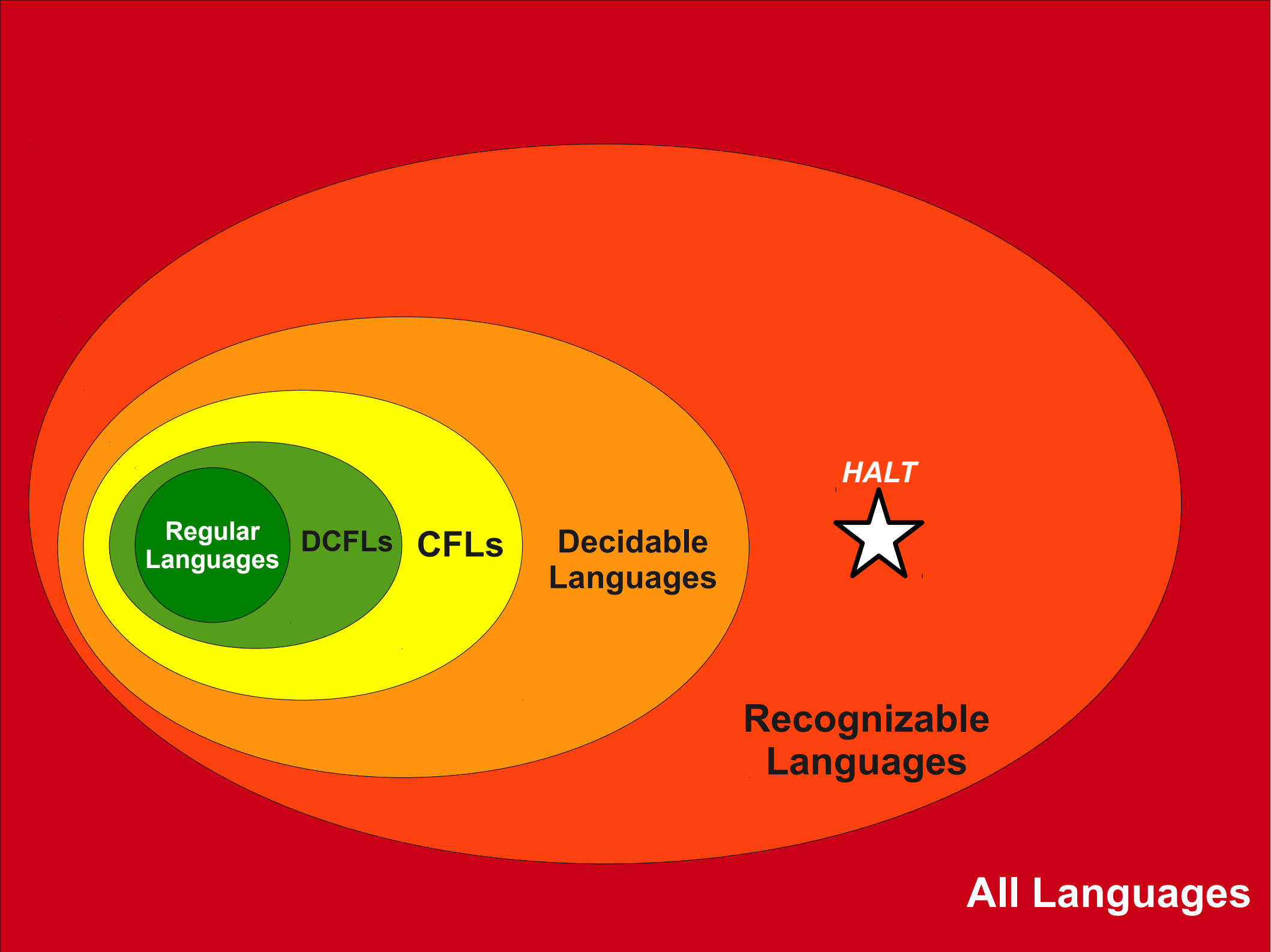
Reduction from the Halting Problem

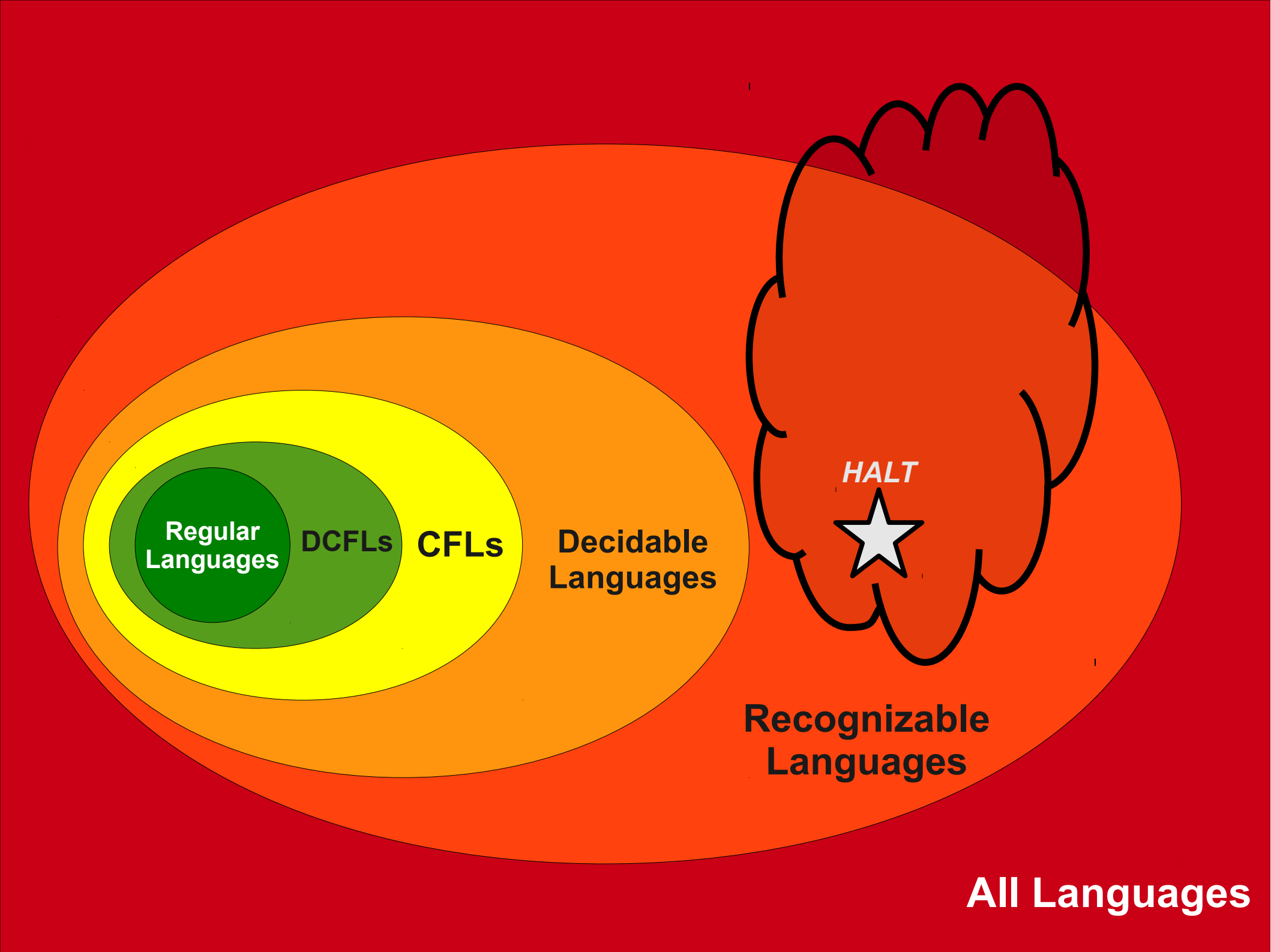
- The language *HALT* is RE but not recursive:

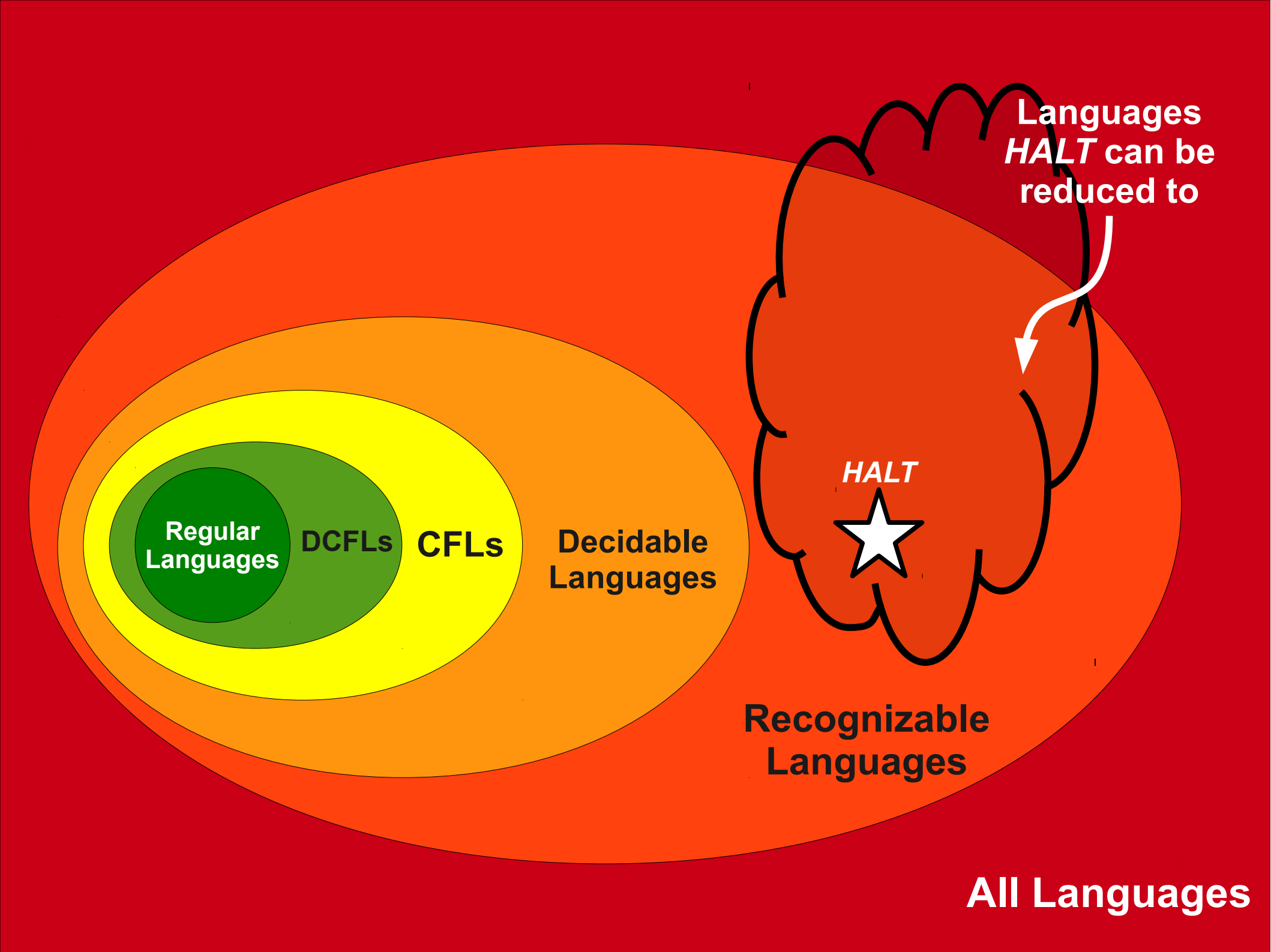
$$HALT = \{ \langle M, w \rangle \mid M \text{ halts on } w \}$$

- If we can **reduce** *HALT* to some problem P, then that new problem cannot be decidable.




Regular Languages
DCFLs
CFLs
Decidable Languages
HALT
Recognizable Languages
All Languages


Regular Languages
DCFLs
CFLs
Decidable Languages
HALT
Recognizable Languages
All Languages


Languages
HALT can be
reduced to
HALT
Regular
Languages
DCFLs
CFLs
Decidable
Languages
Recognizable
Languages
All Languages

Proof by Reduction

- Suppose that we are given a language L that we believe is undecidable.
- We can prove that this is true using the following technique:
  - Assume, for the sake of contradiction, that L is decidable.
  - Show how a decider for L could be used to construct a decider for an undecidable language.
  - Conclude that L must not be decidable.

The Totality Problem

- Last time we saw that there was no way to convert an arbitrary recognizer into a decider.
- However, is there a way to tell whether a given TM is a recognizer or a decider?
- This is called the **totality problem**.

The Totality Problem

- Let

$$TOTAL = \{ \langle M \rangle \mid M \text{ halts on all inputs.} \}$$

- This seems at least as hard as the halting problem.
- How would we prove that this is undecidable?

The Totality Problem

- Let

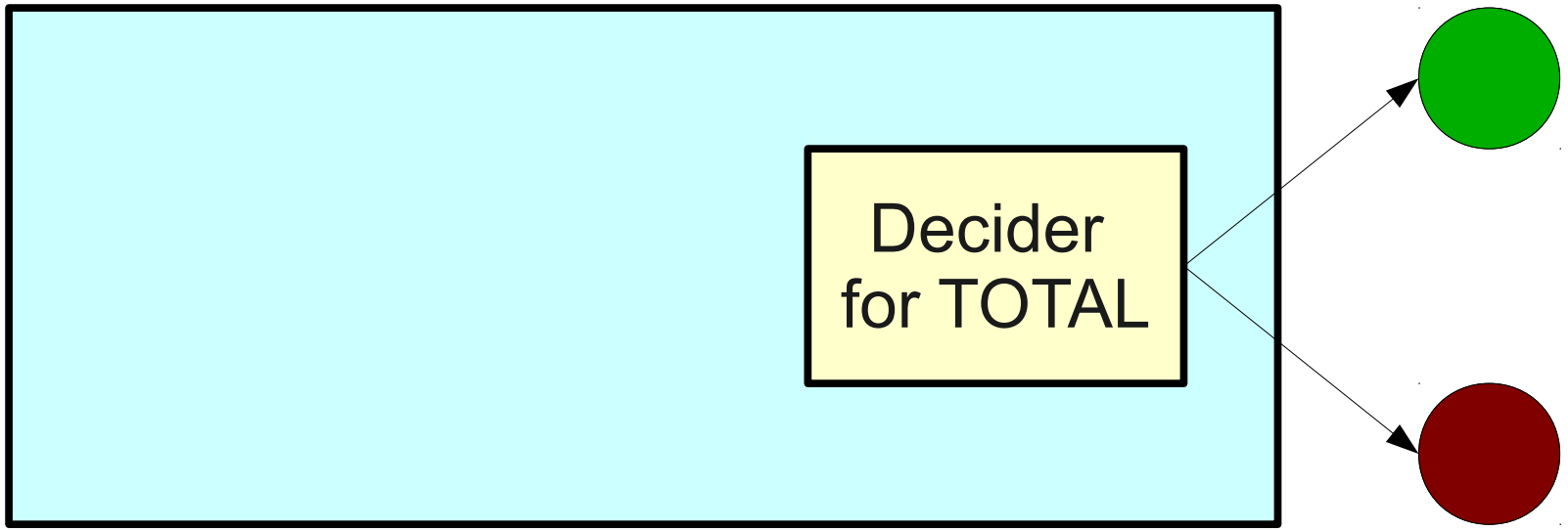
$$TOTAL = \{ \langle M \rangle \mid M \text{ halts on all inputs.} \}$$

- This seems at least as hard as the halting problem.
- How would we prove that this is undecidable?
- **Idea**: Reduce *HALT* to *TOTAL*.
- Show how a decider for *TOTAL* could be used to build a decider for *HALT*.
- Conclude that no such decider can exist.

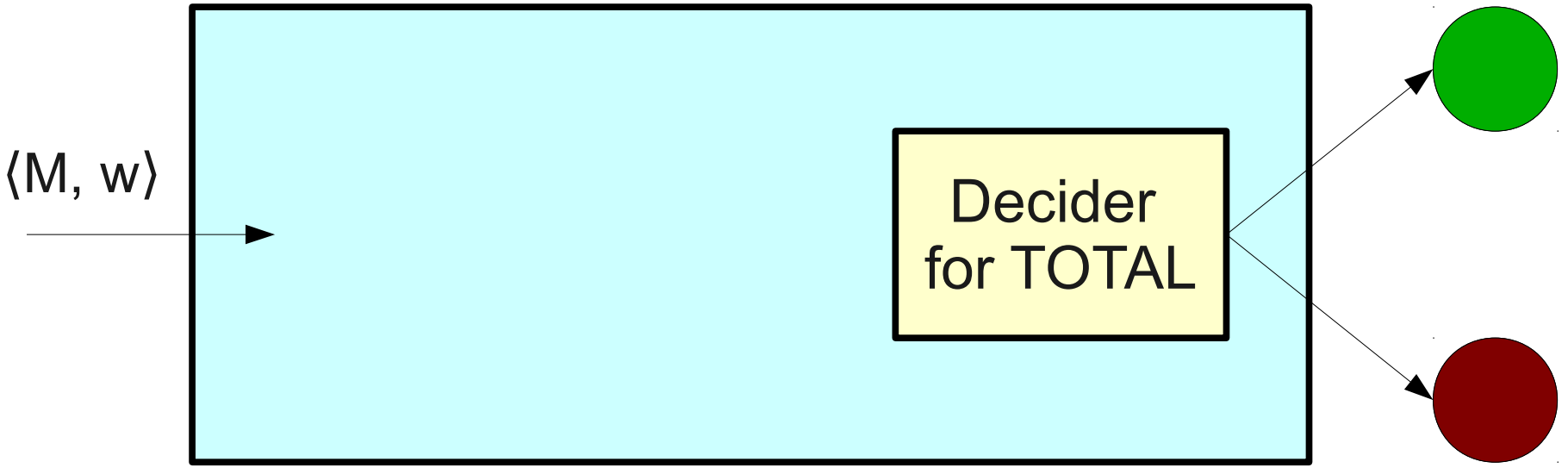
The Totality Problem

Assume, for the sake of contradiction, that TOTAL is decidable.

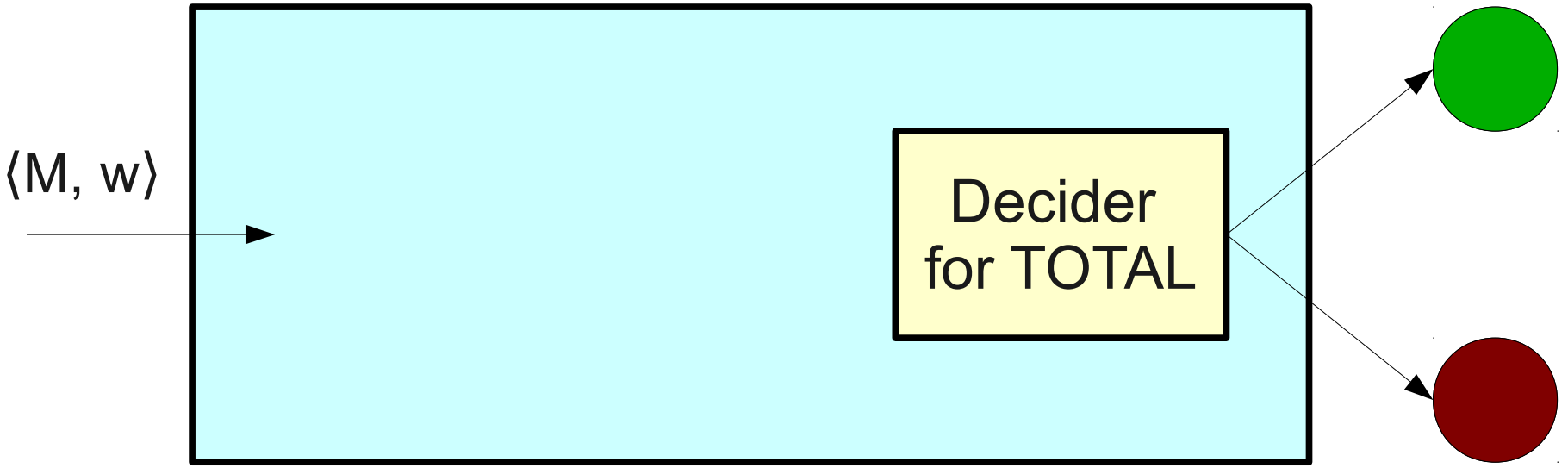
The Totality Problem



The Totality Problem

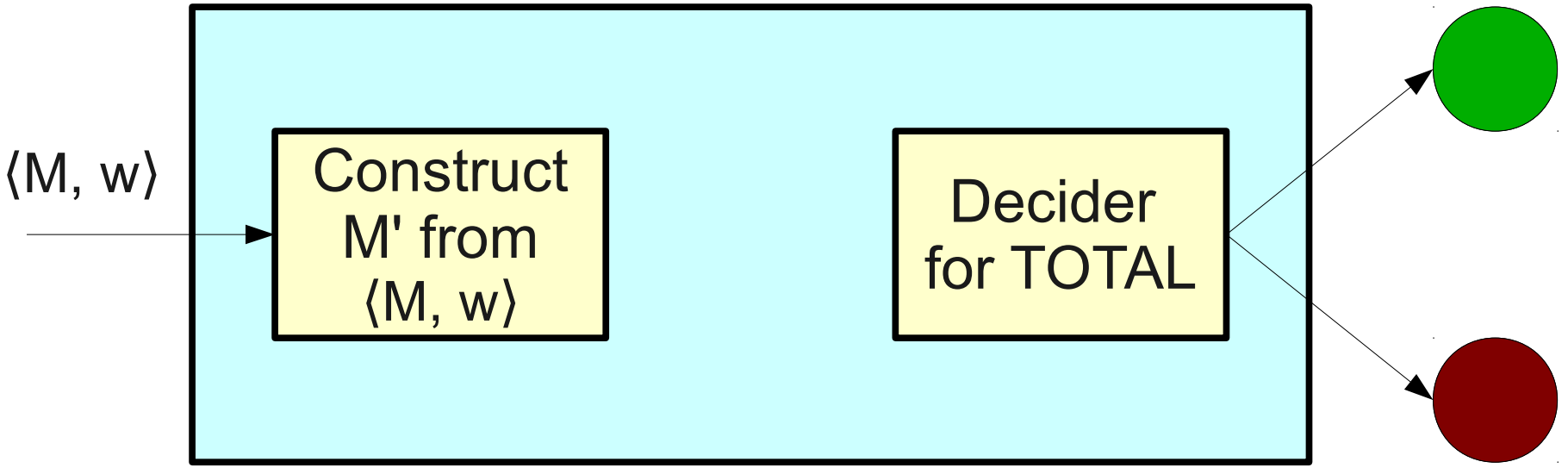


The Totality Problem

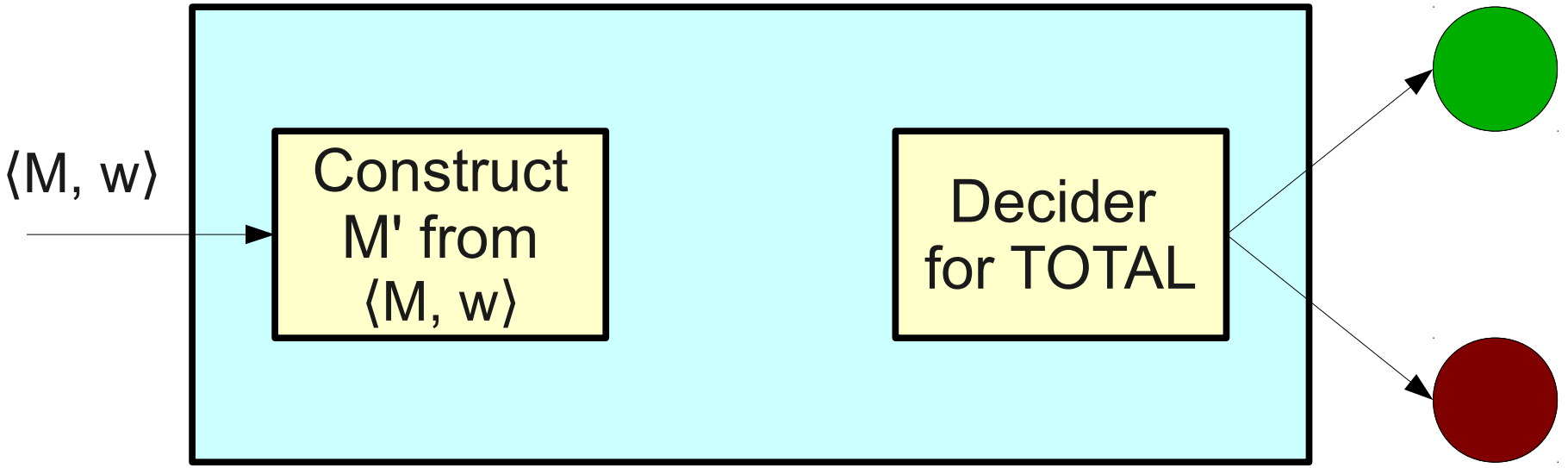


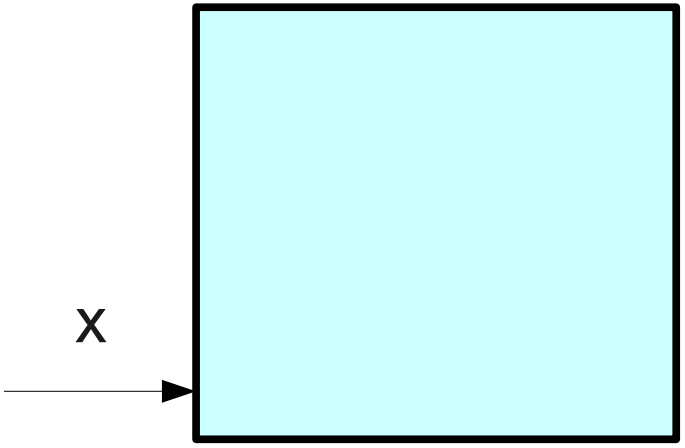
We're going to try to show how a decider for TOTAL decides HALT, so our input must be a TM and a string.

The Totality Problem

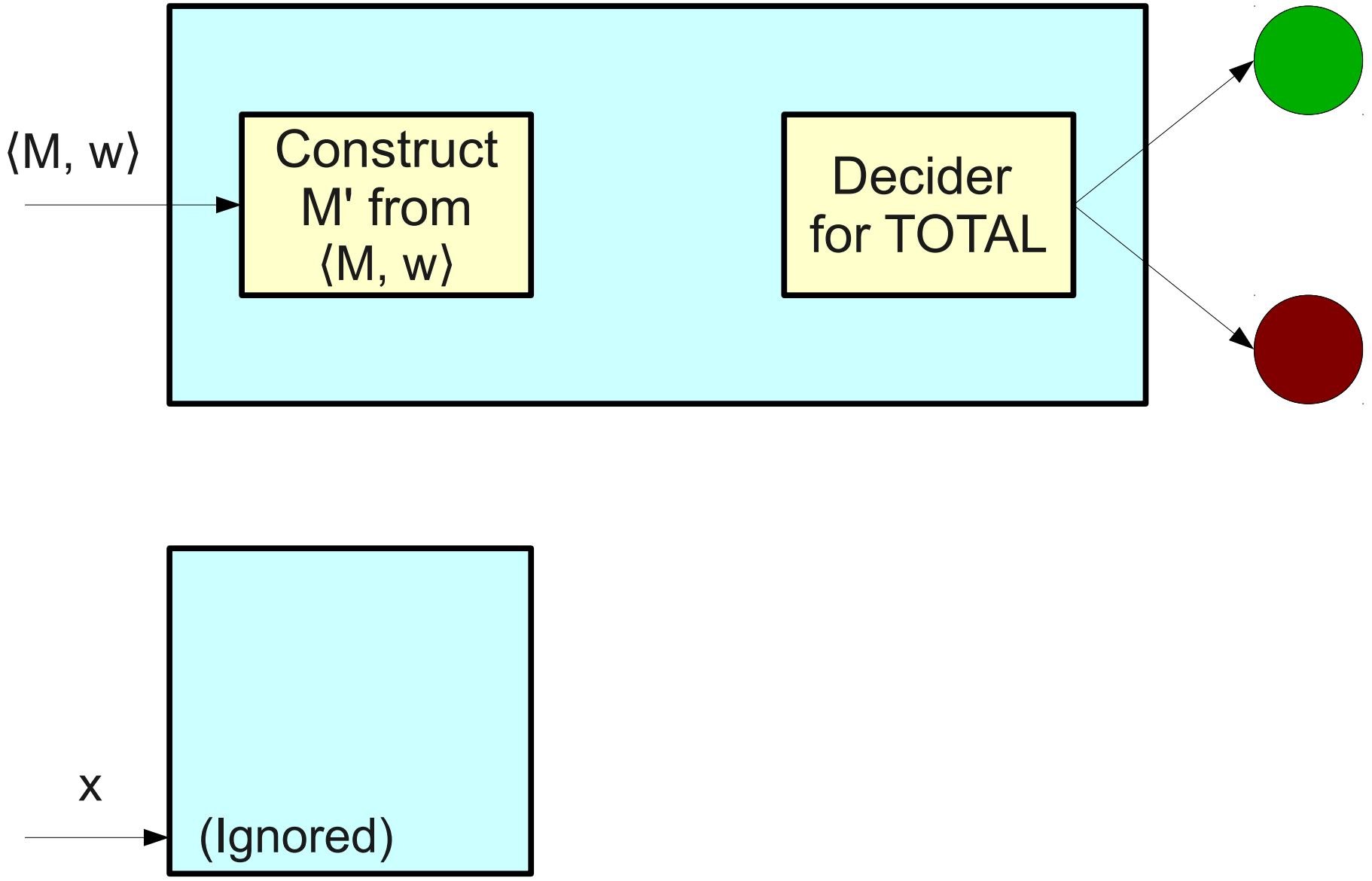


The Totality Problem

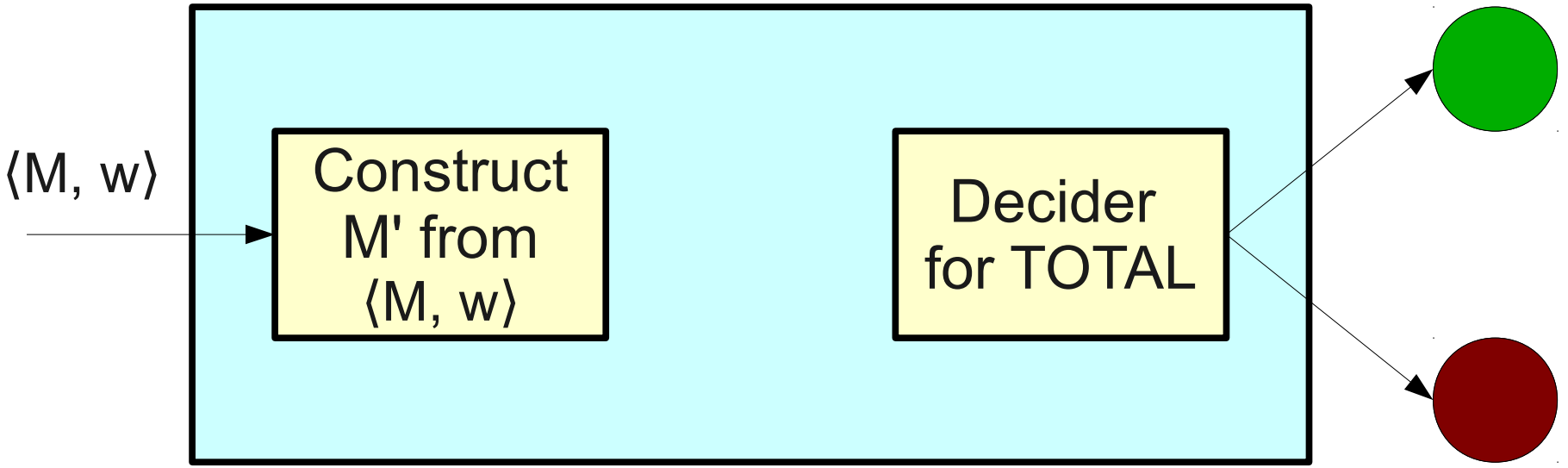


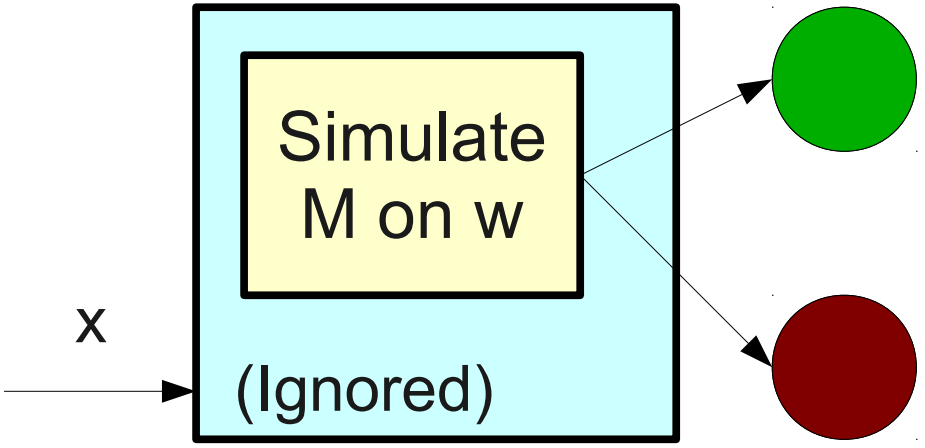


The Totality Problem

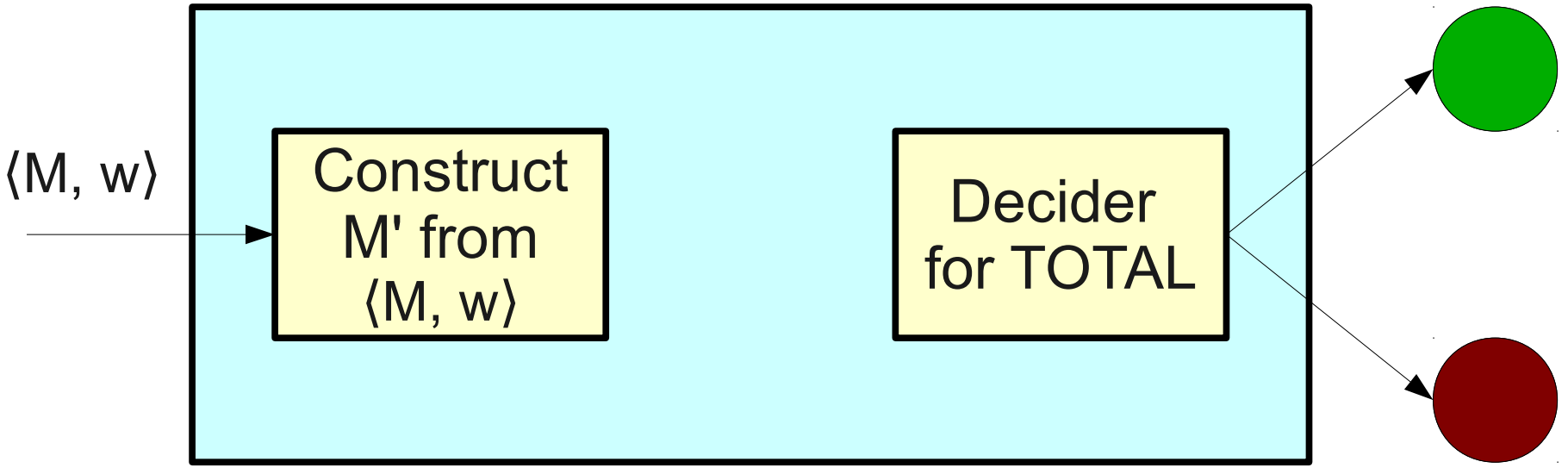


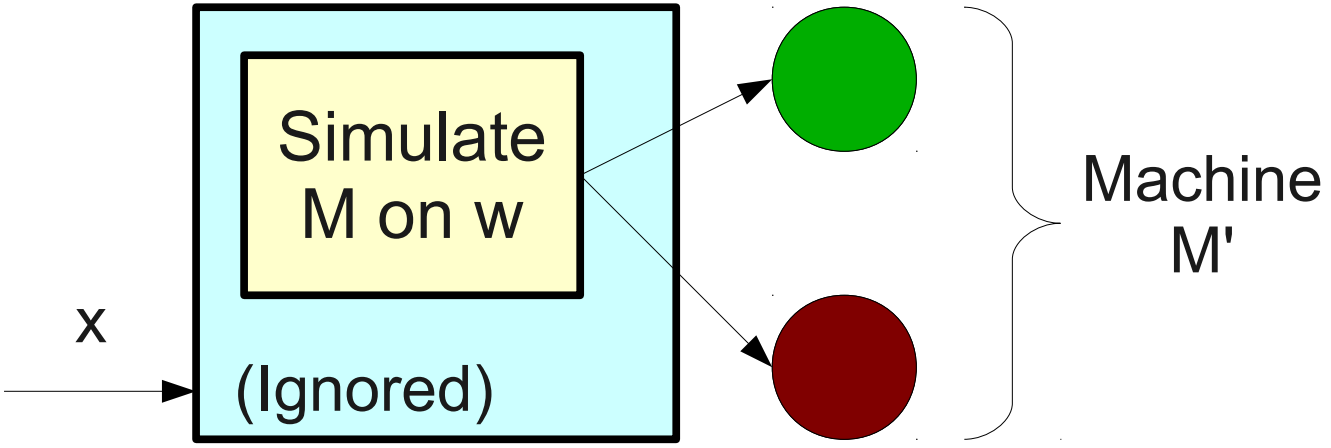
The Totality Problem



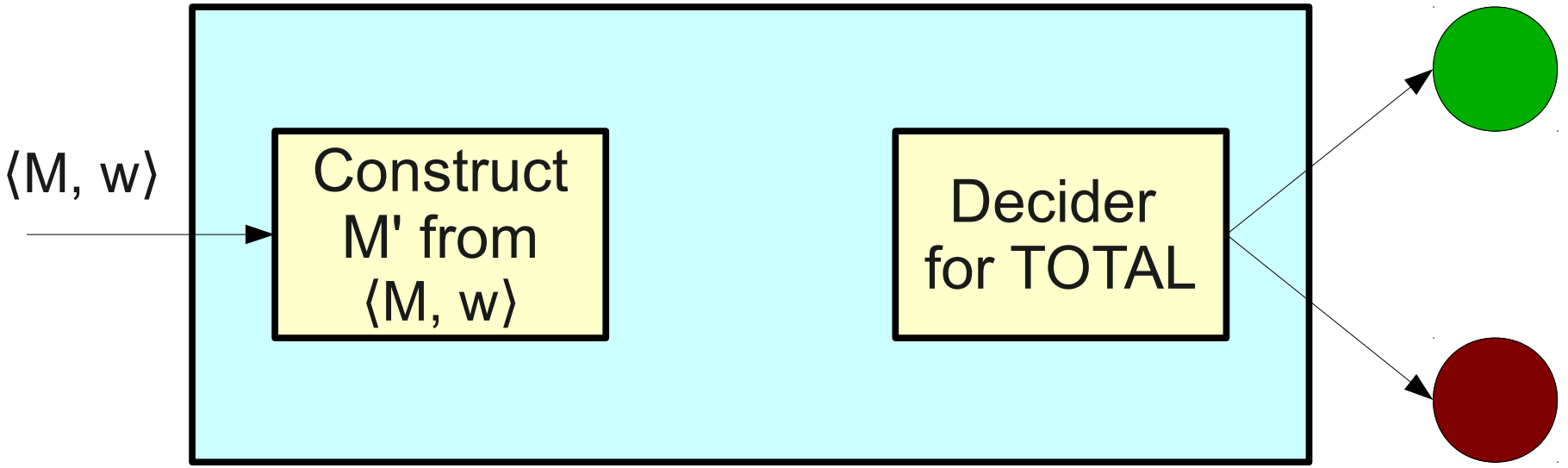


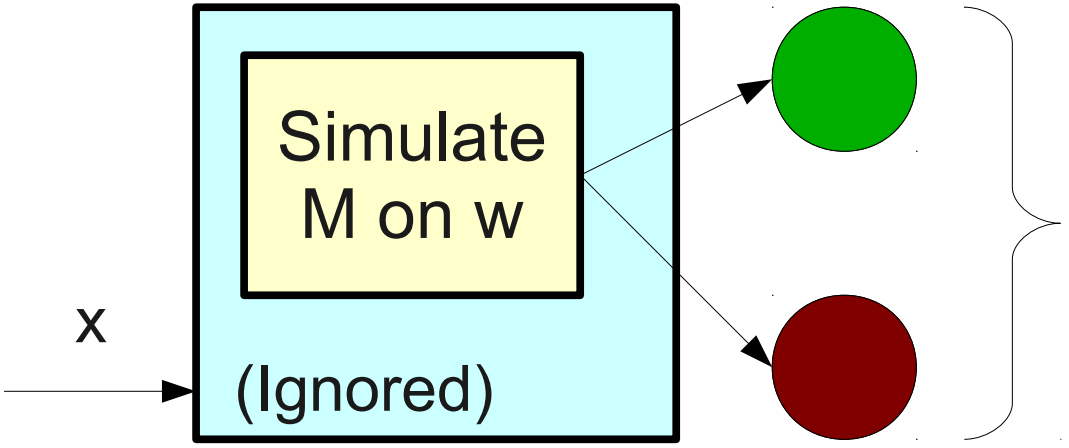
The Totality Problem





The Totality Problem

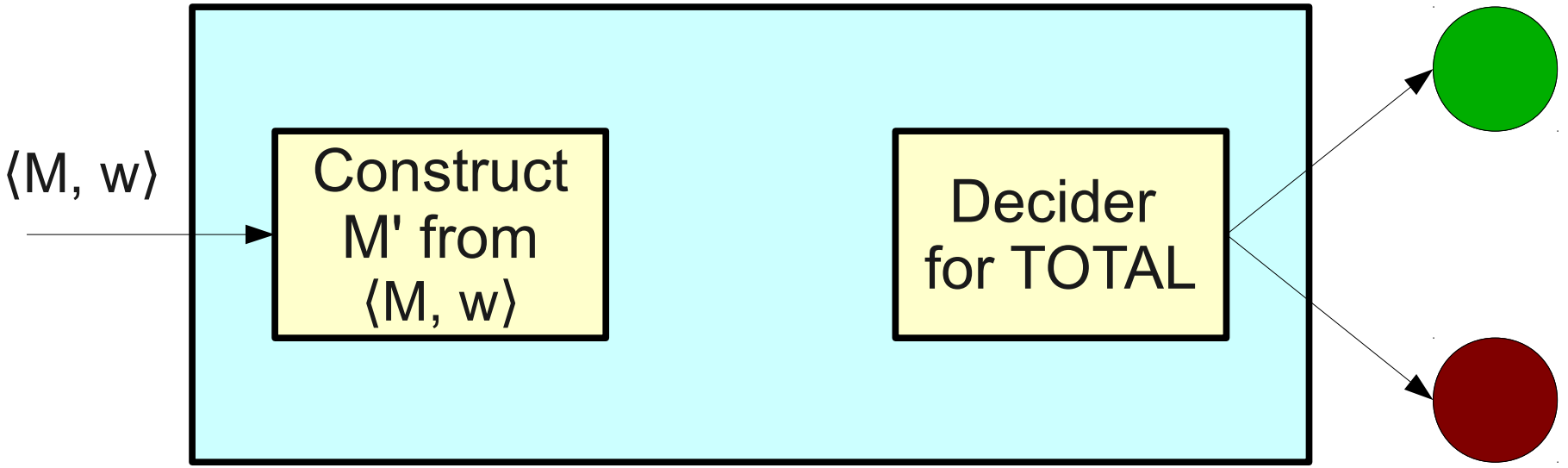


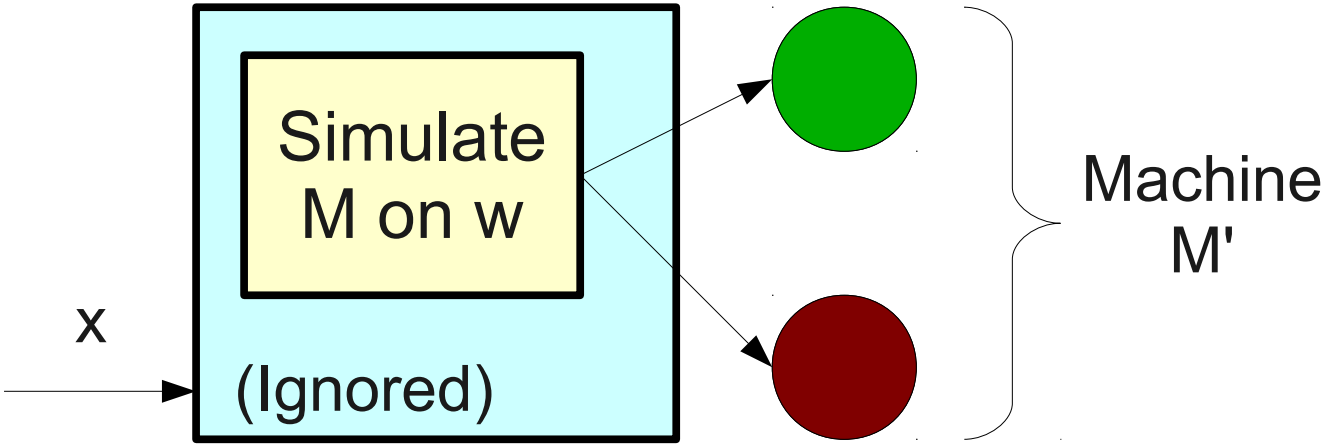


Machine M'

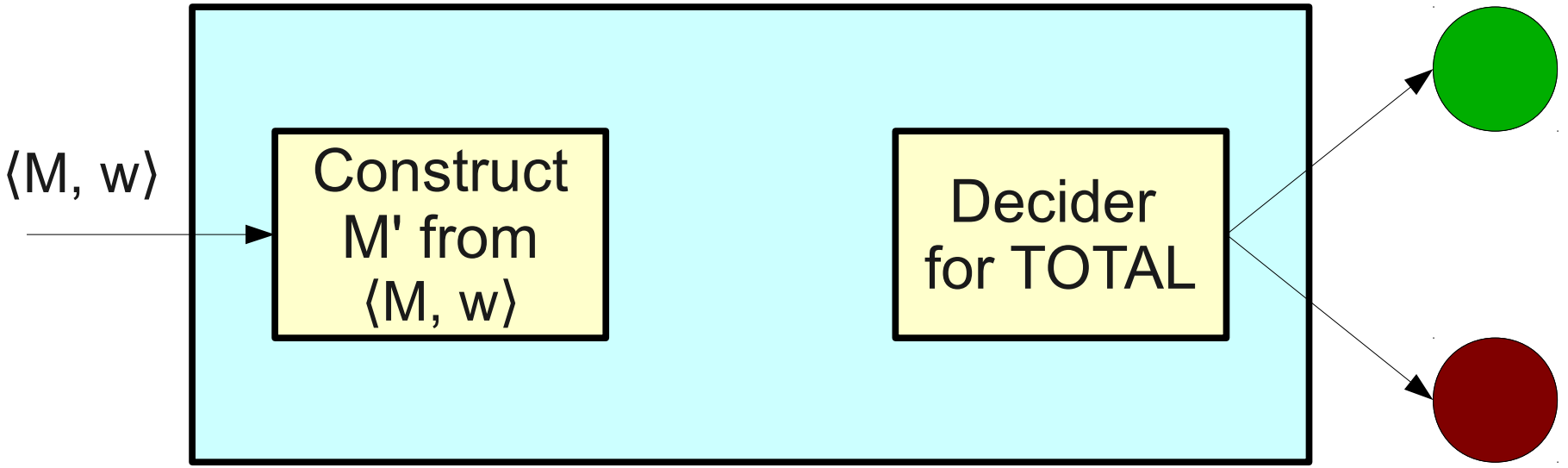
M' = “On input x:
Ignore x.
Run M on w.
If M accepts w, accept.
If M rejects w, reject.”

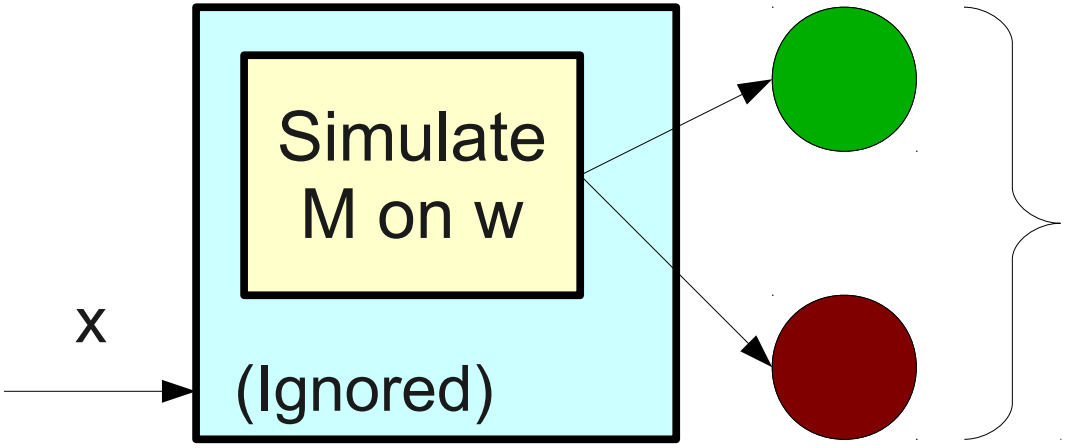
The Totality Problem





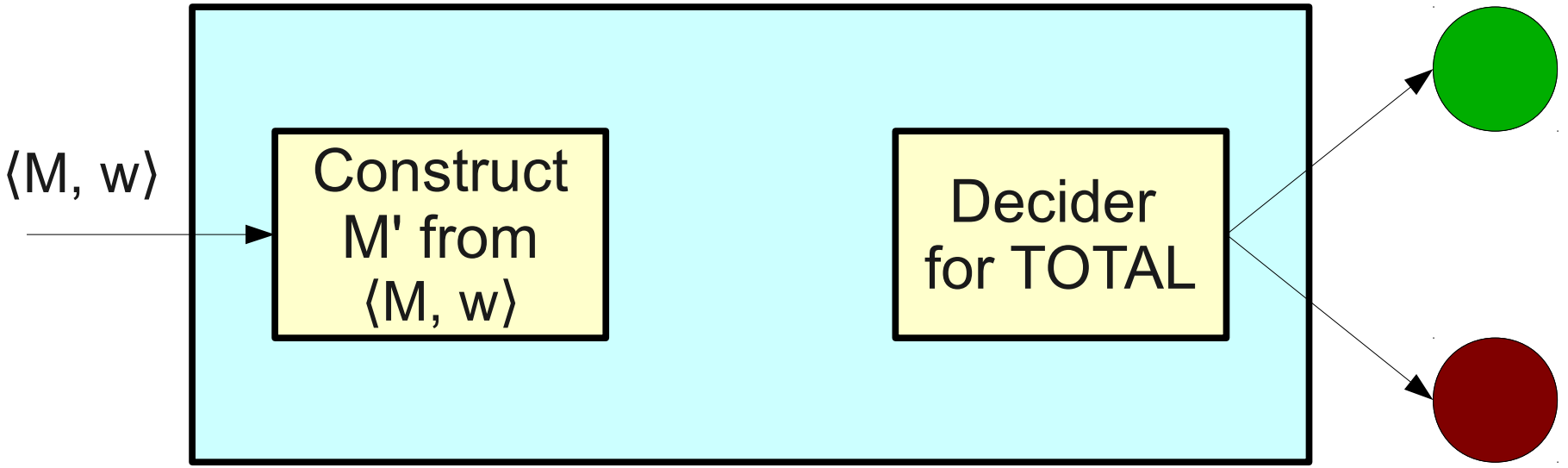
The Totality Problem

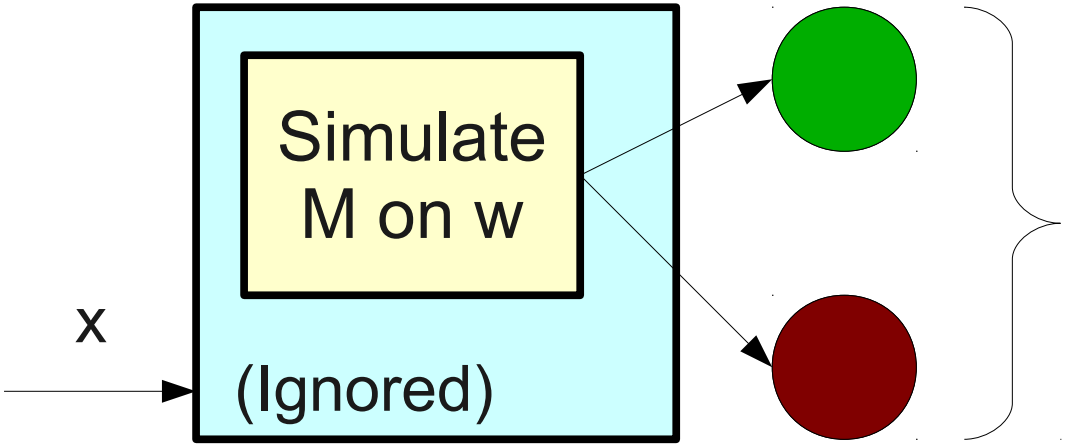




What does M' do if M halts on w?

The Totality Problem

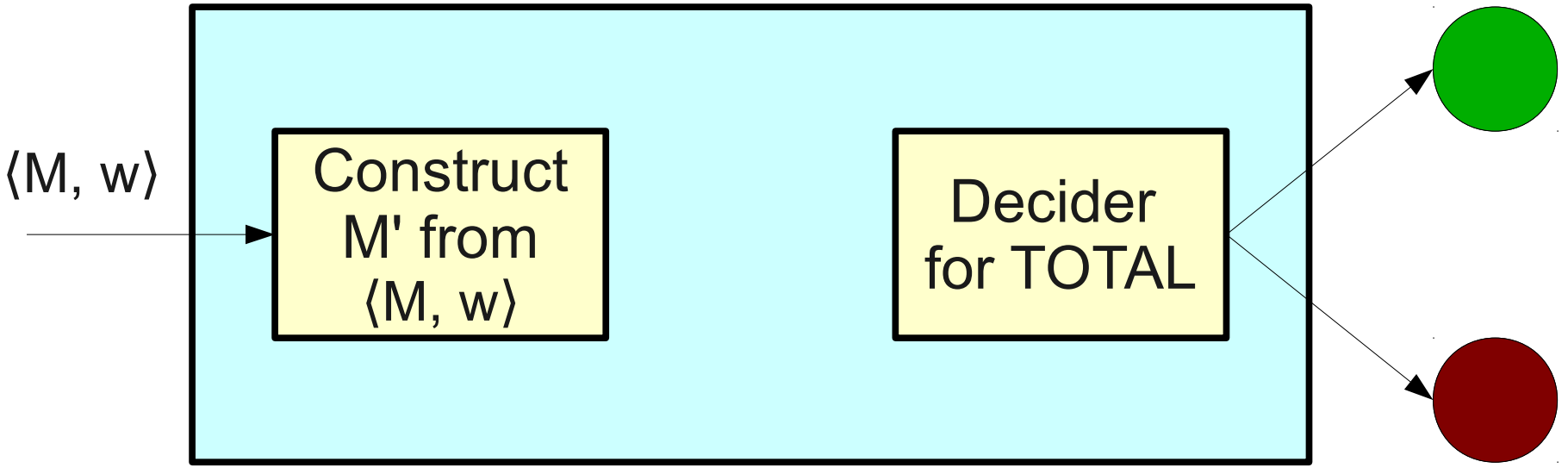


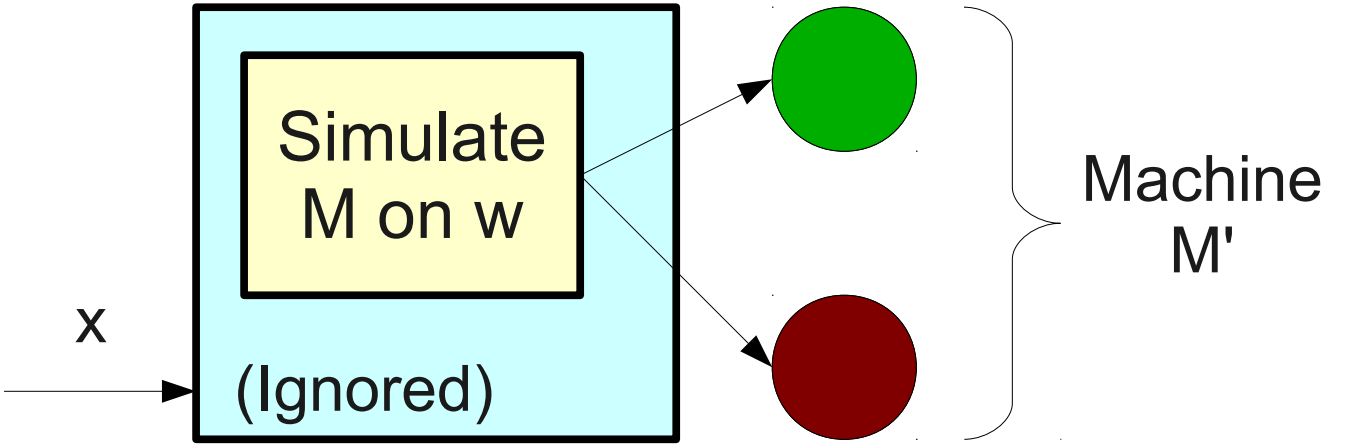


What does M' do if M halts on w?

M' always halts

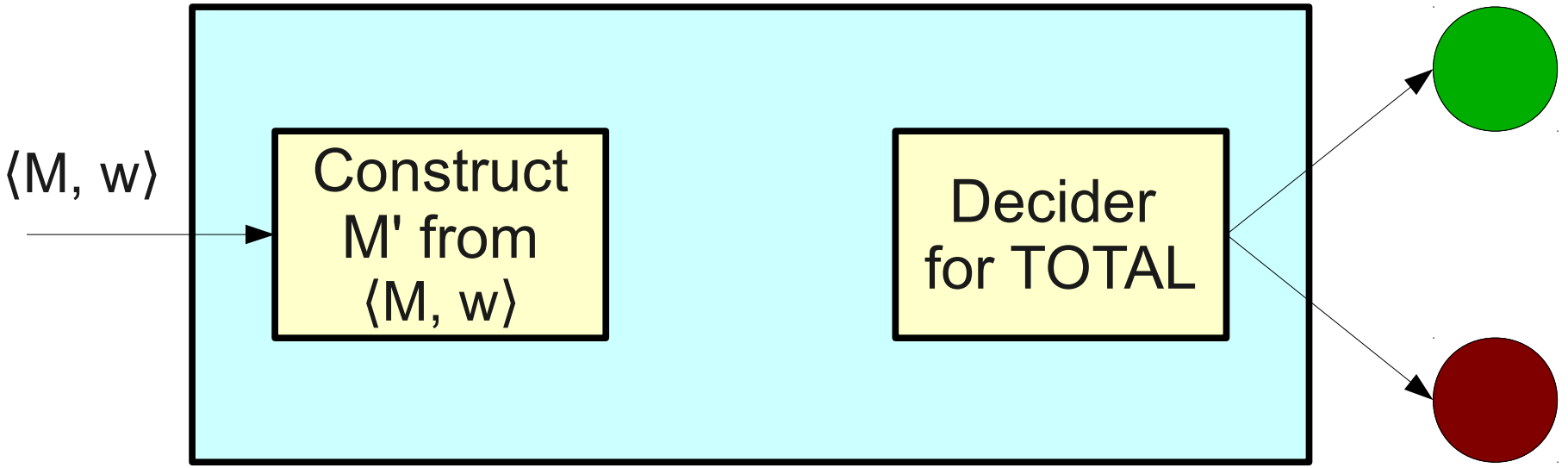
The Totality Problem

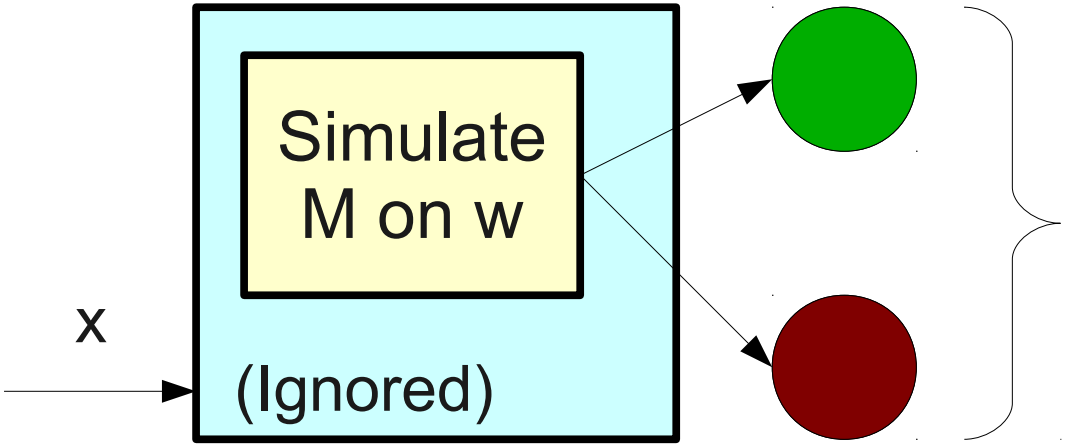




What does M' do if M loops on w?

The Totality Problem

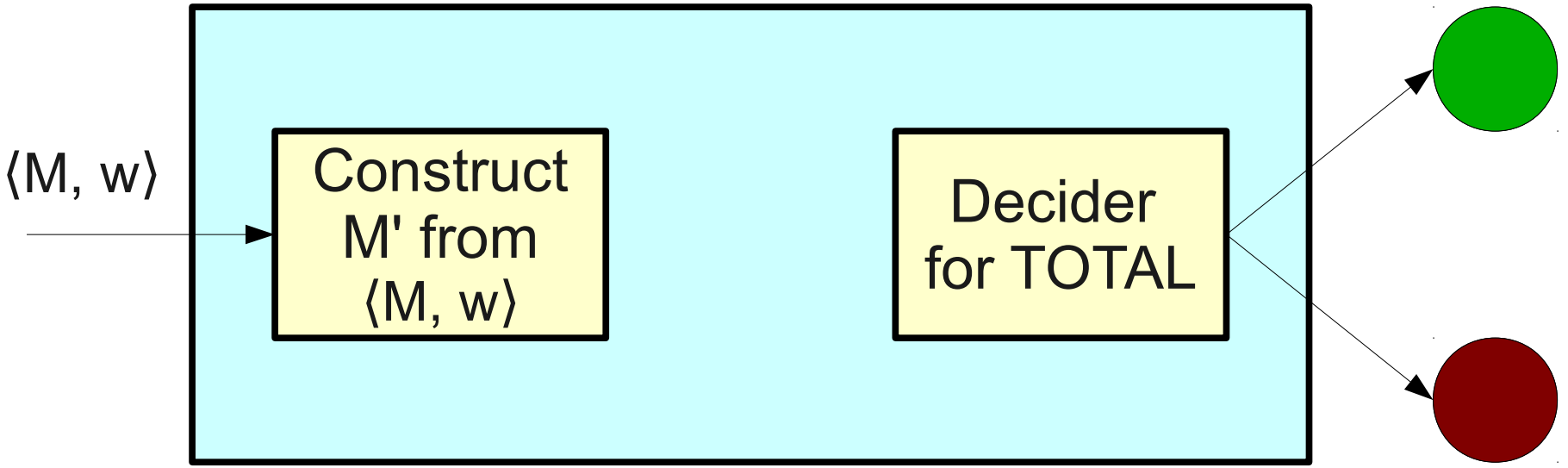


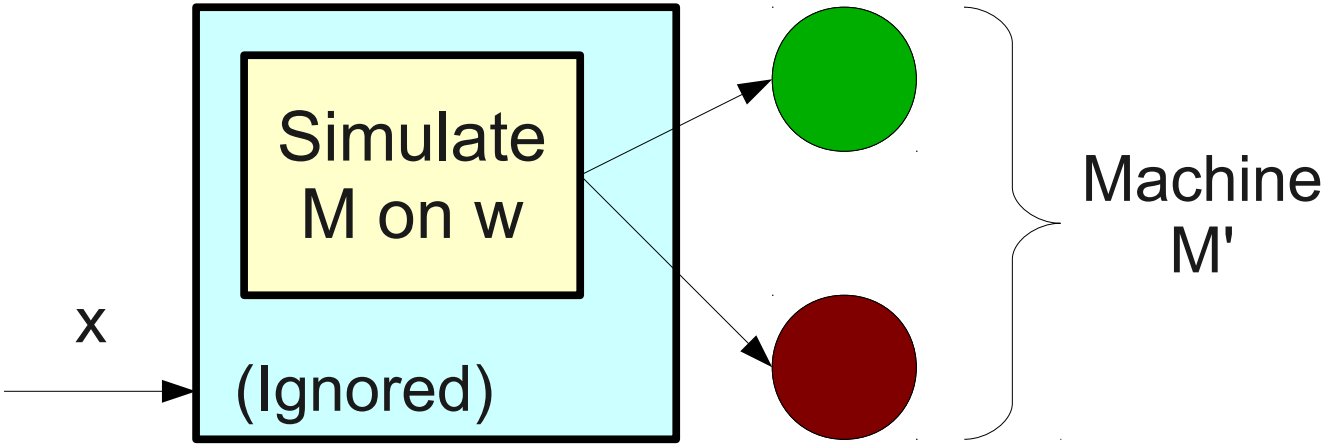


What does M' do if M loops on w?

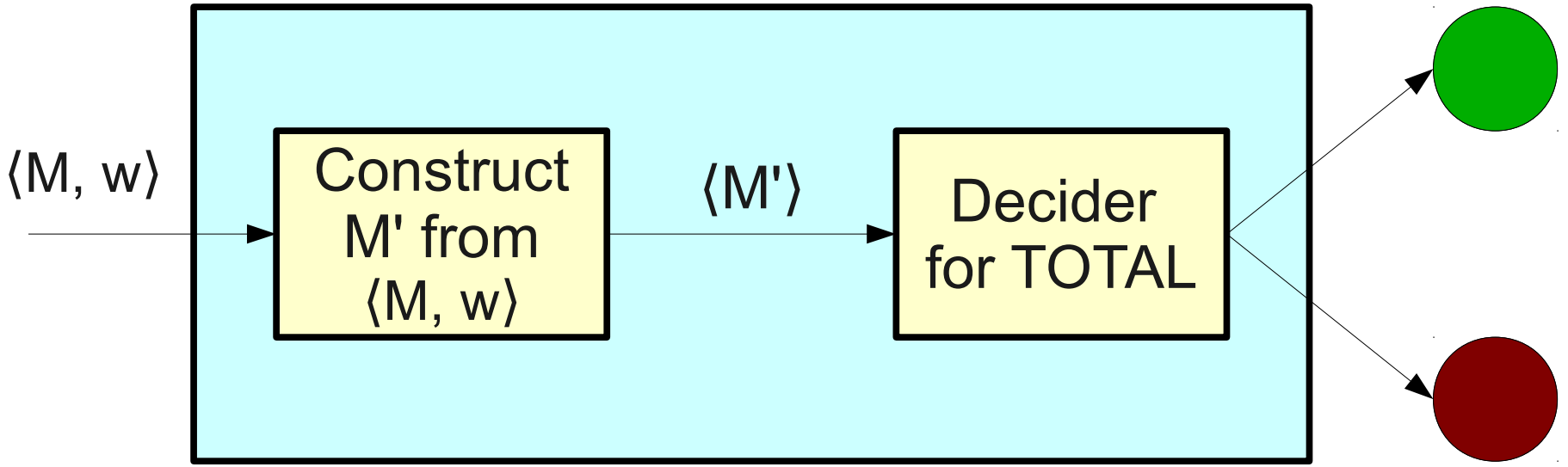
M' never halts

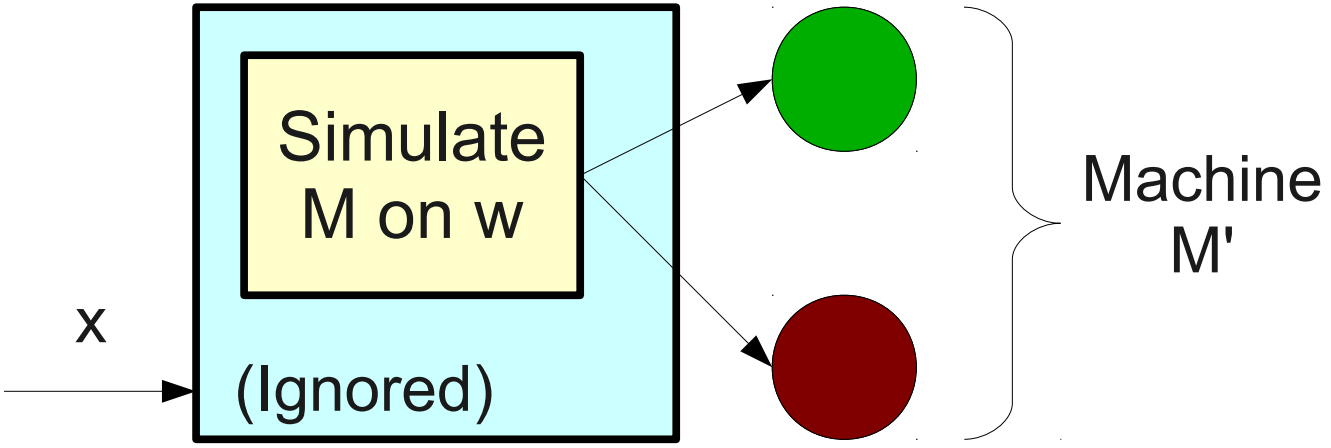
The Totality Problem



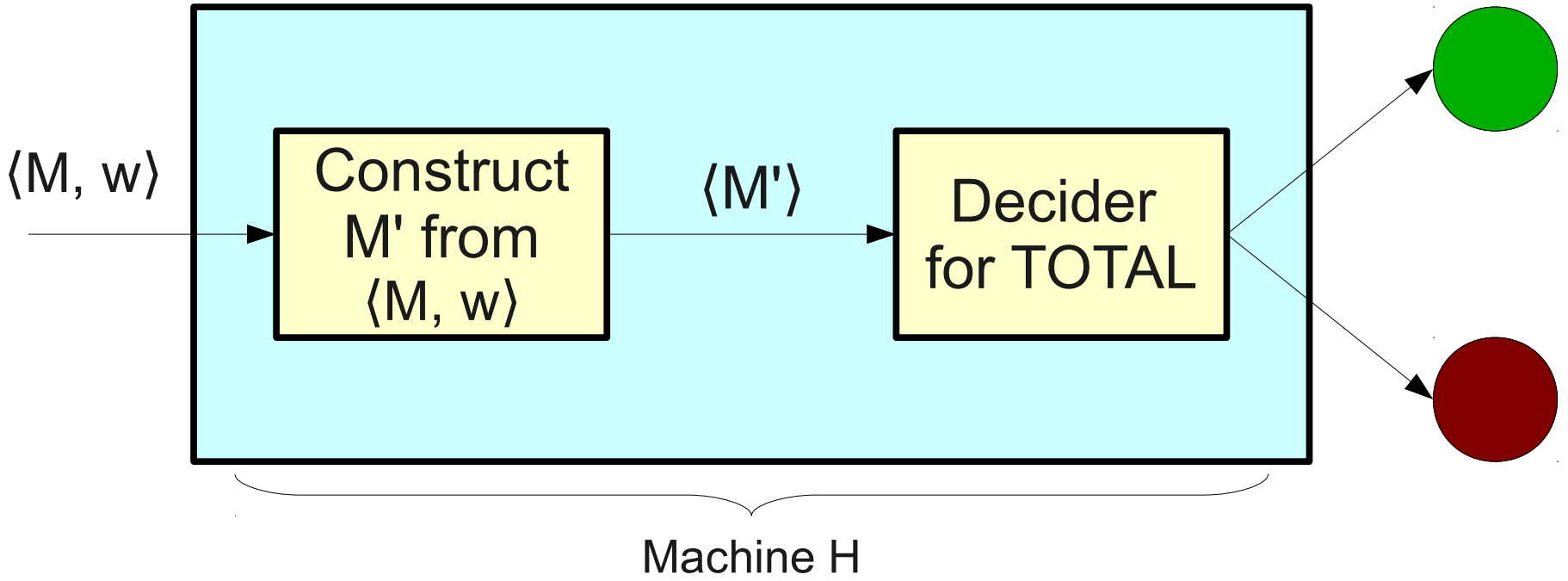


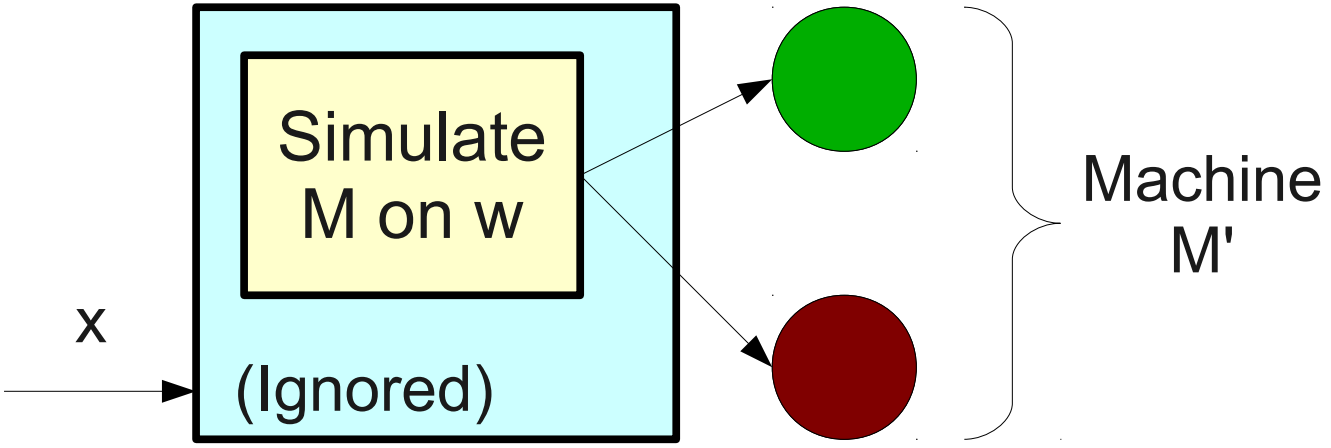
The Totality Problem



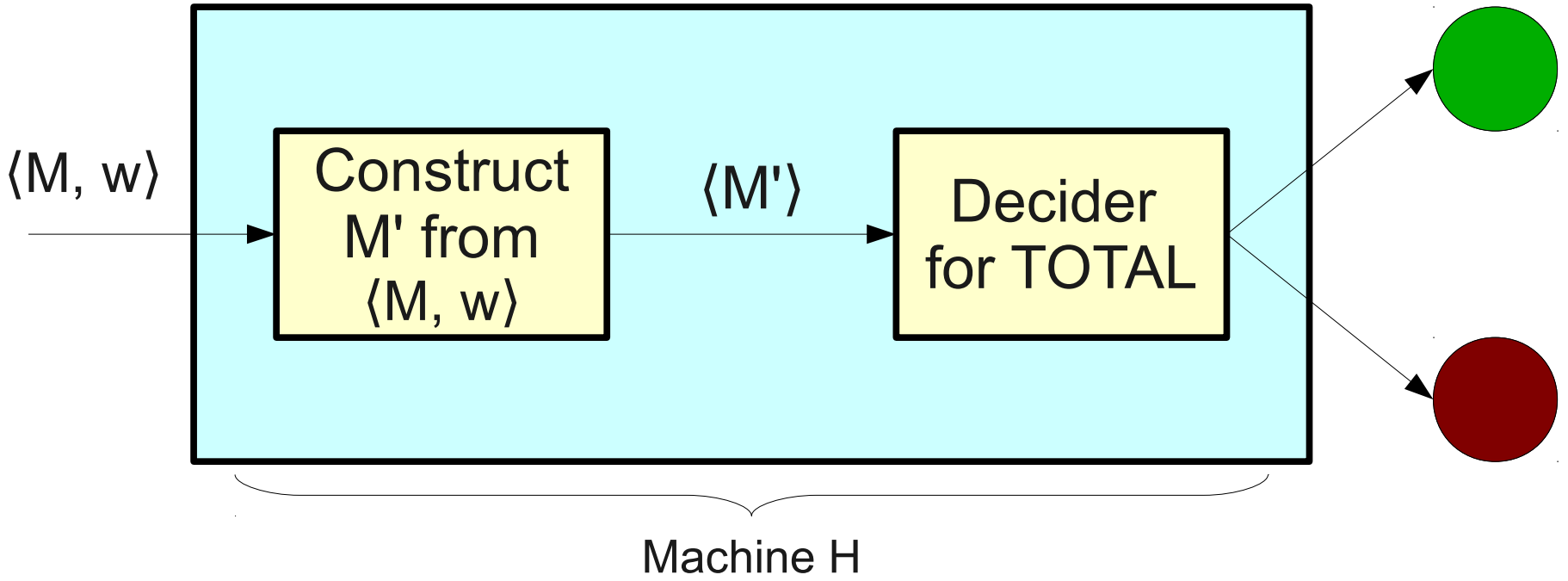


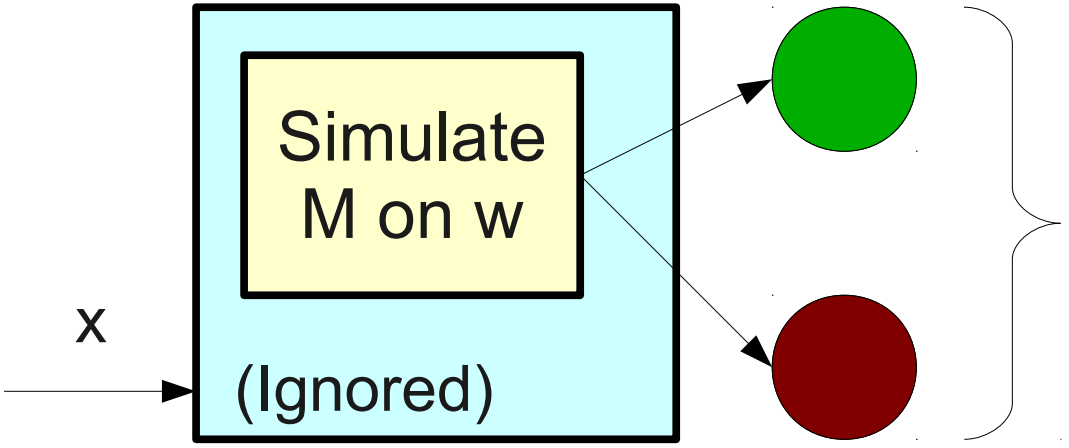
The Totality Problem





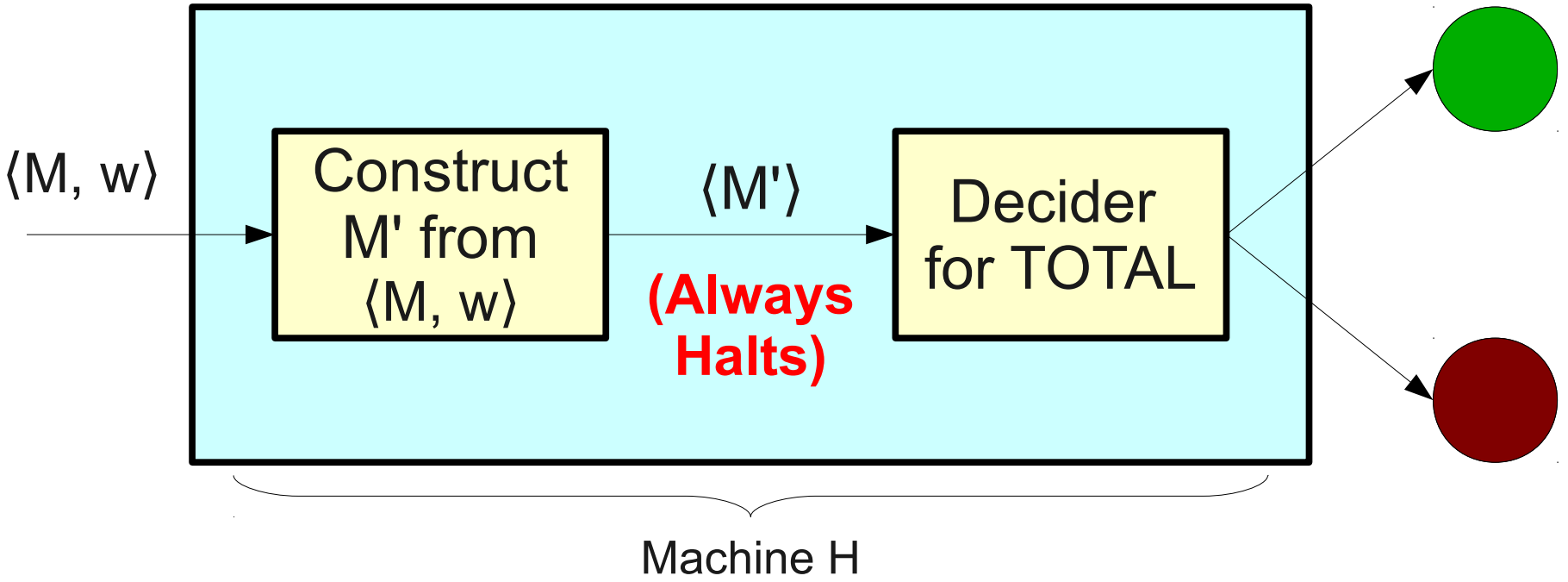
The Totality Problem

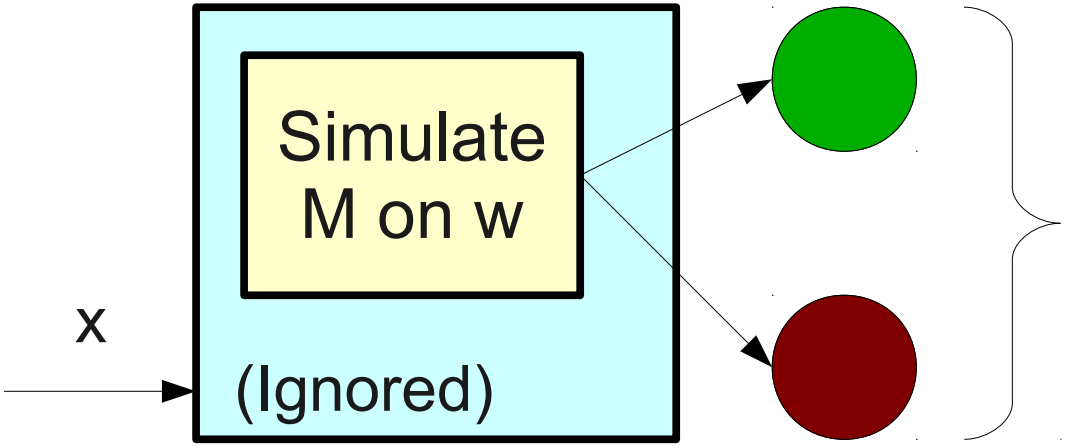




What does H do if M halts on w?

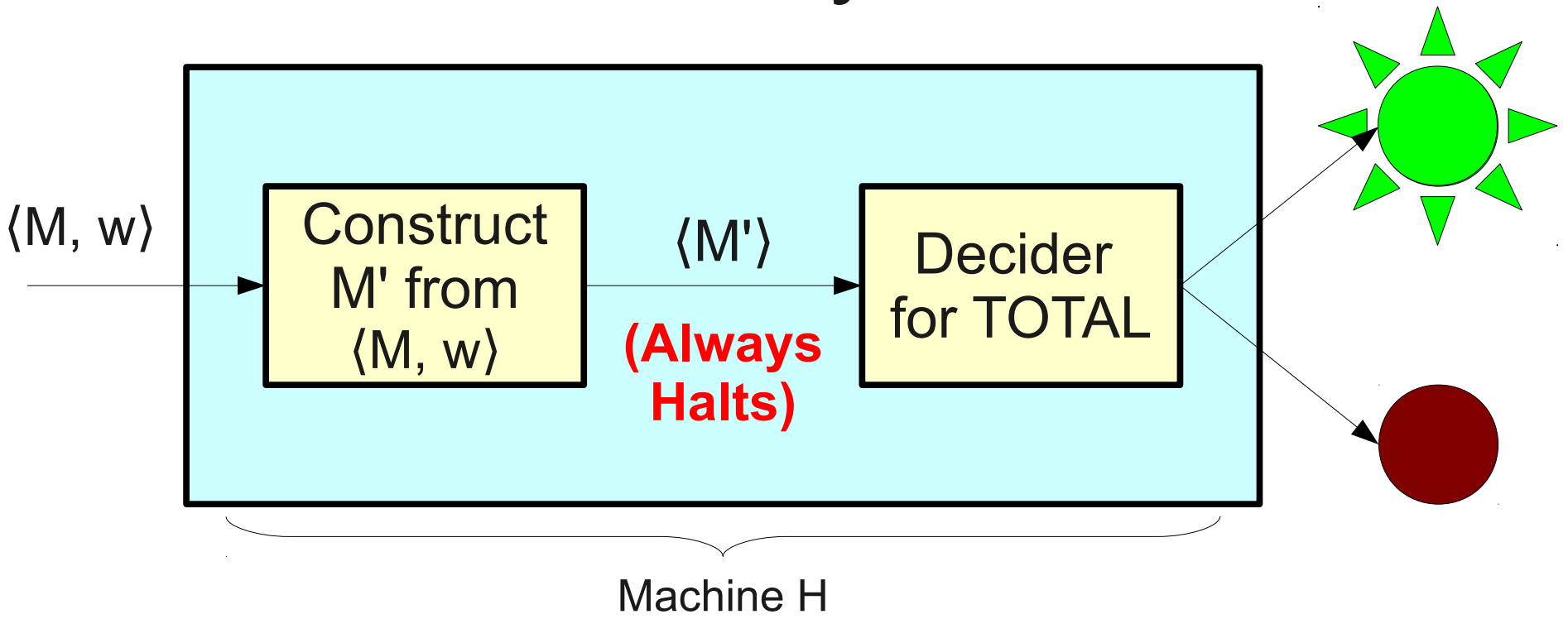
The Totality Problem

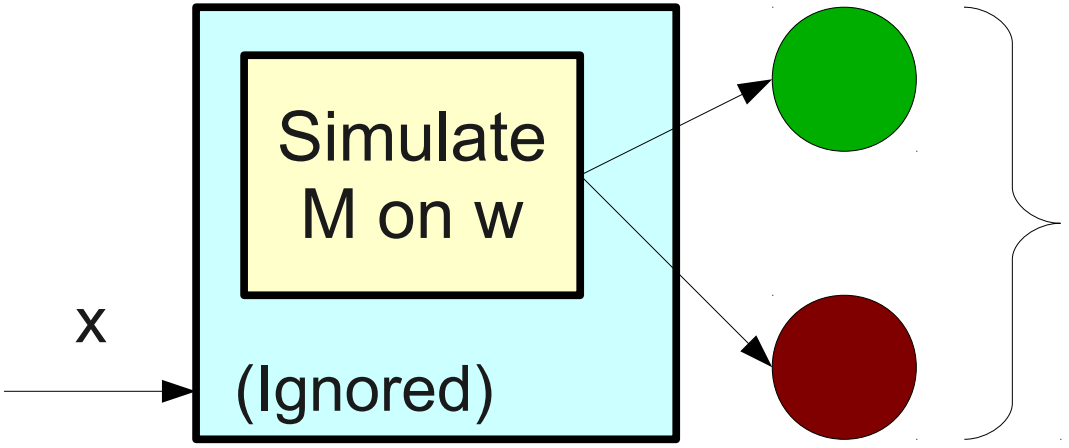




What does H do if M halts on w?

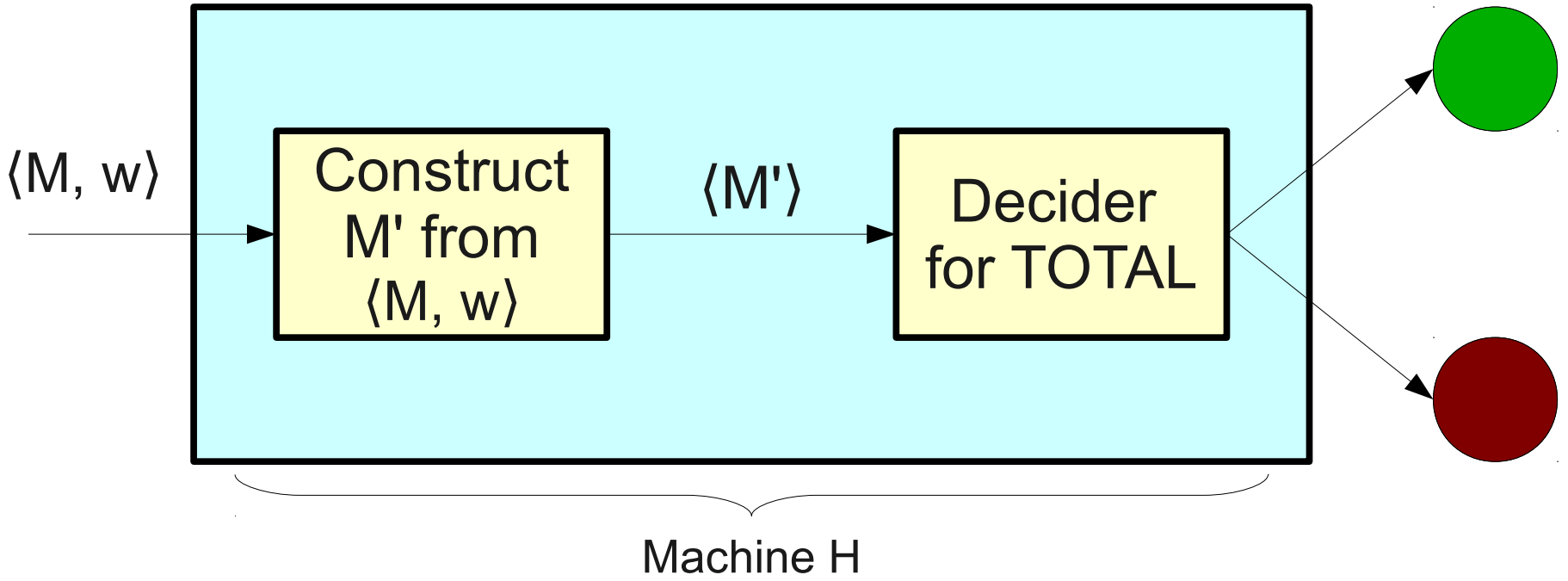
The Totality Problem

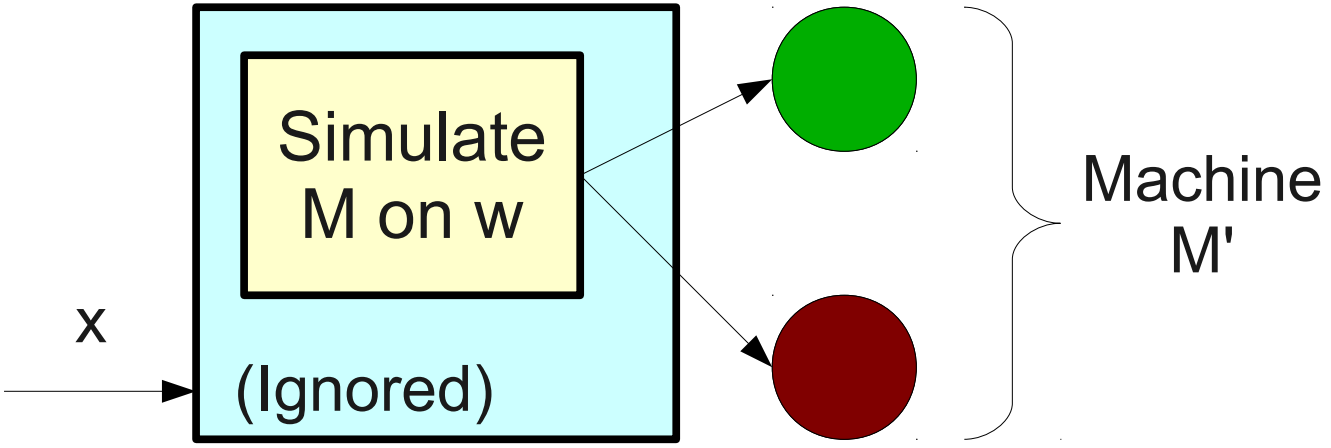




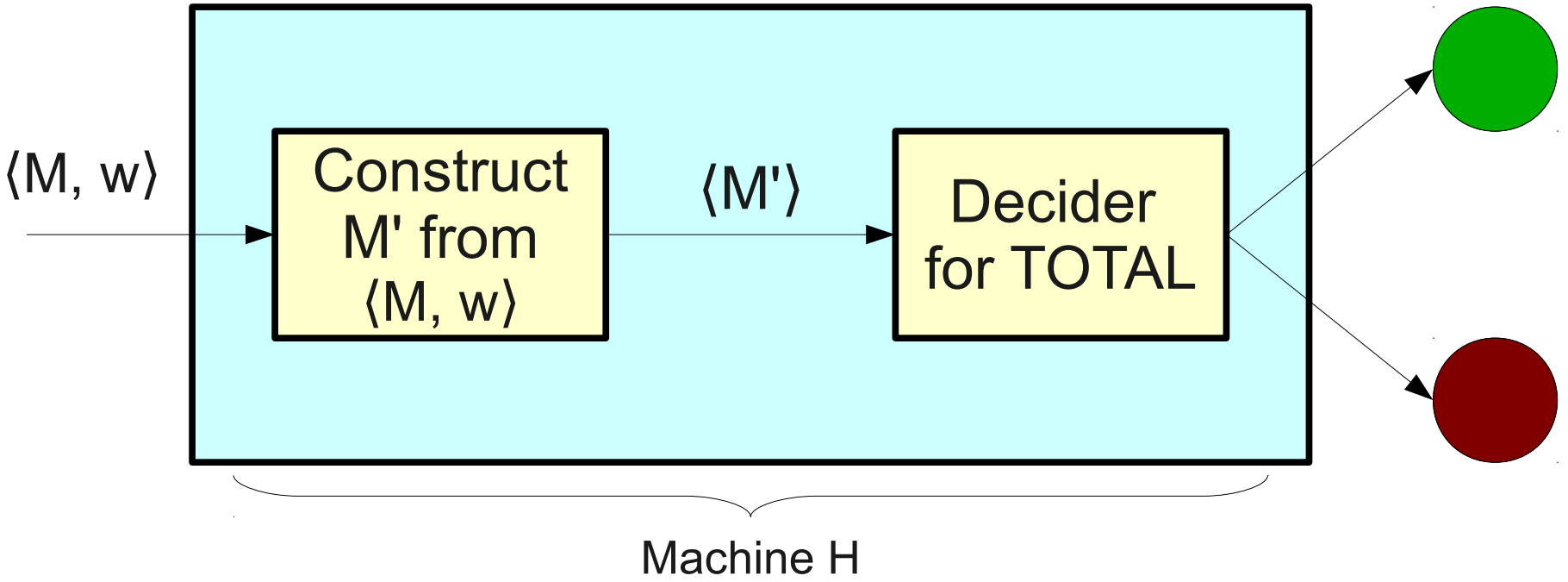
What does H do if M halts on w?

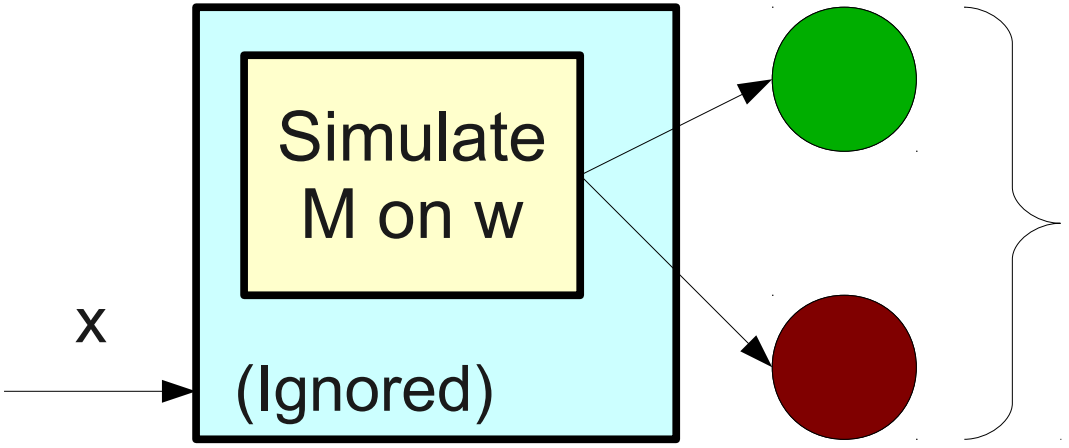
The Totality Problem





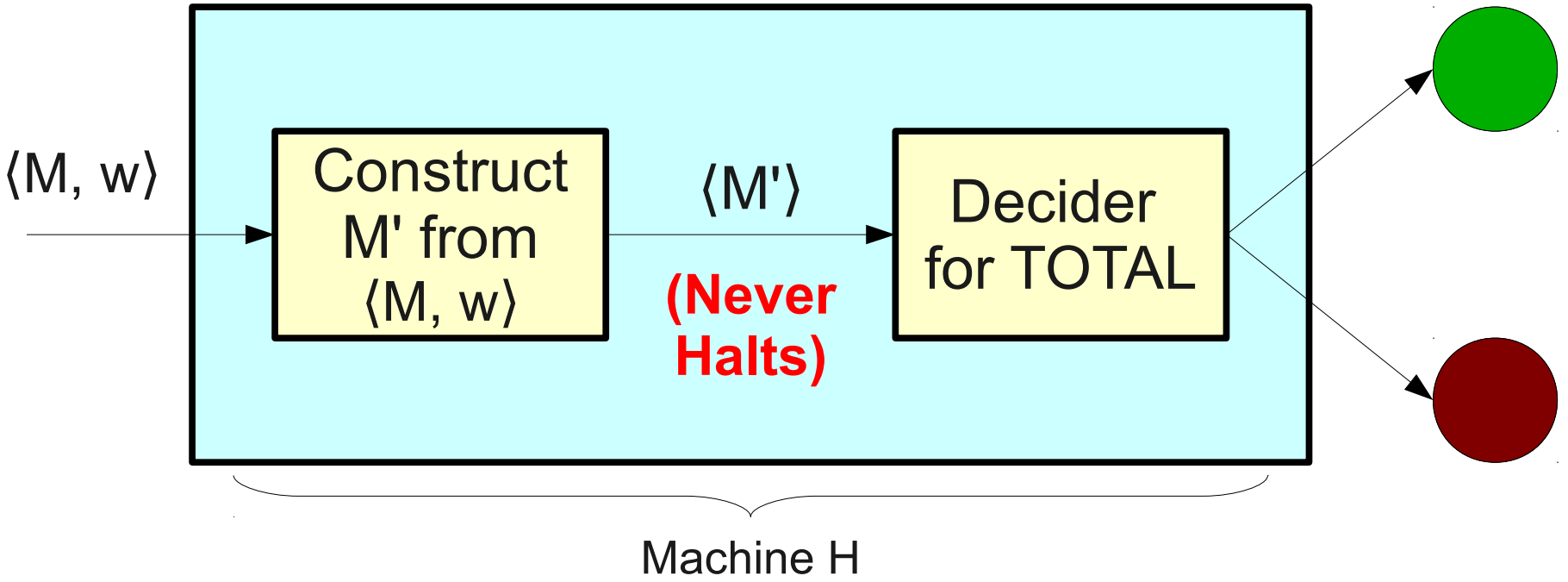
The Totality Problem

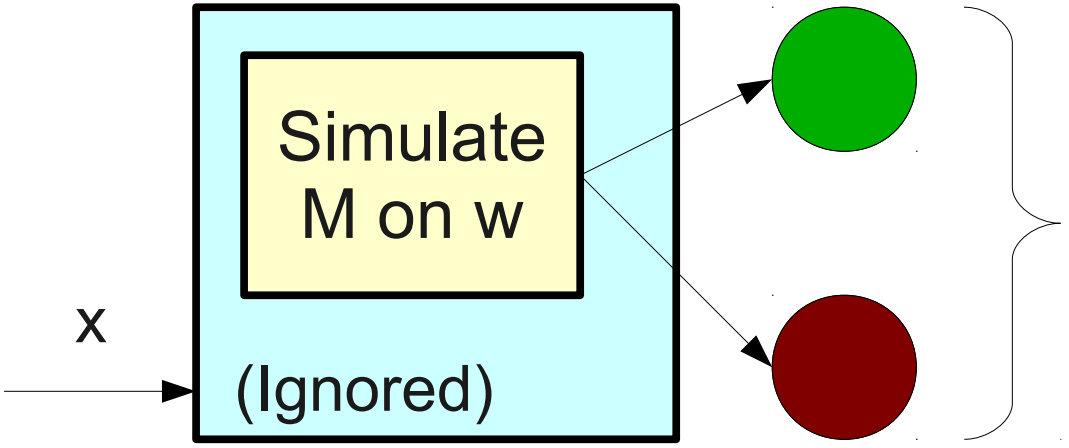




What does H do if M loops on w?

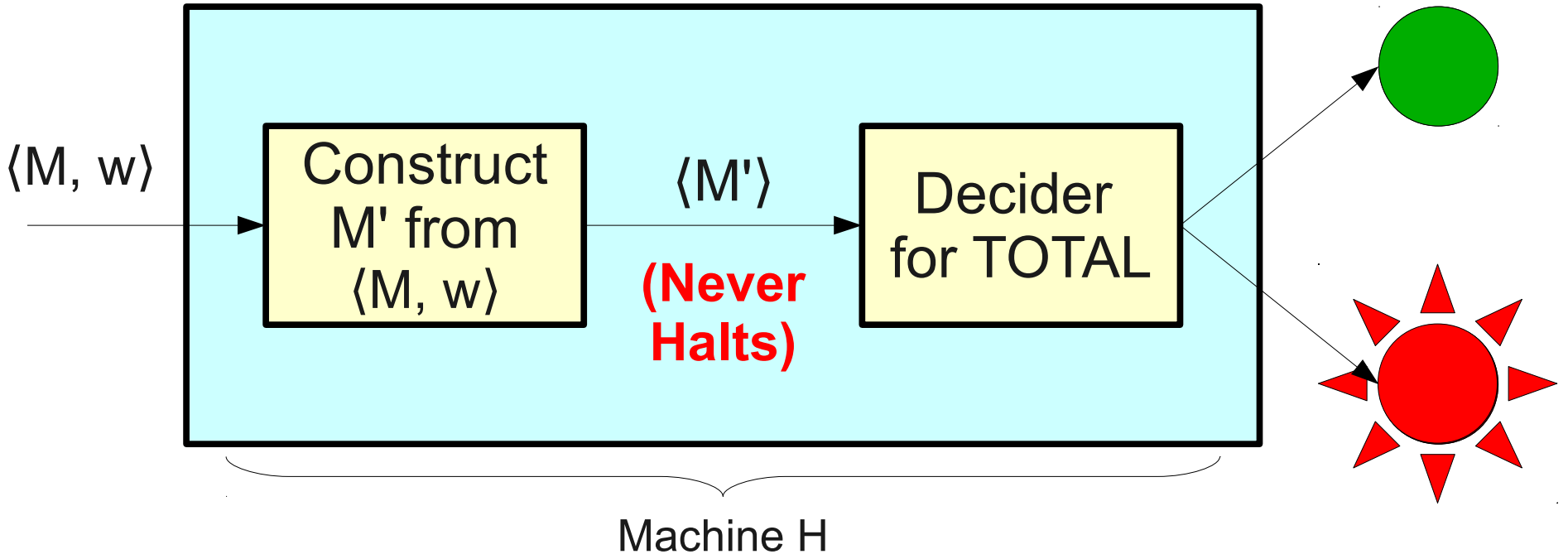
The Totality Problem

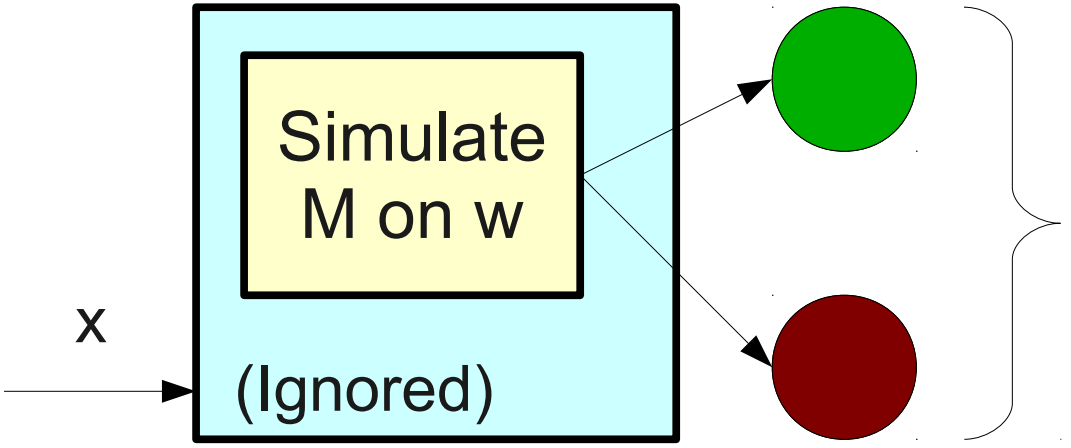




What does H do if M loops on w?

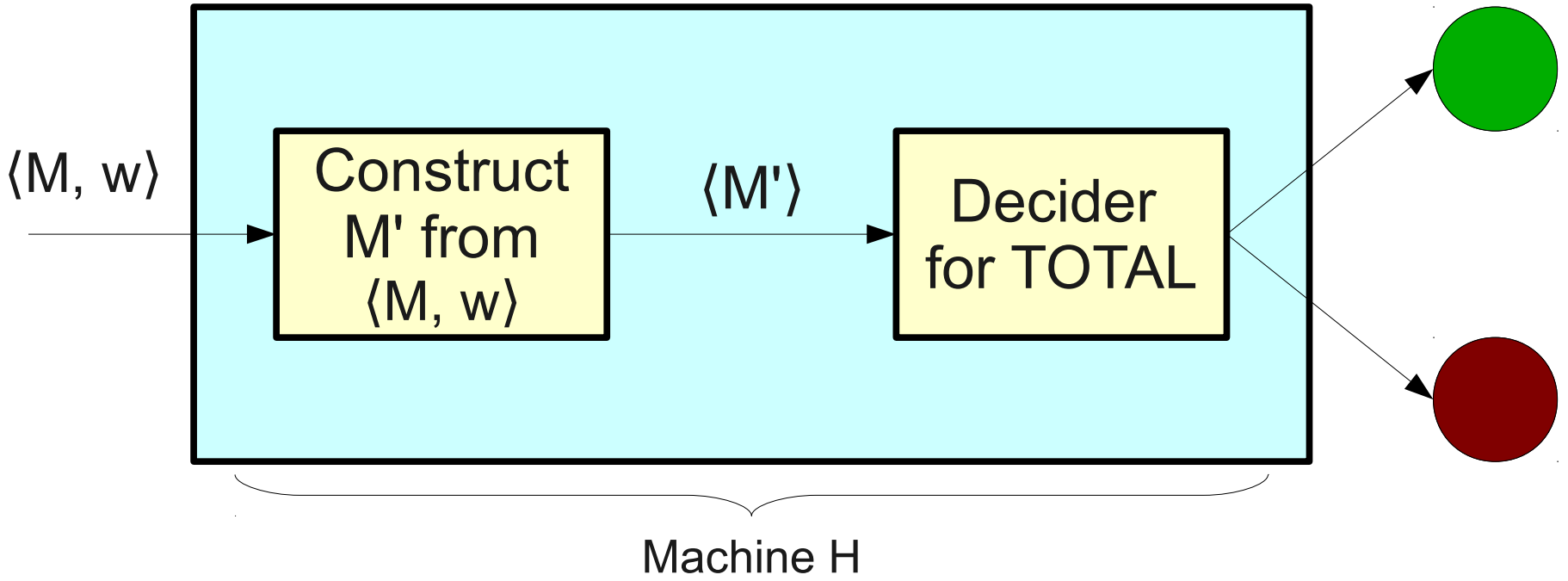
The Totality Problem

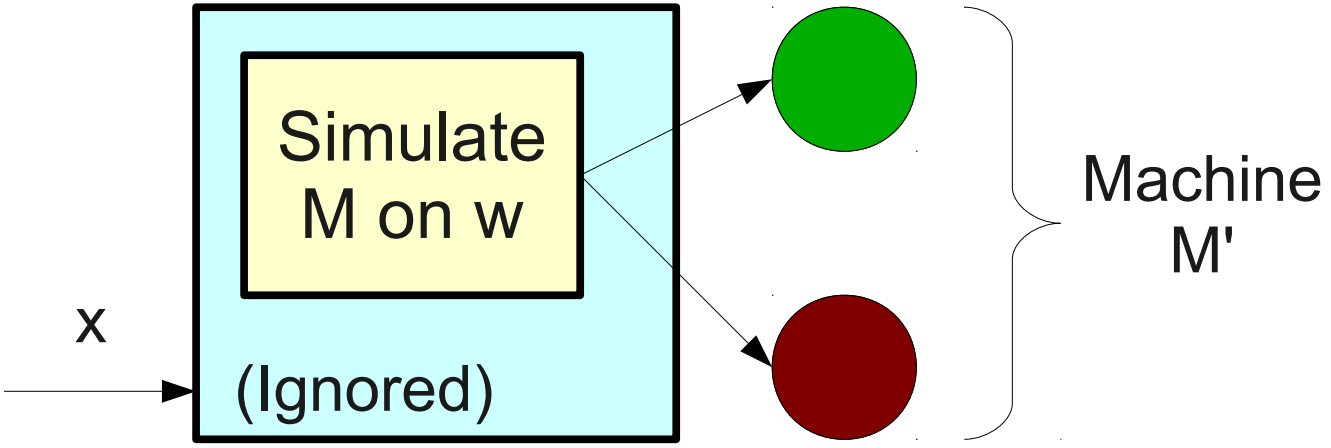




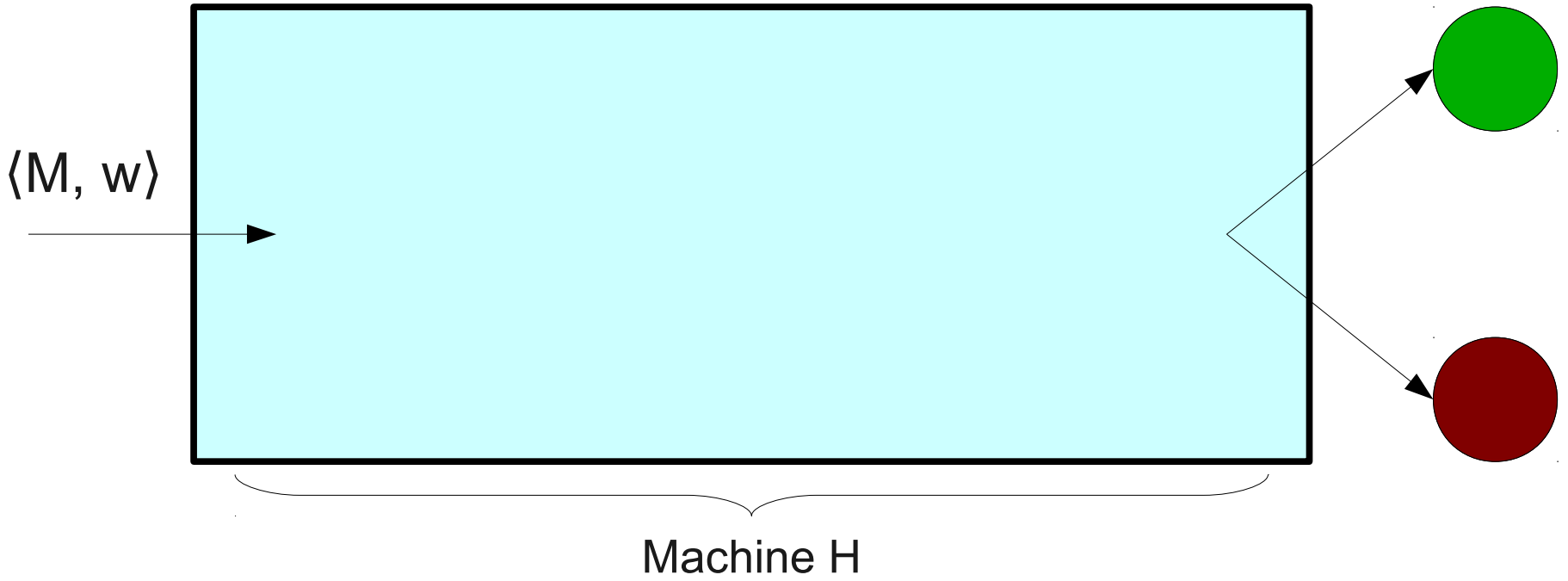
What does H do if M loops on w?

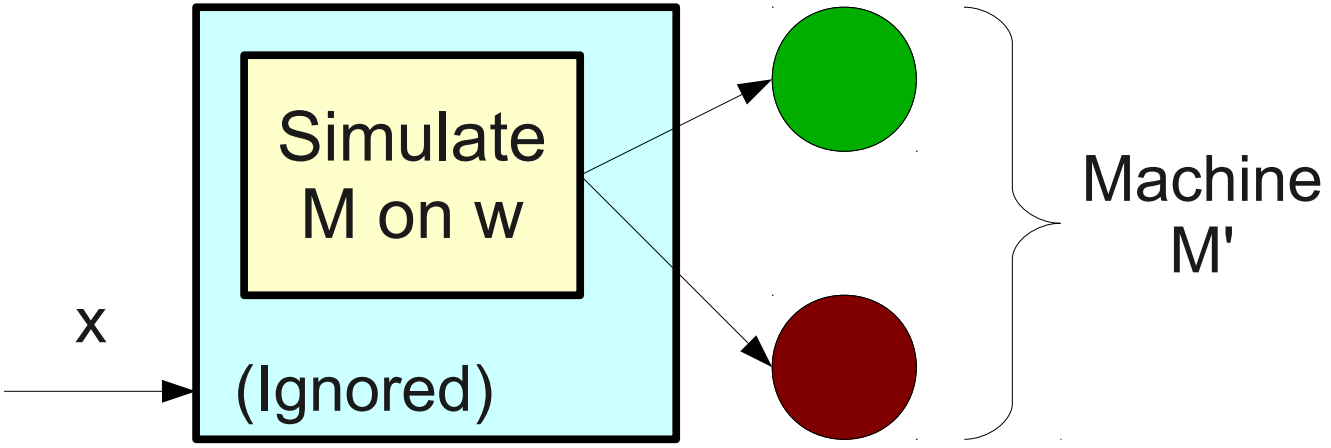
The Totality Problem



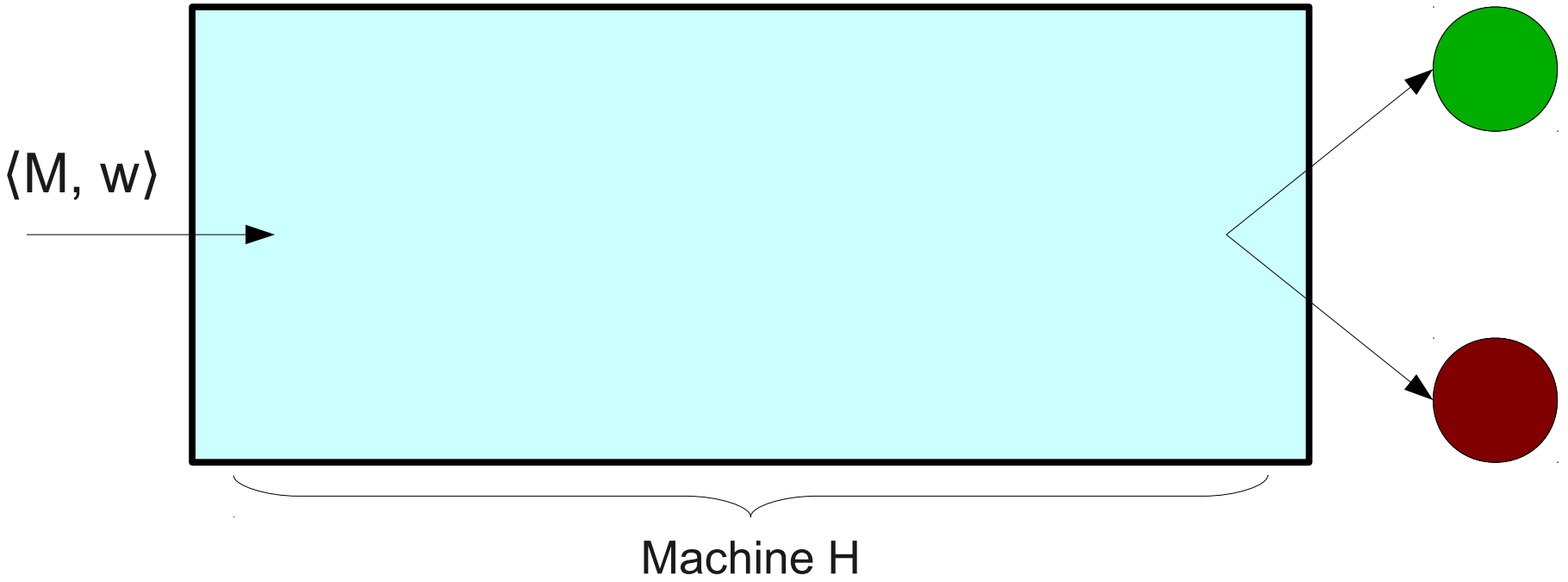


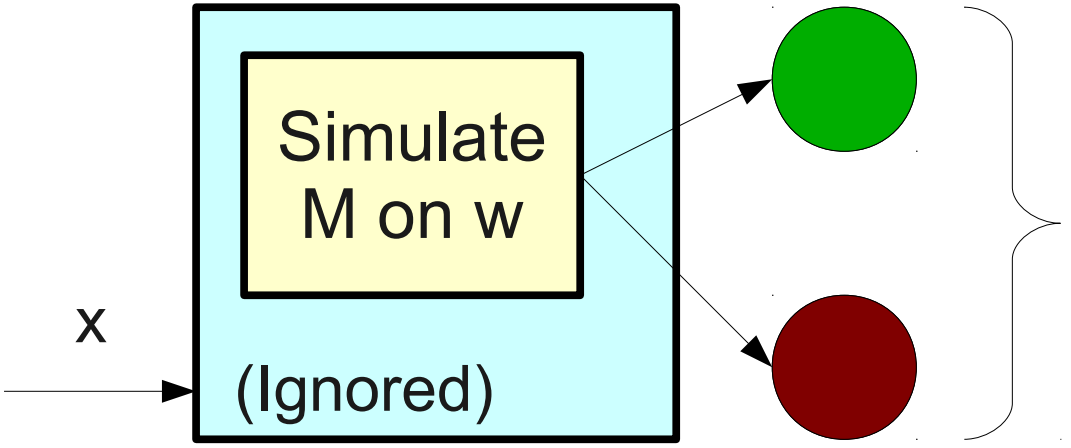
The Totality Problem





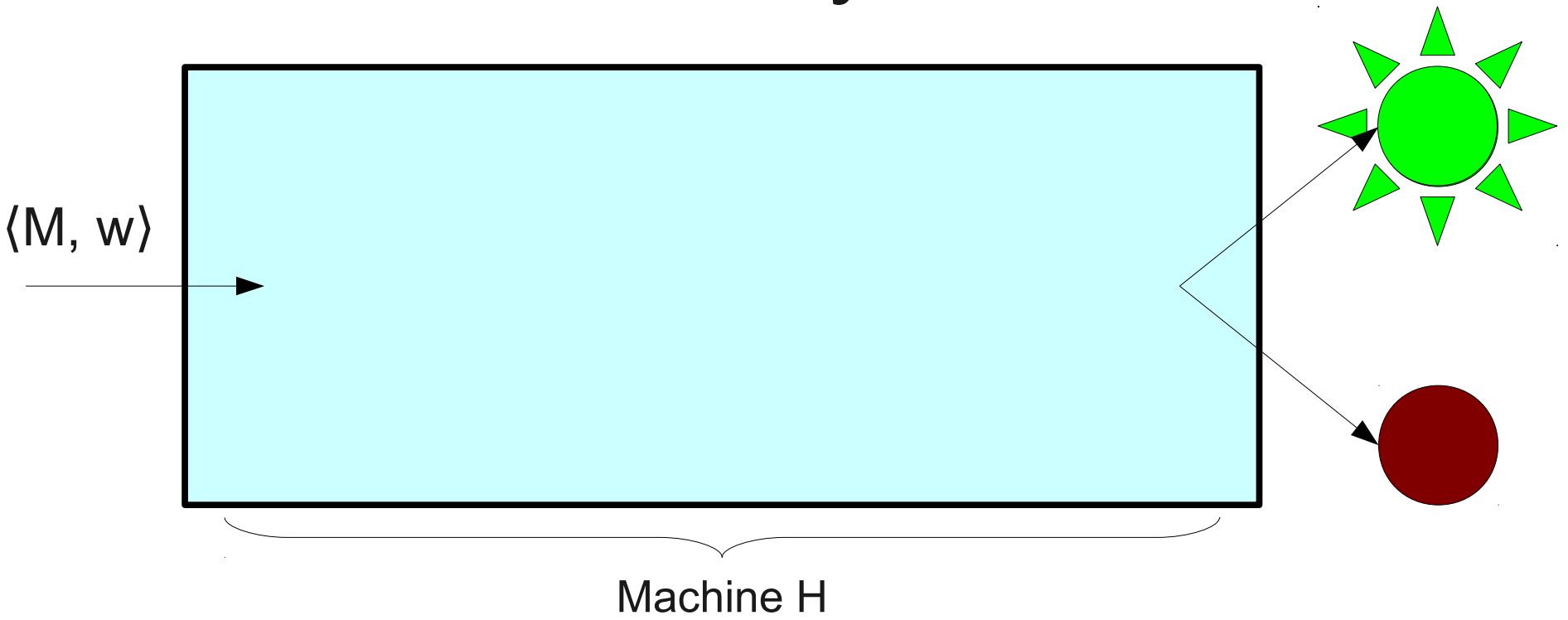
The Totality Problem

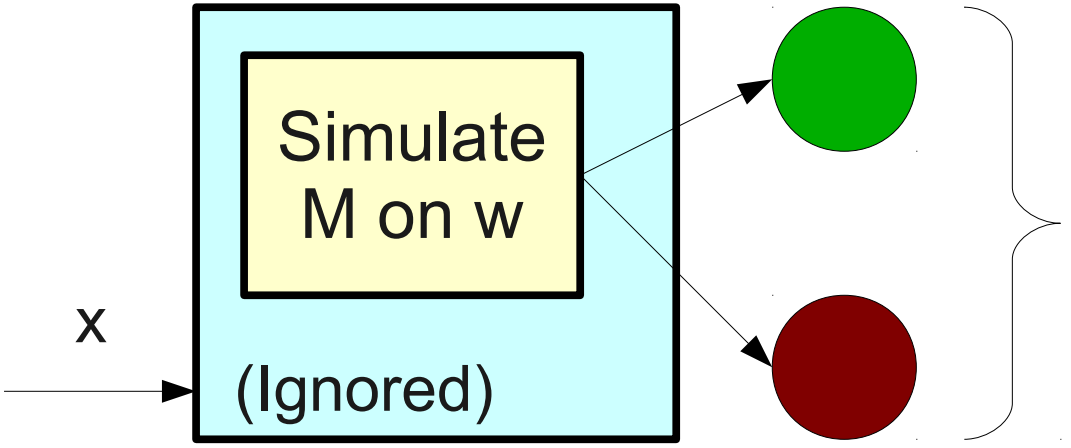




What does H do if M halts on w?

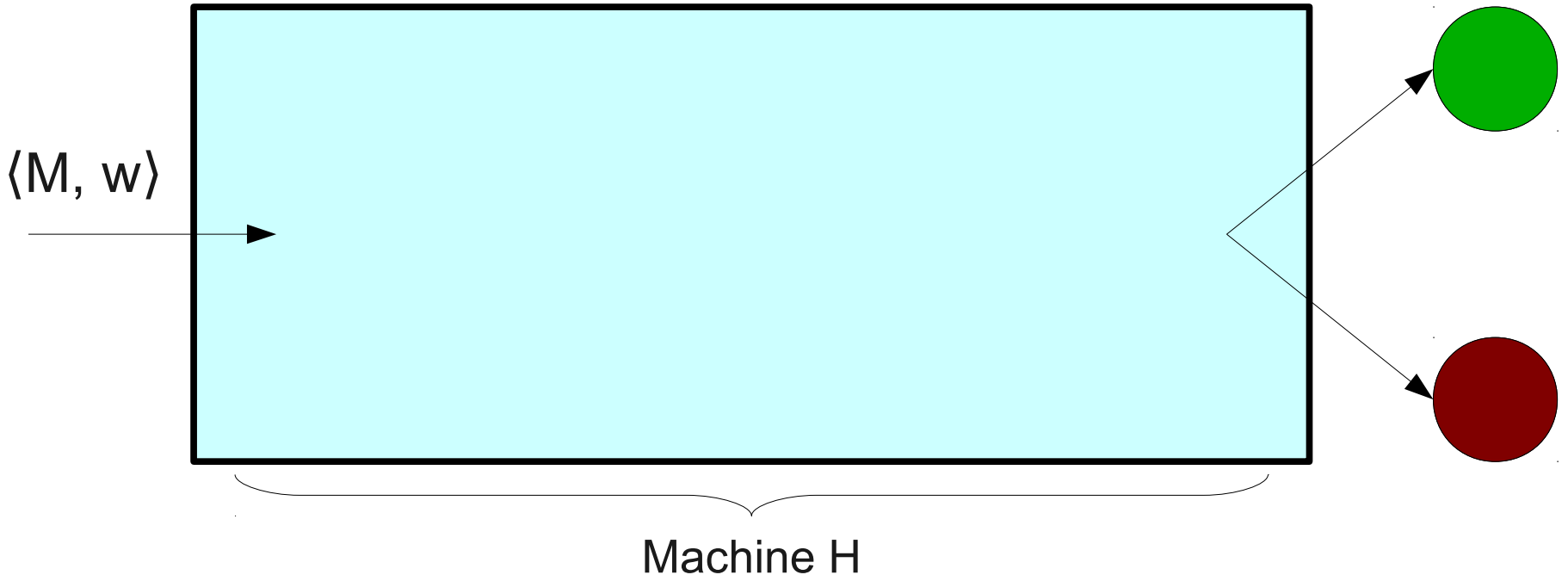
The Totality Problem

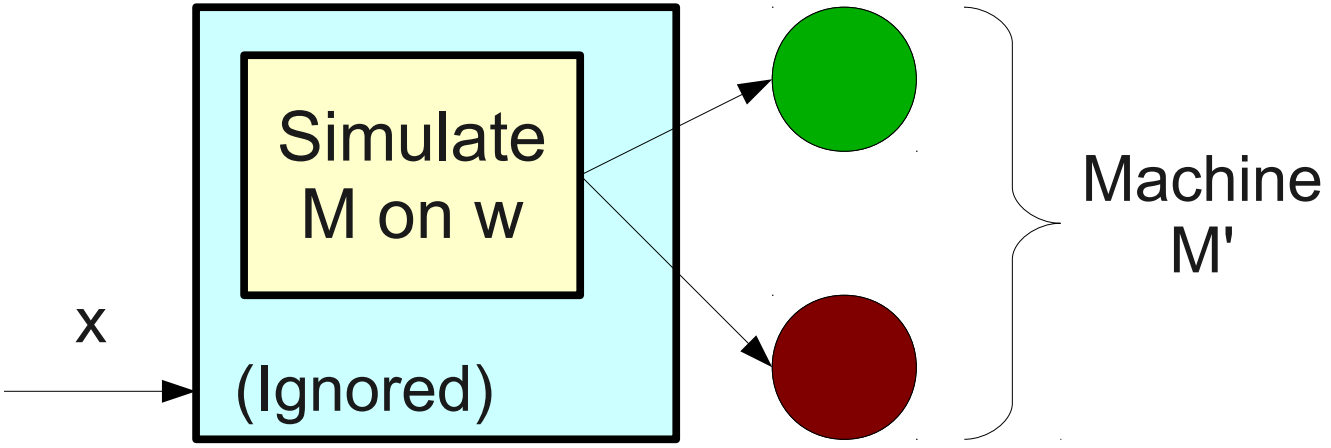




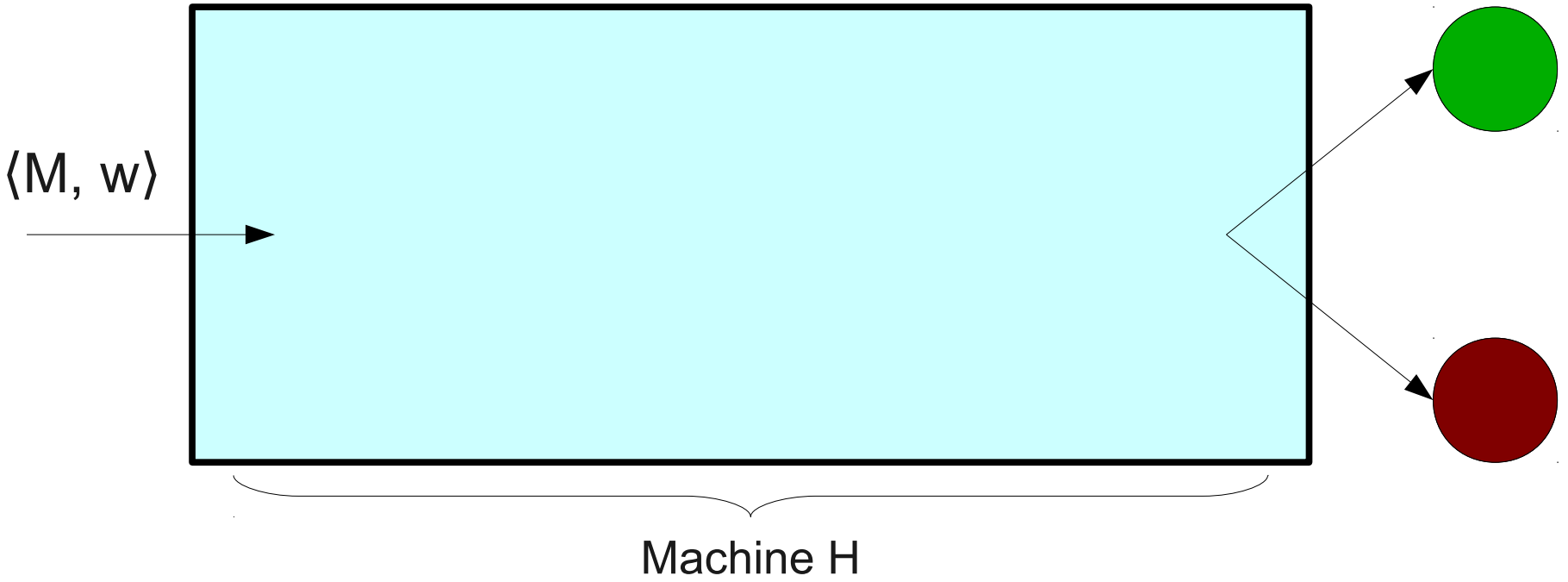
What does H do if M halts on w?

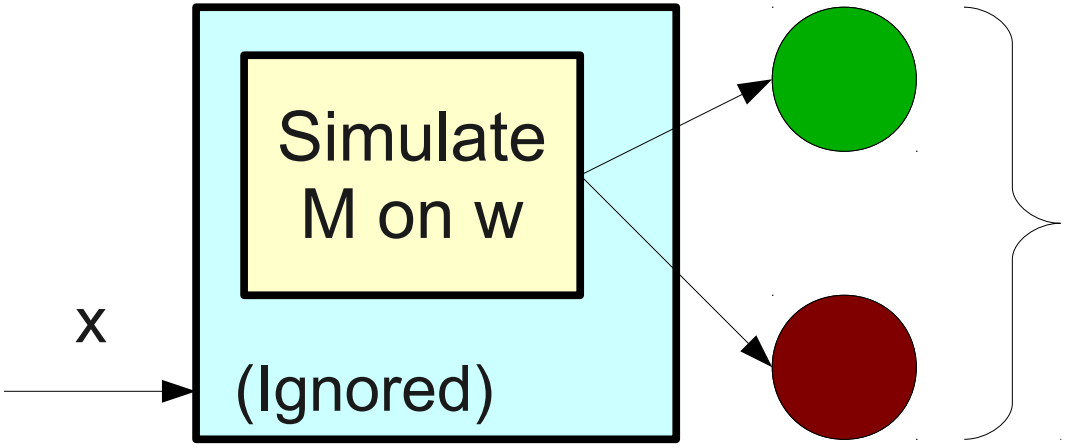
The Totality Problem





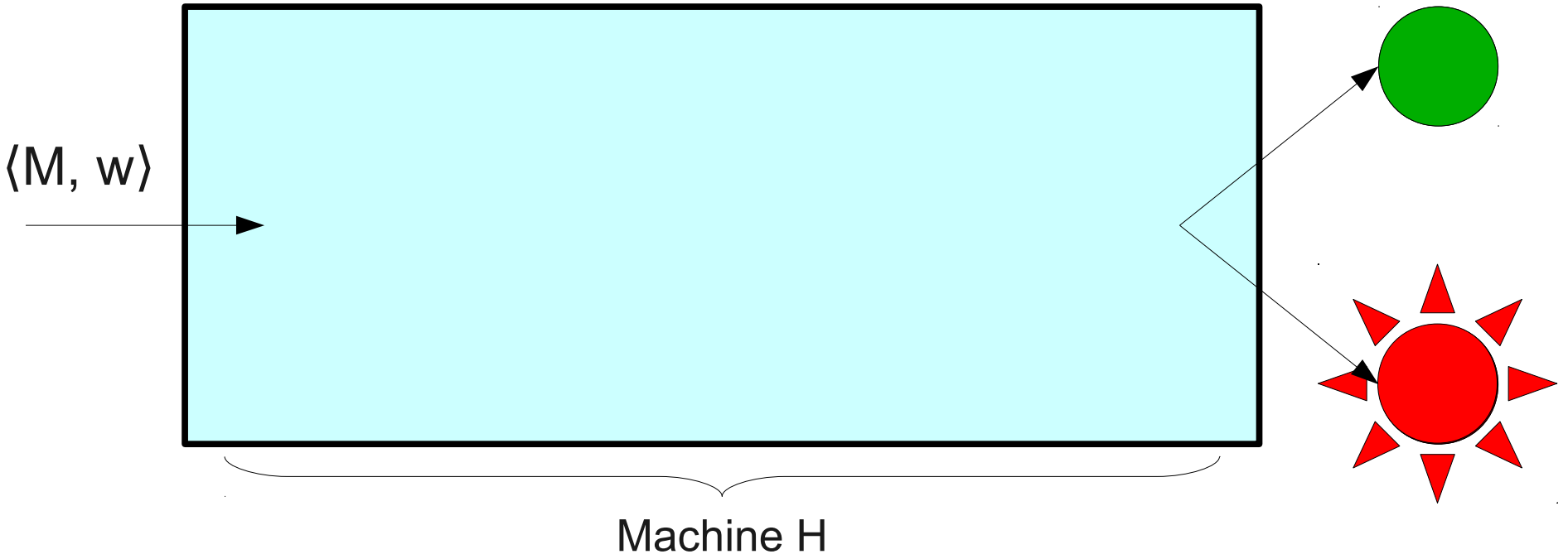
The Totality Problem

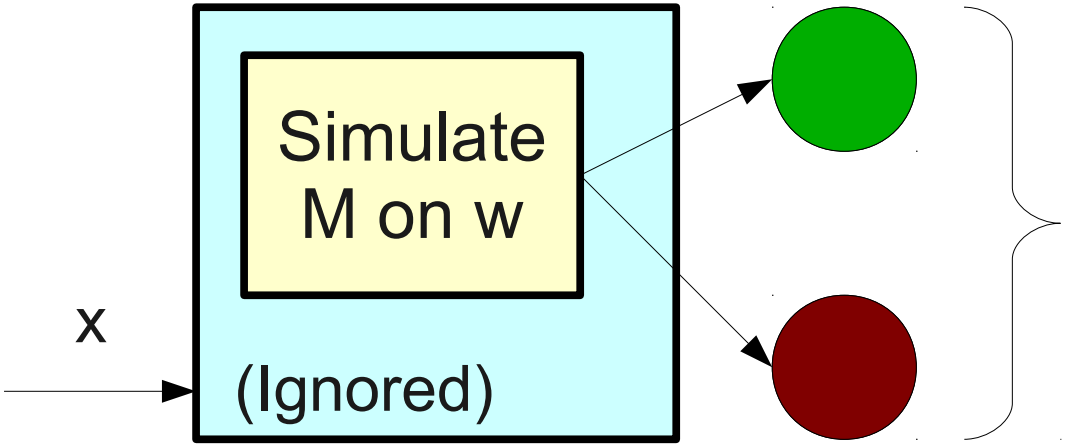




What does H do if M loops on w?

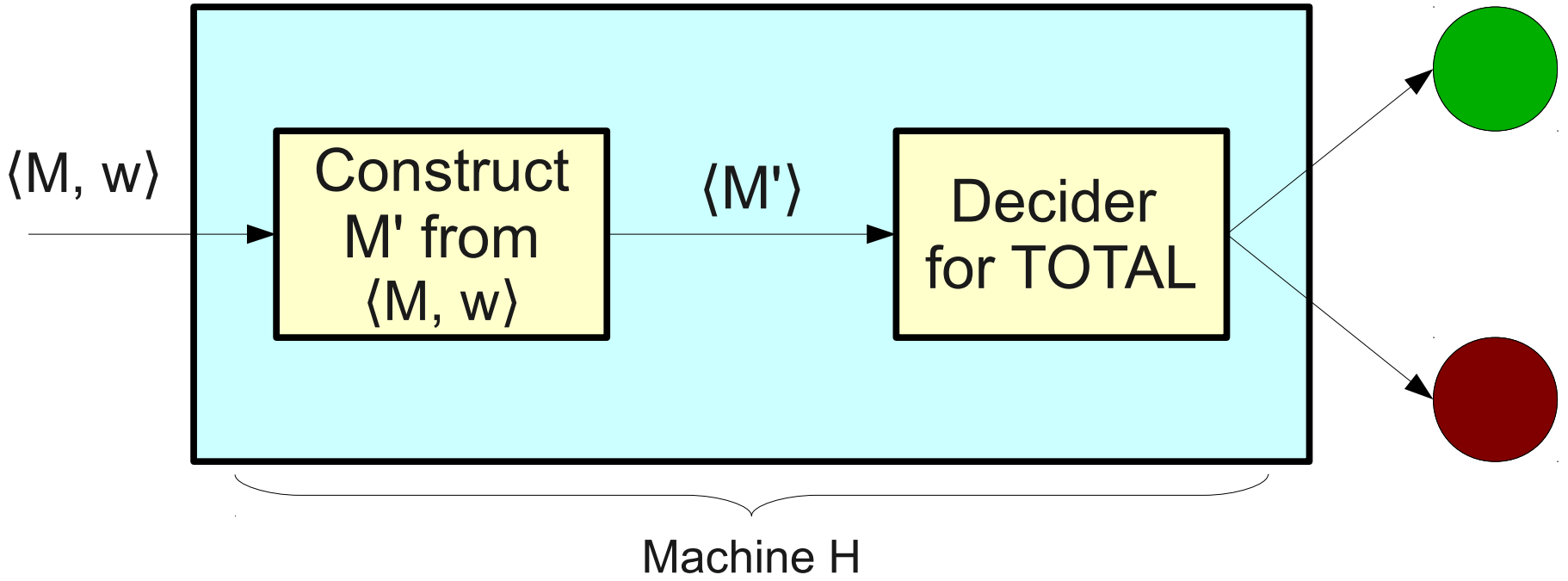
The Totality Problem

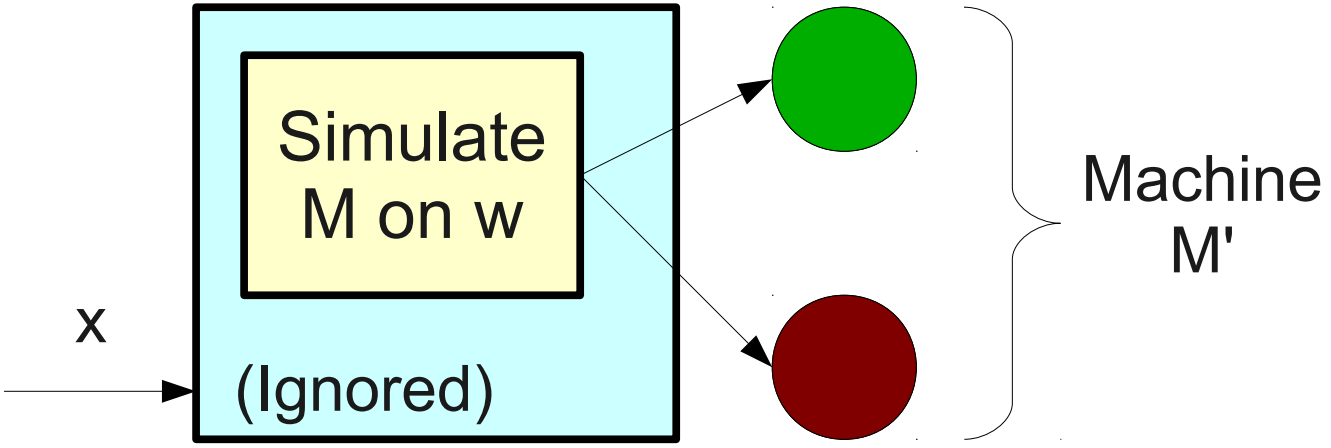




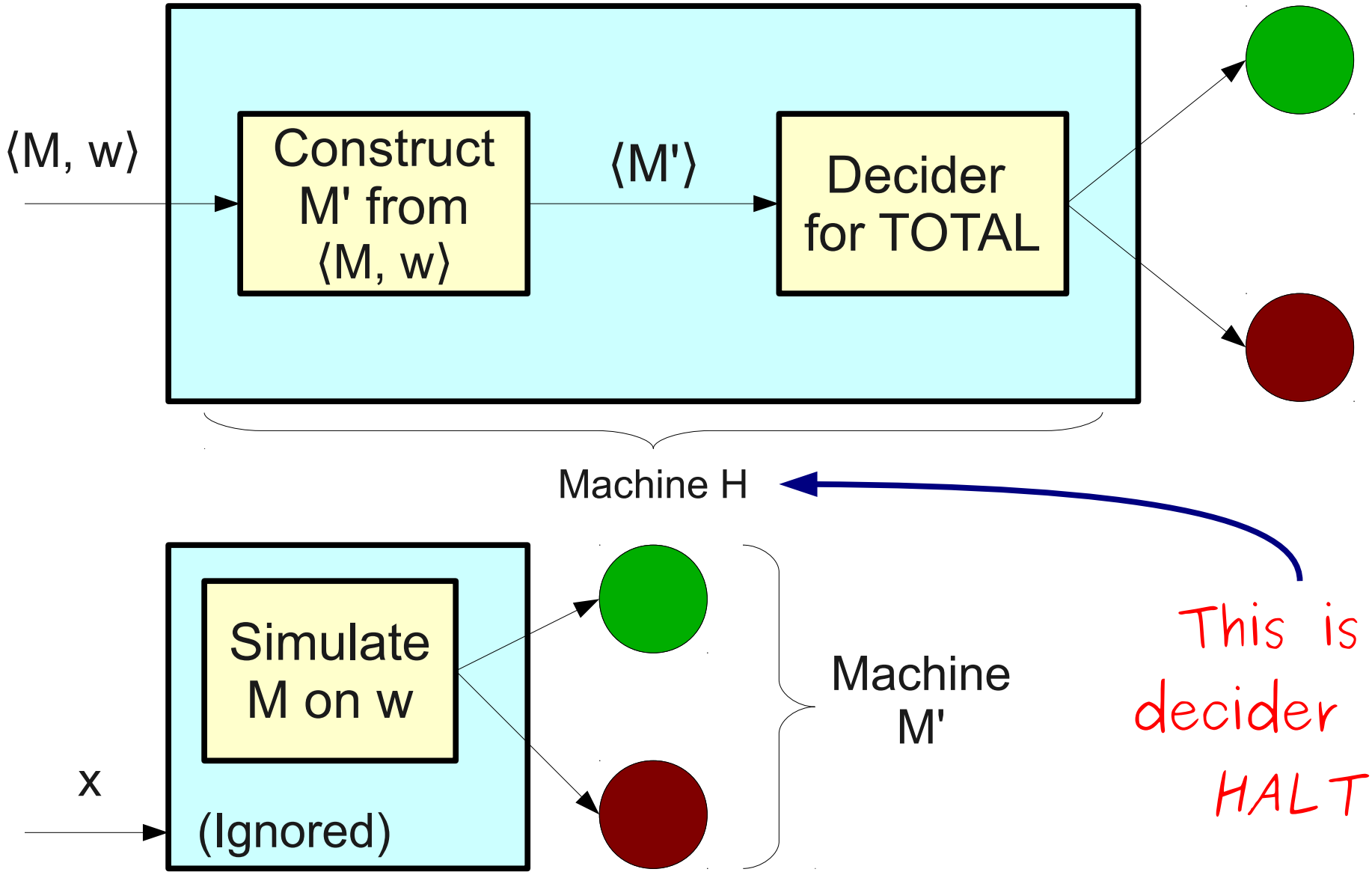
What does H do if M loops on w?

The Totality Problem





The Totality Problem



What Just Happened?

- Let's walk through this construction in detail.

What Just Happened?

- Let's walk through this construction in detail.
  - Suppose, for the sake of contradiction, that *TOTAL* is decidable.

What Just Happened?

- Let's walk through this construction in detail.
  - Suppose, for the sake of contradiction, that *TOTAL* is decidable.
  - Build a TM H that accepts ⟨M, w⟩ and constructs a TM M' that is a decider iff M halts on w.

What Just Happened?

- Let's walk through this construction in detail.
  - Suppose, for the sake of contradiction, that *TOTAL* is decidable.
  - Build a TM H that accepts ⟨M, w⟩ and constructs a TM M' that is a decider iff M halts on w.

This is the key step in most reductions. We build a TM that has a property of the new problem (here, *TOTAL*) based on whether some other TM has a property of the old problem (here, *HALT*)

Deciding whether this TM has the new property thus decides whether some other TM has the old property.

What Just Happened?

- Let's walk through this construction in detail.
  - Suppose, for the sake of contradiction, that *TOTAL* is decidable.
  - Build a TM H that accepts ⟨M, w⟩ and constructs a TM M' that is a decider iff M halts on w.
  - Using the decider for *TOTAL*, H checks whether M' is a decider:
    - If M' is a decider, then M halts on w.
    - If M' is not a decider, then M does not halt on w.

What Just Happened?

- Let's walk through this construction in detail.
  - Suppose, for the sake of contradiction, that *TOTAL* is decidable.
  - Build a TM H that accepts ⟨M, w⟩ and constructs a TM M' that is a decider iff M halts on w.
  - Using the decider for *TOTAL*, H checks whether M' is a decider:
    - If M' is a decider, then M halts on w.
    - If M' is not a decider, then M does not halt on w.
  - Conclude that if *TOTAL* is decidable, then *HALT* is decidable.

What Just Happened?

- Let's walk through this construction in detail.
  - Suppose, for the sake of contradiction, that *TOTAL* is decidable.
  - Build a TM H that accepts ⟨M, w⟩ and constructs a TM M' that is a decider iff M halts on w.
  - Using the decider for *TOTAL*, H checks whether M' is a decider:
    - If M' is a decider, then M halts on w.
    - If M' is not a decider, then M does not halt on w.
  - Conclude that if *TOTAL* is decidable, then *HALT* is decidable.
  - But *HALT* is undecidable, so our assumption was wrong and *TOTAL* is undecidable.

Theorem: TOTAL is undecidable.

Theorem: TOTAL is undecidable.
Proof: By contradiction; assume that *TOTAL* is decidable.

Theorem: TOTAL is undecidable.

Proof: By contradiction; assume that *TOTAL* is decidable. Let T be a decider for *TOTAL*.

Theorem: TOTAL is undecidable.

Proof: By contradiction; assume that *TOTAL* is decidable. Let T be a decider for *TOTAL*. Then consider the following TM:

Theorem: TOTAL is undecidable.
Proof: By contradiction; assume that *TOTAL* is decidable. Let T be a decider for *TOTAL*. Then consider the following TM:

H = “On input ⟨M, w⟩:

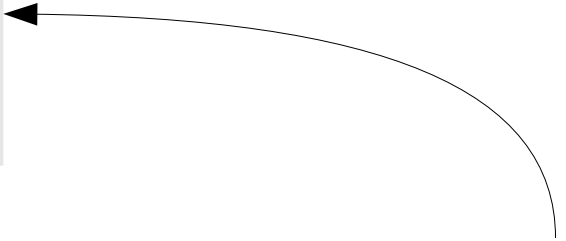
Theorem: TOTAL is undecidable.
Proof: By contradiction; assume that *TOTAL* is decidable. Let T be a decider for *TOTAL*. Then consider the following TM:

H = “On input ⟨M, w⟩:
Construct the TM M' = “On input x:
Ignore x.
Run M on w.
If M accepts w, accept.
If M rejects w, reject.”

Theorem: TOTAL is undecidable.
Proof: By contradiction; assume that *TOTAL* is decidable. Let T be a decider for *TOTAL*. Then consider the following TM:

H = “On input ⟨M, w⟩:

Construct the TM M' = “On input x:
Ignore x.
Run M on w.
If M accepts w, accept.
If M rejects w, reject.”



Most reduction proofs work by building a new TM out of an existing TM and a string. The new TM then has some property iff the old TM/string pair has some property.

Theorem: TOTAL is undecidable.
Proof: By contradiction; assume that *TOTAL* is decidable. Let T be a decider for *TOTAL*. Then consider the following TM:

H = “On input ⟨M, w⟩:
    Construct the TM M' = “On input x:
        Ignore x.
        Run M on w.
        If M accepts w, accept.
        If M rejects w, reject.”
    Run T on ⟨M'⟩.

Theorem: TOTAL is undecidable.
Proof: By contradiction; assume that *TOTAL* is decidable. Let T be a decider for *TOTAL*. Then consider the following TM:

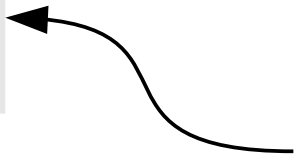
H = “On input ⟨M, w⟩:
    Construct the TM M' = “On input x:
        Ignore x.
        Run M on w.
        If M accepts w, accept.
        If M rejects w, reject.”
    Run T on ⟨M'⟩.
    If T accepts, accept.

Theorem: TOTAL is undecidable.
Proof: By contradiction; assume that *TOTAL* is decidable. Let T be a decider for *TOTAL*. Then consider the following TM:

H = “On input ⟨M, w⟩:
Construct the TM M' = “On input x:
Ignore x.
Run M on w.
If M accepts w, accept.
If M rejects w, reject.”
Run T on ⟨M'⟩.
If T accepts, accept.
If T rejects, reject.”

Theorem: *TOTAL* is undecidable.
Proof: By contradiction; assume that *TOTAL* is decidable. Let T be a decider for *TOTAL*. Then consider the following TM:

H = “On input ⟨M, w⟩:
Construct the TM M' = “On input x:
Ignore x.
Run M on w.
If M accepts w, accept.
If M rejects w, reject.”
Run T on ⟨M'⟩.
If T accepts, accept.
If T rejects, reject.”

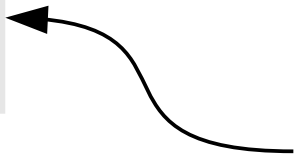


The behavior of H depends on what T does on ⟨M'⟩.

The behavior of T on ⟨M'⟩ depends on what M does on w.

Theorem: TOTAL is undecidable.
Proof: By contradiction; assume that *TOTAL* is decidable. Let T be a decider for *TOTAL*. Then consider the following TM:

H = “On input ⟨M, w⟩:
Construct the TM M' = “On input x:
Ignore x.
Run M on w.
If M accepts w, accept.
If M rejects w, reject.”
Run T on ⟨M'⟩.
If T accepts, accept.
If T rejects, reject.”



The behavior of H depends on what T does on ⟨M'⟩.

The behavior of T on ⟨M'⟩ depends on what M does on w.

So the behavior of TM H depends on what M does on w.

Theorem: TOTAL is undecidable.
Proof: By contradiction; assume that *TOTAL* is decidable. Let T be a decider for *TOTAL*. Then consider the following TM:

H = “On input ⟨M, w⟩:
    Construct the TM M' = “On input x:
        Ignore x.
        Run M on w.
        If M accepts w, accept.
        If M rejects w, reject.”
    Run T on ⟨M'⟩.
    If T accepts, accept.
    If T rejects, reject.”

We claim that H decides *HALT*.

Theorem: TOTAL is undecidable.
Proof: By contradiction; assume that *TOTAL* is decidable. Let T be a decider for *TOTAL*. Then consider the following TM:

H = “On input ⟨M, w⟩:
    Construct the TM M' = “On input x:
        Ignore x.
        Run M on w.
        If M accepts w, accept.
        If M rejects w, reject.”
    Run T on ⟨M'⟩.
    If T accepts, accept.
    If T rejects, reject.”

We claim that H decides *HALT*. To see this, we show that H is a decider and that L(H) = *HALT*.

Theorem: TOTAL is undecidable.
Proof: By contradiction; assume that *TOTAL* is decidable. Let T be a decider for *TOTAL*. Then consider the following TM:

H = “On input ⟨M, w⟩:
    Construct the TM M' = “On input x:
        Ignore x.
        Run M on w.
        If M accepts w, accept.
        If M rejects w, reject.”
    Run T on ⟨M'⟩.
    If T accepts, accept.
    If T rejects, reject.”

We claim that H decides *HALT*. To see this, we show that H is a decider and that L(H) = *HALT*. To see that H is a decider, note that after we construct M', we run T on ⟨M'⟩.

Theorem: *TOTAL* is undecidable.
Proof: By contradiction; assume that *TOTAL* is decidable. Let T be a decider for *TOTAL*. Then consider the following TM:

H = "On input ⟨M, w⟩:
    Construct the TM M' = "On input x:
        Ignore x.
        Run M on w.
        If M accepts w, accept.
        If M rejects w, reject."
    Run T on ⟨M'⟩.
    If T accepts, accept.
    If T rejects, reject."

We claim that H decides *HALT*. To see this, we show that H is a decider and that L(H) = *HALT*. To see that H is a decider, note that after we construct M', we run T on ⟨M'⟩.

To be completely formal in our proof, we should show that this construction can be done in finite time so that H does not loop infinitely trying to construct M'. However, conventionally this is just assumed to be true and you do not need to justify it.

Theorem: TOTAL is undecidable.
Proof: By contradiction; assume that *TOTAL* is decidable. Let T be a decider for *TOTAL*. Then consider the following TM:

H = “On input ⟨M, w⟩:
    Construct the TM M' = “On input x:
        Ignore x.
        Run M on w.
        If M accepts w, accept.
        If M rejects w, reject.”
    Run T on ⟨M'⟩.
    If T accepts, accept.
    If T rejects, reject.”

We claim that H decides *HALT*. To see this, we show that H is a decider and that L(H) = *HALT*. To see that H is a decider, note that after we construct M', we run T on ⟨M'⟩. Since T is a decider, it always halts, so H always halts.

Theorem: TOTAL is undecidable.
Proof: By contradiction; assume that *TOTAL* is decidable. Let T be a decider for *TOTAL*. Then consider the following TM:

H = “On input ⟨M, w⟩:
    Construct the TM M' = “On input x:
        Ignore x.
        Run M on w.
        If M accepts w, accept.
        If M rejects w, reject.”
    Run T on ⟨M'⟩.
    If T accepts, accept.
    If T rejects, reject.”

We claim that H decides *HALT*. To see this, we show that H is a decider and that L(H) = *HALT*. To see that H is a decider, note that after we construct M', we run T on ⟨M'⟩. Since T is a decider, it always halts, so H always halts.

To see that L(H) = *HALT*, note that H accepts ⟨M, w⟩ iff T accepts ⟨M'⟩.

Theorem: TOTAL is undecidable.
Proof: By contradiction; assume that *TOTAL* is decidable. Let T be a decider for *TOTAL*. Then consider the following TM:

H = “On input ⟨M, w⟩:
    Construct the TM M' = “On input x:
        Ignore x.
        Run M on w.
        If M accepts w, accept.
        If M rejects w, reject.”
    Run T on ⟨M'⟩.
    If T accepts, accept.
    If T rejects, reject.”

We claim that H decides *HALT*. To see this, we show that H is a decider and that L(H) = *HALT*. To see that H is a decider, note that after we construct M', we run T on ⟨M'⟩. Since T is a decider, it always halts, so H always halts.

To see that L(H) = *HALT*, note that H accepts ⟨M, w⟩ iff T accepts ⟨M'⟩. Because T is a decider for *TOTAL*, T accepts ⟨M'⟩ iff M' halts on all inputs.

Theorem: TOTAL is undecidable.
Proof: By contradiction; assume that *TOTAL* is decidable. Let T be a decider for *TOTAL*. Then consider the following TM:

H = “On input ⟨M, w⟩:
    Construct the TM M' = “On input x:
        Ignore x.
        Run M on w.
        If M accepts w, accept.
        If M rejects w, reject.”
    Run T on ⟨M'⟩.
    If T accepts, accept.
    If T rejects, reject.”

We claim that H decides *HALT*. To see this, we show that H is a decider and that L(H) = *HALT*. To see that H is a decider, note that after we construct M', we run T on ⟨M'⟩. Since T is a decider, it always halts, so H always halts.

To see that L(H) = *HALT*, note that H accepts ⟨M, w⟩ iff T accepts ⟨M'⟩. Because T is a decider for *TOTAL*, T accepts ⟨M'⟩ iff M' halts on all inputs. By construction, M' halts on any input iff M halts on w.

Theorem: TOTAL is undecidable.
Proof: By contradiction; assume that *TOTAL* is decidable. Let T be a decider for *TOTAL*. Then consider the following TM:

H = “On input ⟨M, w⟩:
    Construct the TM M' = “On input x:
        Ignore x.
        Run M on w.
        If M accepts w, accept.
        If M rejects w, reject.”
    Run T on ⟨M'⟩.
    If T accepts, accept.
    If T rejects, reject.”

We claim that H decides *HALT*. To see this, we show that H is a decider and that L(H) = *HALT*. To see that H is a decider, note that after we construct M', we run T on ⟨M'⟩. Since T is a decider, it always halts, so H always halts.

To see that L(H) = *HALT*, note that H accepts ⟨M, w⟩ iff T accepts ⟨M'⟩. Because T is a decider for *TOTAL*, T accepts ⟨M'⟩ iff M' halts on all inputs. By construction, M' halts on any input iff M halts on w. Finally, M halts on w iff ⟨M, w⟩ ∈ *HALT*.

Theorem: TOTAL is undecidable.
Proof: By contradiction; assume that *TOTAL* is decidable. Let T be a decider for *TOTAL*. Then consider the following TM:

H = “On input ⟨M, w⟩:
Construct the TM M' = “On input x:
Ignore x.
Run M on w.
If M accepts w, accept.
If M rejects w, reject.”
Run T on ⟨M'⟩.
If T accepts, accept.
If T rejects, reject.”

We claim that H decides *HALT*. To see this, we show that H is a decider and that L(H) = *HALT*. To see that H is a decider, note that after we construct M', we run T on ⟨M'⟩. Since T is a decider, it always halts, so H always halts.

To see that L(H) = *HALT*, note that H accepts ⟨M, w⟩ iff T accepts ⟨M'⟩. Because T is a decider for *TOTAL*, T accepts ⟨M'⟩ iff M' halts on all inputs. By construction, M' halts on any input iff M halts on w. Finally, M halts on w iff ⟨M, w⟩ ∈ *HALT*. This means that H accepts ⟨M, w⟩ iff ⟨M, w⟩ ∈ *HALT*, so L(H) = *HALT*.

Theorem: TOTAL is undecidable.
Proof: By contradiction; assume that *TOTAL* is decidable. Let T be a decider for *TOTAL*. Then consider the following TM:

H = “On input ⟨M, w⟩:
Construct the TM M' = “On input x:
Ignore x.
Run M on w.
If M accepts w, accept.
If M rejects w, reject.”
Run T on ⟨M'⟩.
If T accepts, accept.
If T rejects, reject.”

We claim that H decides *HALT*. To see this, we show that H is a decider and that L(H) = *HALT*. To see that H is a decider, note that after we construct M', we run T on ⟨M'⟩. Since T is a decider, it always halts, so H always halts.

To see that L(H) = *HALT*, note that H accepts ⟨M, w⟩ iff T accepts ⟨M'⟩. Because T is a decider for *TOTAL*, T accepts ⟨M'⟩ iff M' halts on all inputs. By construction, M' halts on any input iff M halts on w. Finally, M halts on w iff ⟨M, w⟩ ∈ *HALT*. This means that H accepts ⟨M, w⟩ iff ⟨M, w⟩ ∈ *HALT*, so L(H) = *HALT*.

We have reached a contradiction, because H decides *HALT*, which we know is undecidable.

Theorem: TOTAL is undecidable.
Proof: By contradiction; assume that *TOTAL* is decidable. Let T be a decider for *TOTAL*. Then consider the following TM:

H = "On input ⟨M, w⟩:
    Construct the TM M' = "On input x:
        Ignore x.
        Run M on w.
        If M accepts w, accept.
        If M rejects w, reject."
    Run T on ⟨M'⟩.
    If T accepts, accept.
    If T rejects, reject."

We claim that H decides *HALT*. To see this, we show that H is a decider and that L(H) = *HALT*. To see that H is a decider, note that after we construct M', we run T on ⟨M'⟩. Since T is a decider, it always halts, so H always halts.

To see that L(H) = *HALT*, note that H accepts ⟨M, w⟩ iff T accepts ⟨M'⟩. Because T is a decider for *TOTAL*, T accepts ⟨M'⟩ iff M' halts on all inputs. By construction, M' halts on any input iff M halts on w. Finally, M halts on w iff ⟨M, w⟩ ∈ *HALT*. This means that H accepts ⟨M, w⟩ iff ⟨M, w⟩ ∈ *HALT*, so L(H) = *HALT*.

We have reached a contradiction, because H decides *HALT*, which we know is undecidable. Thus our assumption was wrong and *TOTAL* is undecidable.

Theorem: TOTAL is undecidable.
Proof: By contradiction; assume that *TOTAL* is decidable. Let T be a decider for *TOTAL*. Then consider the following TM:

H = "On input ⟨M, w⟩:
    Construct the TM M' = "On input x:
        Ignore x.
        Run M on w.
        If M accepts w, accept.
        If M rejects w, reject."
    Run T on ⟨M'⟩.
    If T accepts, accept.
    If T rejects, reject."

We claim that H decides *HALT*. To see this, we show that H is a decider and that L(H) = *HALT*. To see that H is a decider, note that after we construct M', we run T on ⟨M'⟩. Since T is a decider, it always halts, so H always halts.

To see that L(H) = *HALT*, note that H accepts ⟨M, w⟩ iff T accepts ⟨M'⟩. Because T is a decider for *TOTAL*, T accepts ⟨M'⟩ iff M' halts on all inputs. By construction, M' halts on any input iff M halts on w. Finally, M halts on w iff ⟨M, w⟩ ∈ *HALT*. This means that H accepts ⟨M, w⟩ iff ⟨M, w⟩ ∈ *HALT*, so L(H) = *HALT*.

We have reached a contradiction, because H decides *HALT*, which we know is undecidable. Thus our assumption was wrong and *TOTAL* is undecidable. ■

Testing Regularity

- We know that every regular language is RE.
- Thus some TMs must recognize regular languages.
- Could we detect whether a TM recognizes a regular language?

Testing Regularity

- Let

$$REGULAR_{TM} = \{ \langle M \rangle \mid L(M) \text{ is regular.} \}$$

- Unlike *TOTAL*, this doesn't seem to have any obvious connection to *HALT*.
- Is $REGULAR_{TM}$ decidable?

Testing Regularity

- Let

$$REGULAR_{TM} = \{ \langle M \rangle \mid L(M) \text{ is regular.} \}$$

- Unlike *TOTAL*, this doesn't seem to have any obvious connection to *HALT*.
- Is $REGULAR_{TM}$ decidable?
- Unfortunately, **no**, via a reduction from *HALT*.

$REGULAR_{TM}$ is Undecidable

- **Proof idea:** Suppose that $REGULAR_{TM}$ is decidable by some machine D.

$REGULAR_{TM}$ is Undecidable

- **Proof idea:** Suppose that $REGULAR_{TM}$ is decidable by some machine D.
- Given a TM M and a string w, construct a TM M' with these properties:
  - If M halts on w, then L(M') is regular.
  - If M loops on w, then L(M') is not regular.

$REGULAR_{TM}$ is Undecidable

- **Proof idea:** Suppose that $REGULAR_{TM}$ is decidable by some machine D.
- Given a TM M and a string w, construct a TM M' with these properties:
  - If M halts on w, then L(M') is regular.
  - If M loops on w, then L(M') is not regular.

How do we build a machine with these properties?

$REGULAR_{TM}$ is Undecidable

- **Proof idea:** Suppose that $REGULAR_{TM}$ is decidable by some machine D.
- Given a TM M and a string w, construct a TM M' with these properties:
  - If M halts on w, then **L(M') is regular.**
  - If M loops on w, then L(M') is not regular.

$REGULAR_{TM}$ is Undecidable

- **Proof idea:** Suppose that $REGULAR_{TM}$ is decidable by some machine D.
- Given a TM M and a string w, construct a TM M' with these properties:
  - If M halts on w, then **L(M') = Σ***.
  - If M loops on w, then L(M') is not regular.

$REGULAR_{TM}$ is Undecidable

- **Proof idea:** Suppose that $REGULAR_{TM}$ is decidable by some machine D.
- Given a TM M and a string w, construct a TM M' with these properties:
  - If M halts on w, then L(M') = Σ*.
  - If M loops on w, then **L(M') is not regular**.

$REGULAR_{TM}$ is Undecidable

- **Proof idea:** Suppose that $REGULAR_{TM}$ is decidable by some machine D.
- Given a TM M and a string w, construct a TM M' with these properties:
  - If M halts on w, then L(M') = Σ*.
  - If M loops on w, then **$L(M') = \{0^n1^n \mid n \in \mathbb{N}\}$**.

$REGULAR_{TM}$ is Undecidable

- **Proof idea:** Suppose that $REGULAR_{TM}$ is decidable by some machine D.
- Given a TM M and a string w, construct a TM M' with these properties:
  - If M halts on w, then L(M') = Σ*.
  - If M loops on w, then L(M') = $\{0^n1^n \mid n \in \mathbb{N}\}$.

$REGULAR_{TM}$ is Undecidable

- **Proof idea:** Suppose that $REGULAR_{TM}$ is decidable by some machine D.
- Given a TM M and a string w, construct a TM M' with these properties:
  - If M halts on w, then L(M') = Σ*.
  - If M loops on w, then L(M') = $\{0^n1^n \mid n \in \mathbb{N}\}$.
- Have D decide whether or not L(M') is regular.
  - If L(M') is regular, M halts on w.
  - If L(M') is not regular, M loops on w.

$REGULAR_{TM}$ is Undecidable

- **Proof idea:** Suppose that $REGULAR_{TM}$ is decidable by some machine D.
- Given a TM M and a string w, construct a TM M' with these properties:
  - If M halts on w, then L(M') = Σ*.
  - If M loops on w, then L(M') = $\{0^n1^n \mid n \in \mathbb{N}\}$.
- Have D decide whether or not L(M') is regular.
  - If L(M') is regular, M halts on w.
  - If L(M') is not regular, M loops on w.
- We can use D to decide *HALT*, which is a contradiction.

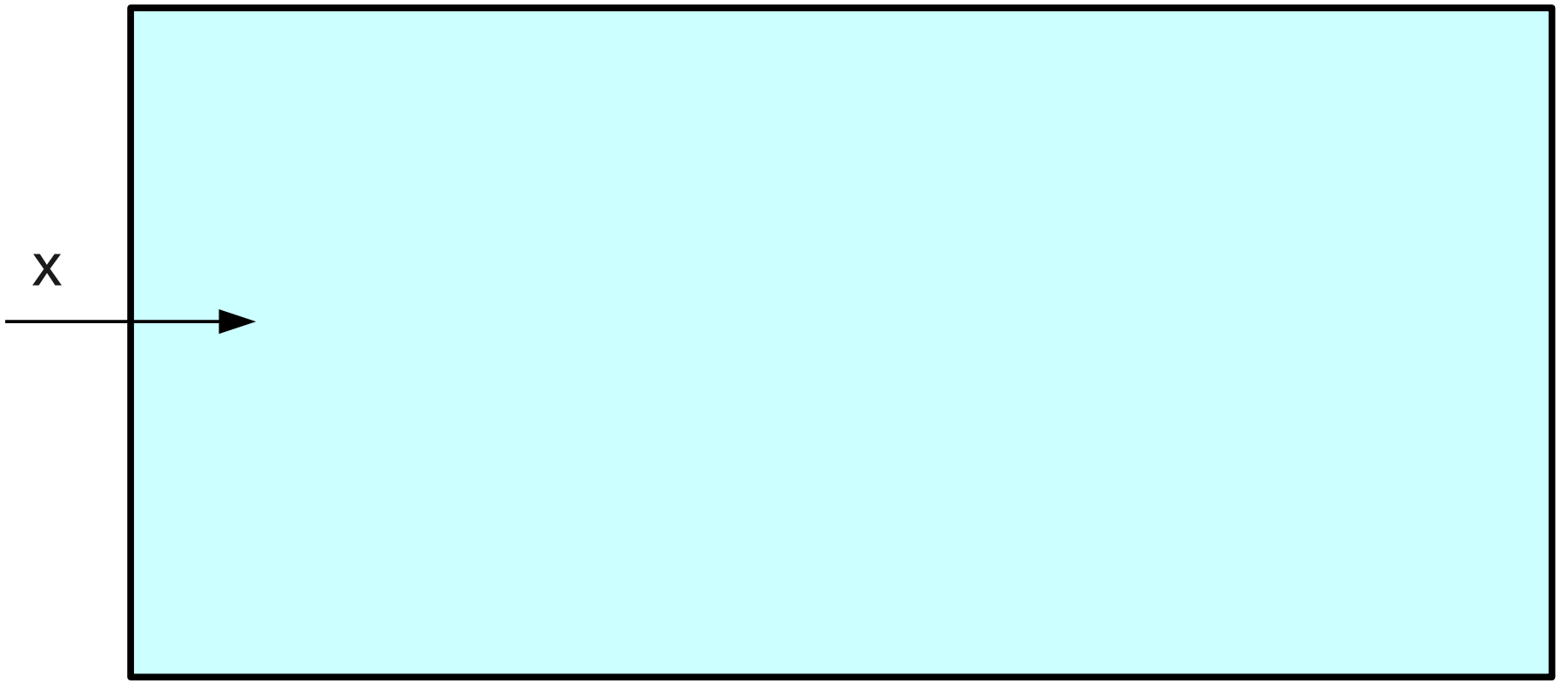
$REGULAR_{TM}$ is Undecidable

- **Proof idea:** Suppose that $REGULAR_{TM}$ is decidable by some machine D.
- Given a TM M and a string w, construct a TM M' with these properties:
  - If M halts on w, then L(M') = Σ*.
  - If M loops on w, then L(M') = $\{0^n1^n \mid n \in \mathbb{N}\}$.
- Have D decide whether or not L(M') is regular.
  - If L(M') is regular, M halts on w.
  - If L(M') is not regular, M loops on w.
- We can use D to decide *HALT*, which is a contradiction.

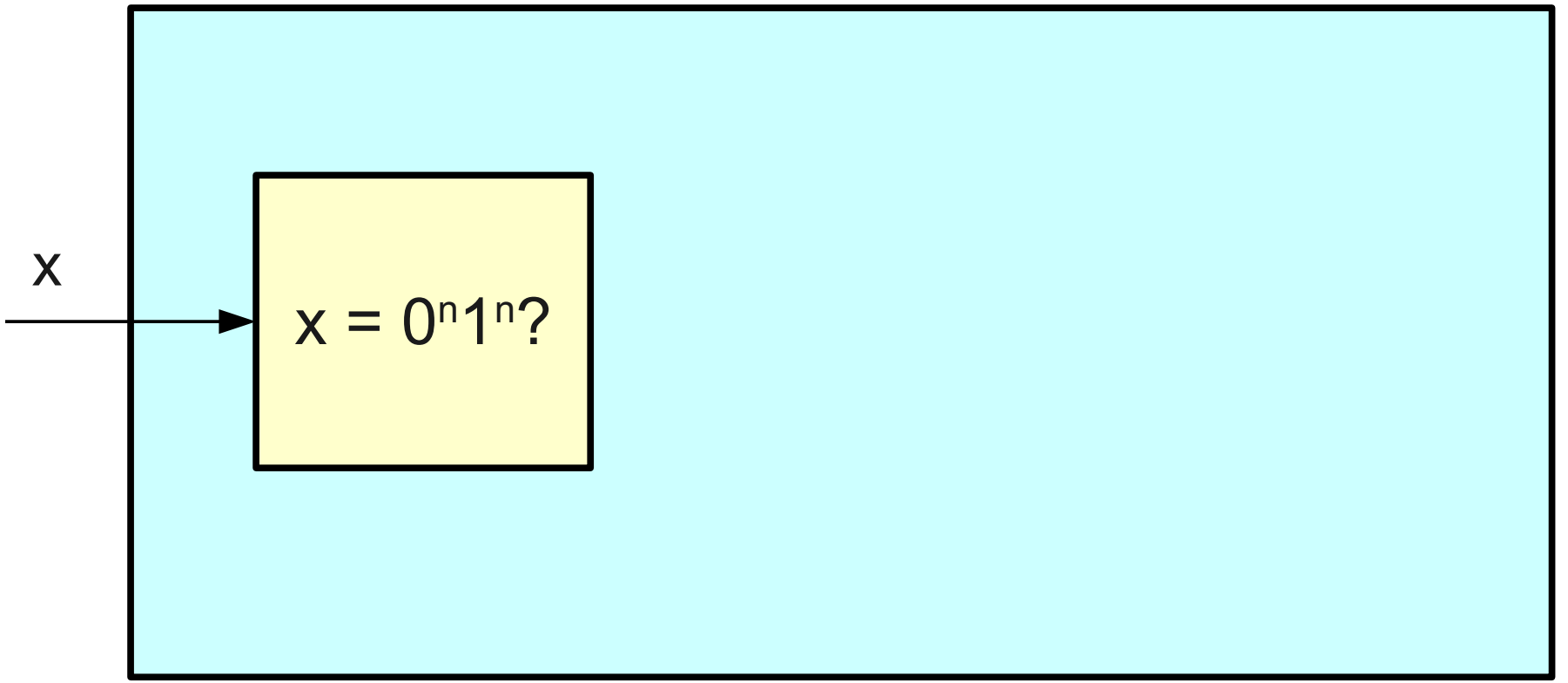
How do we build a machine with these properties?

The Mysterious Machine

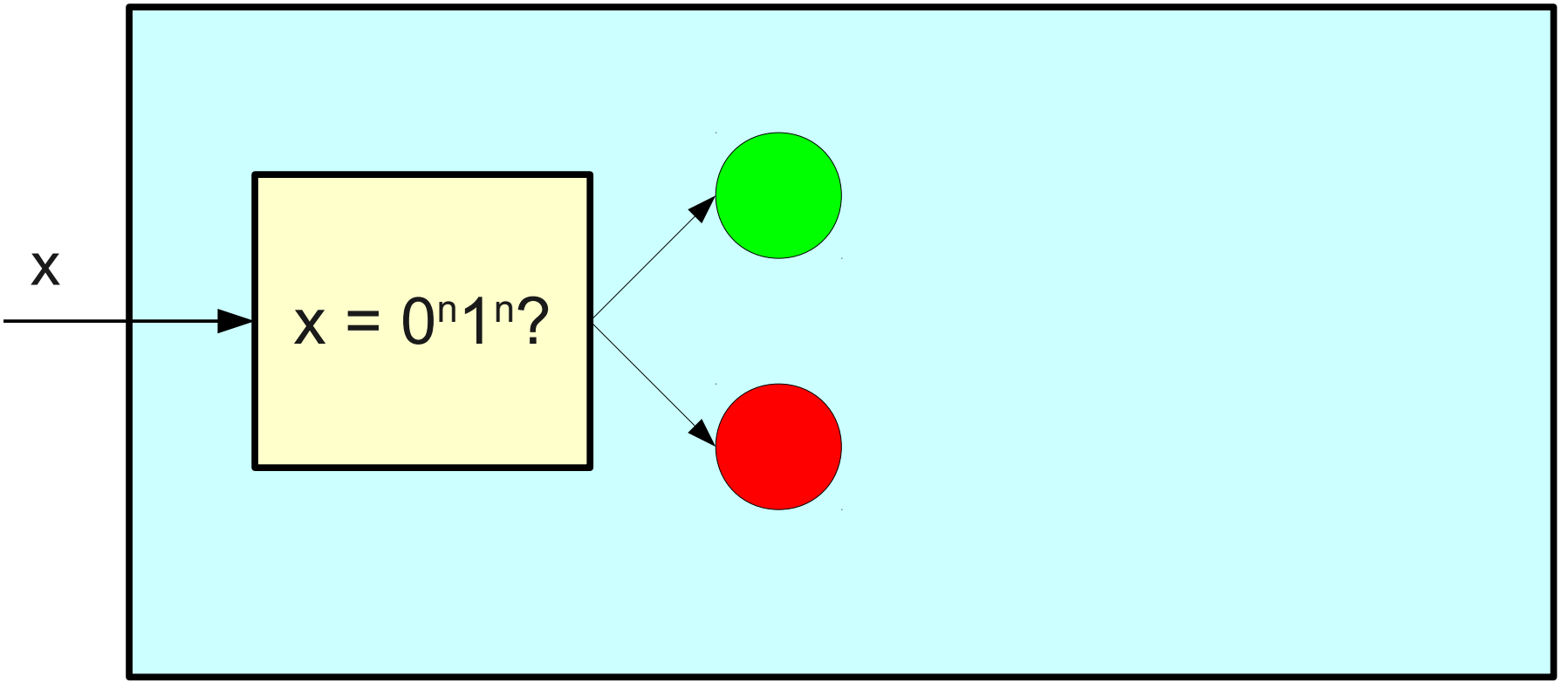
The Mysterious Machine



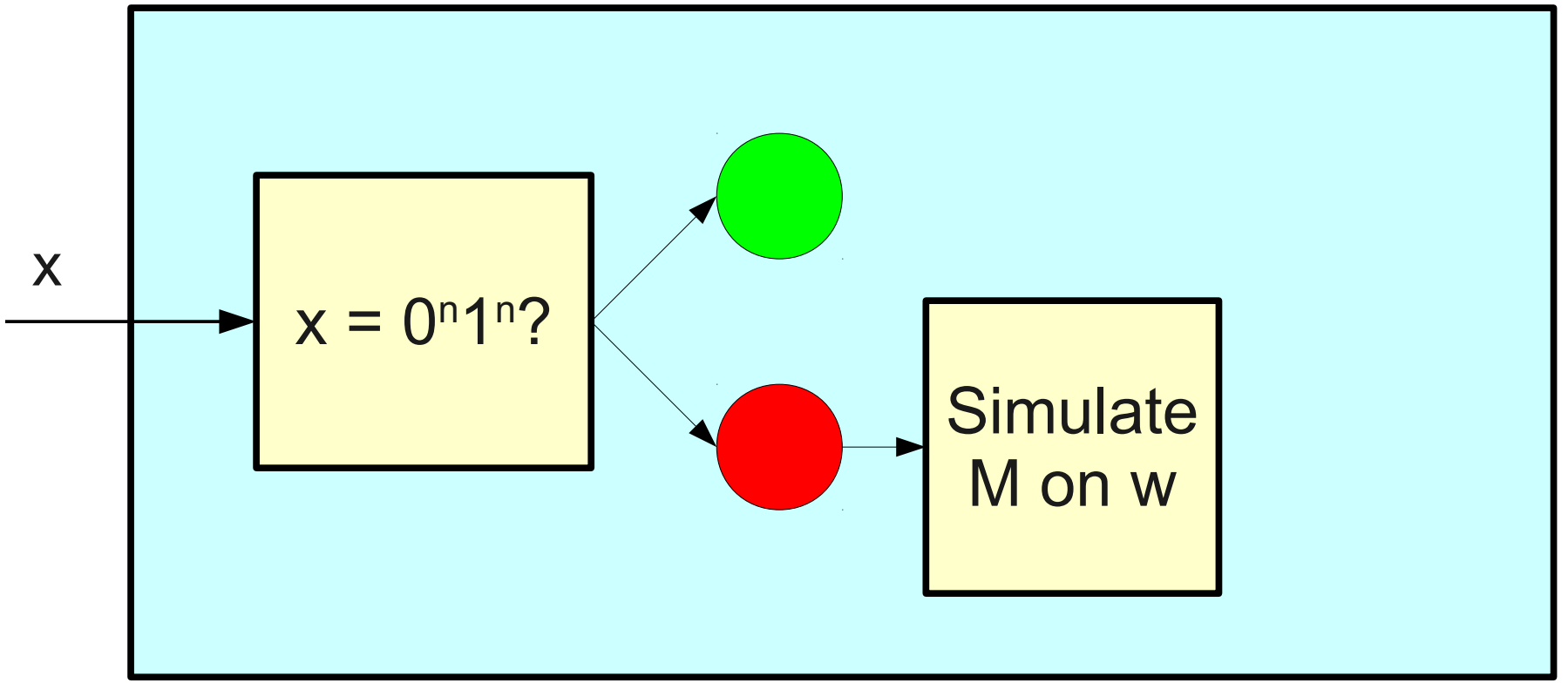
The Mysterious Machine



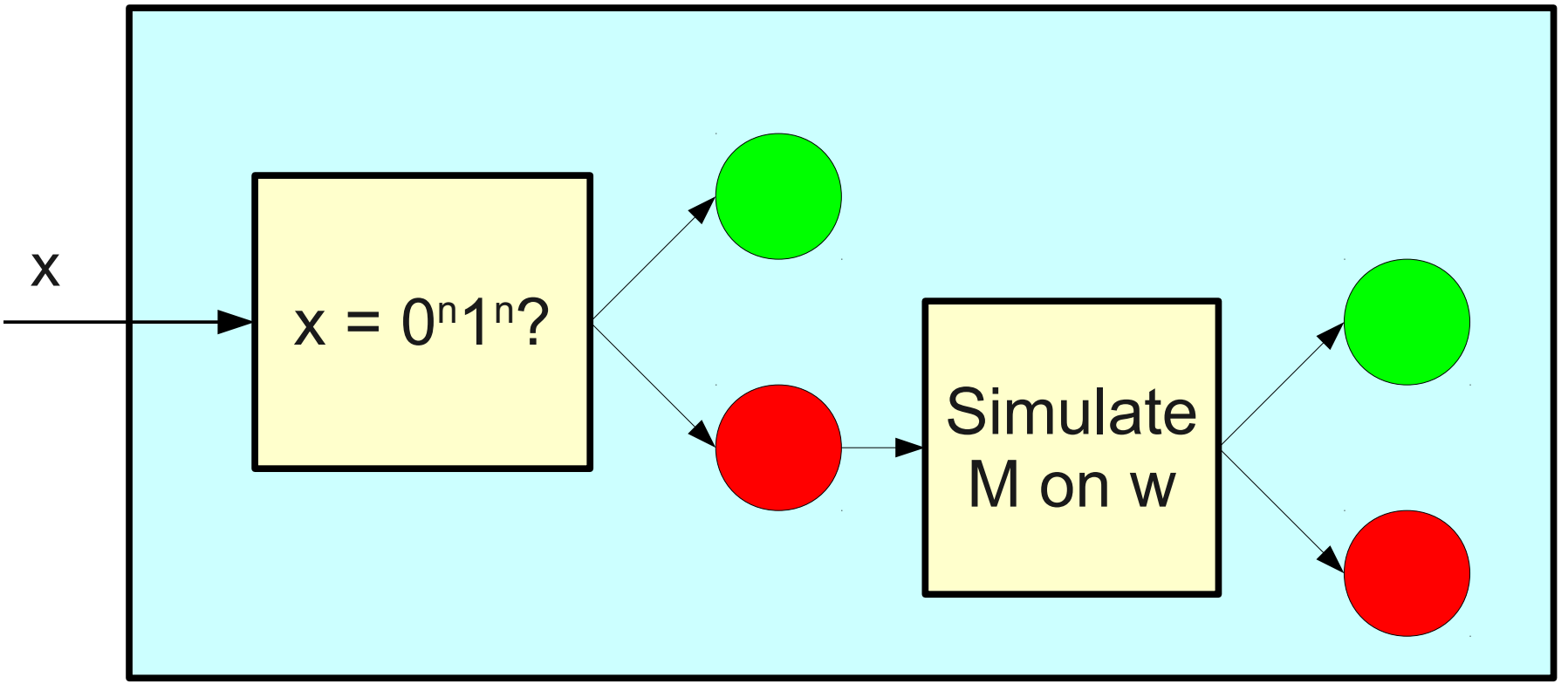
The Mysterious Machine



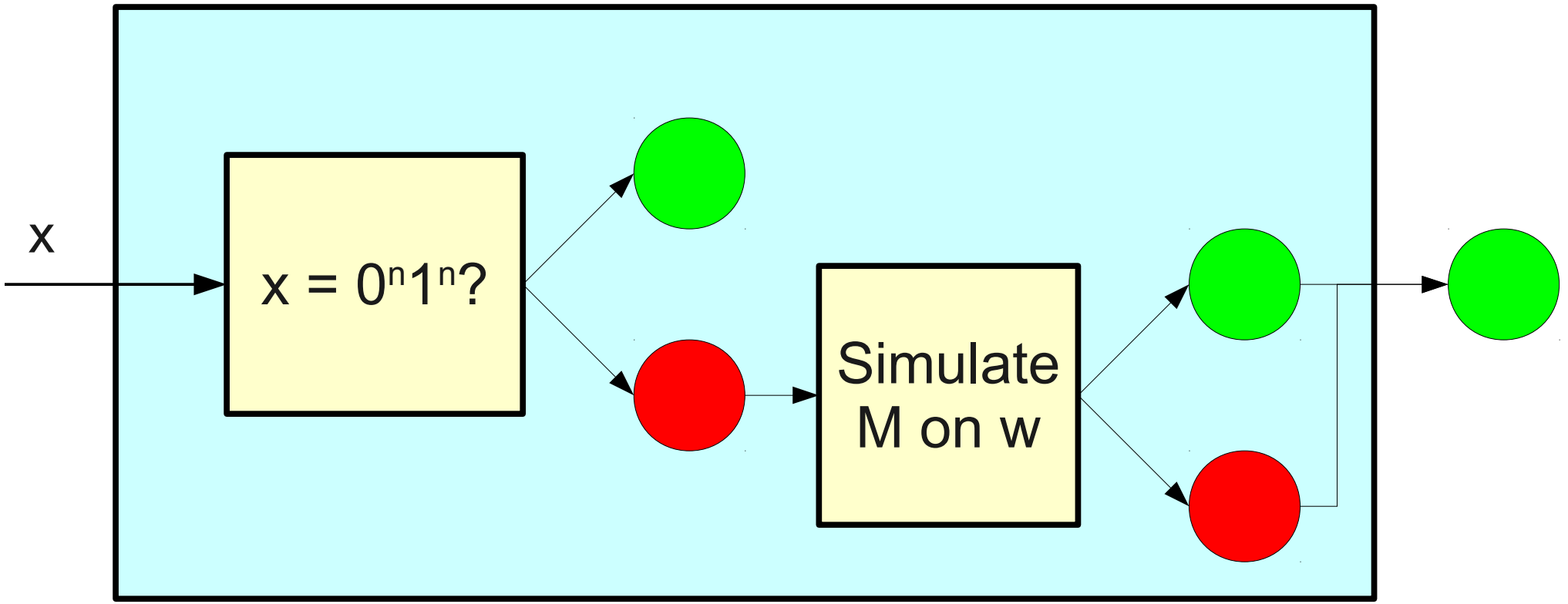
The Mysterious Machine



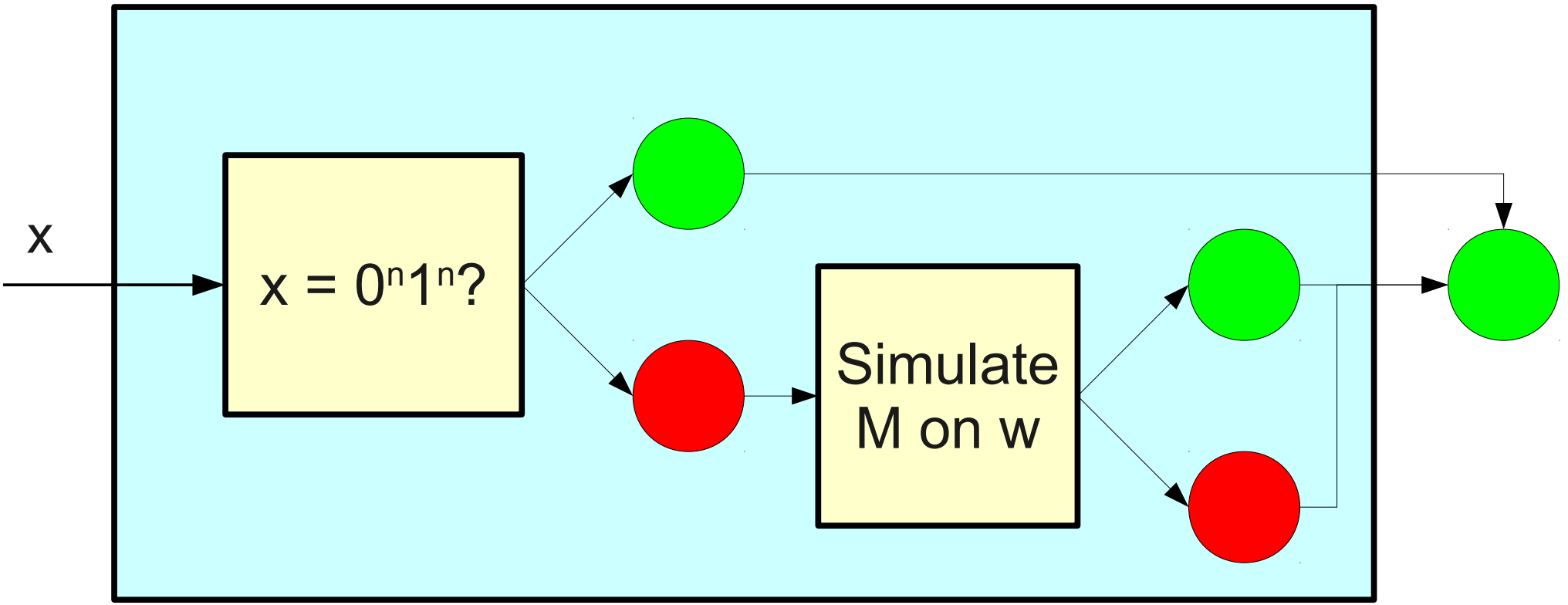
The Mysterious Machine



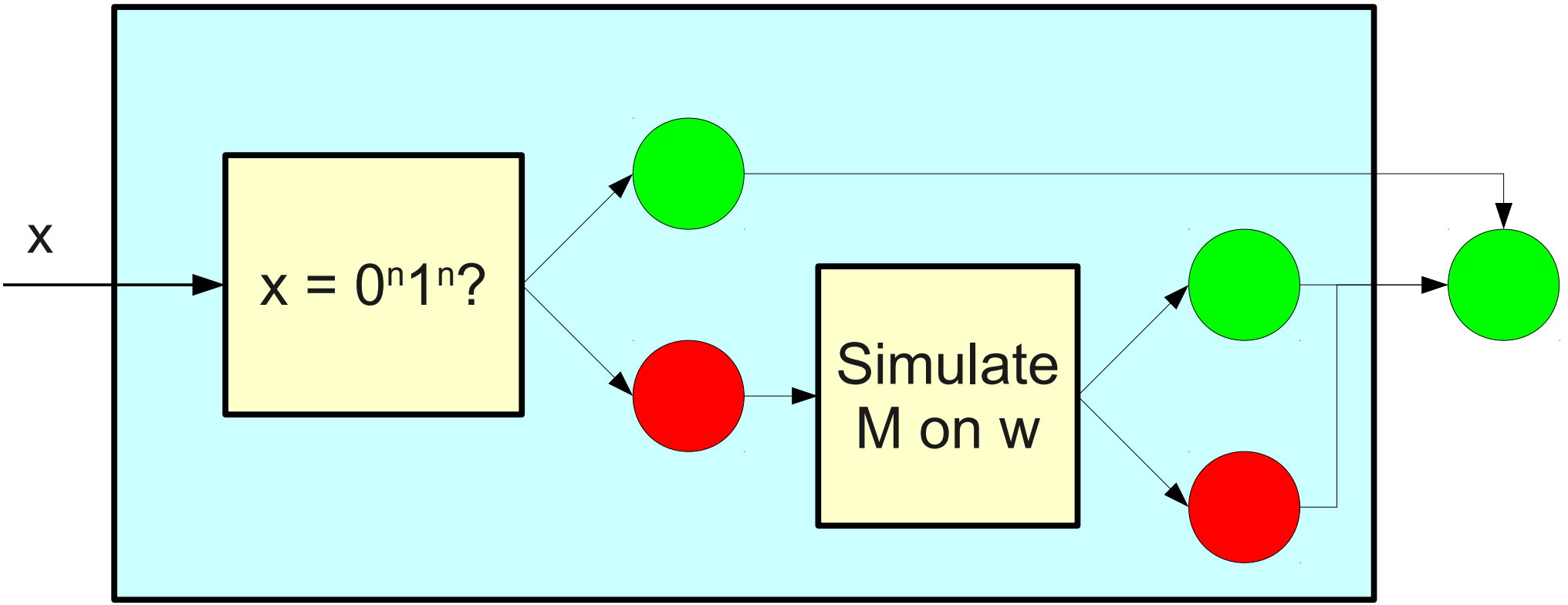
The Mysterious Machine



The Mysterious Machine

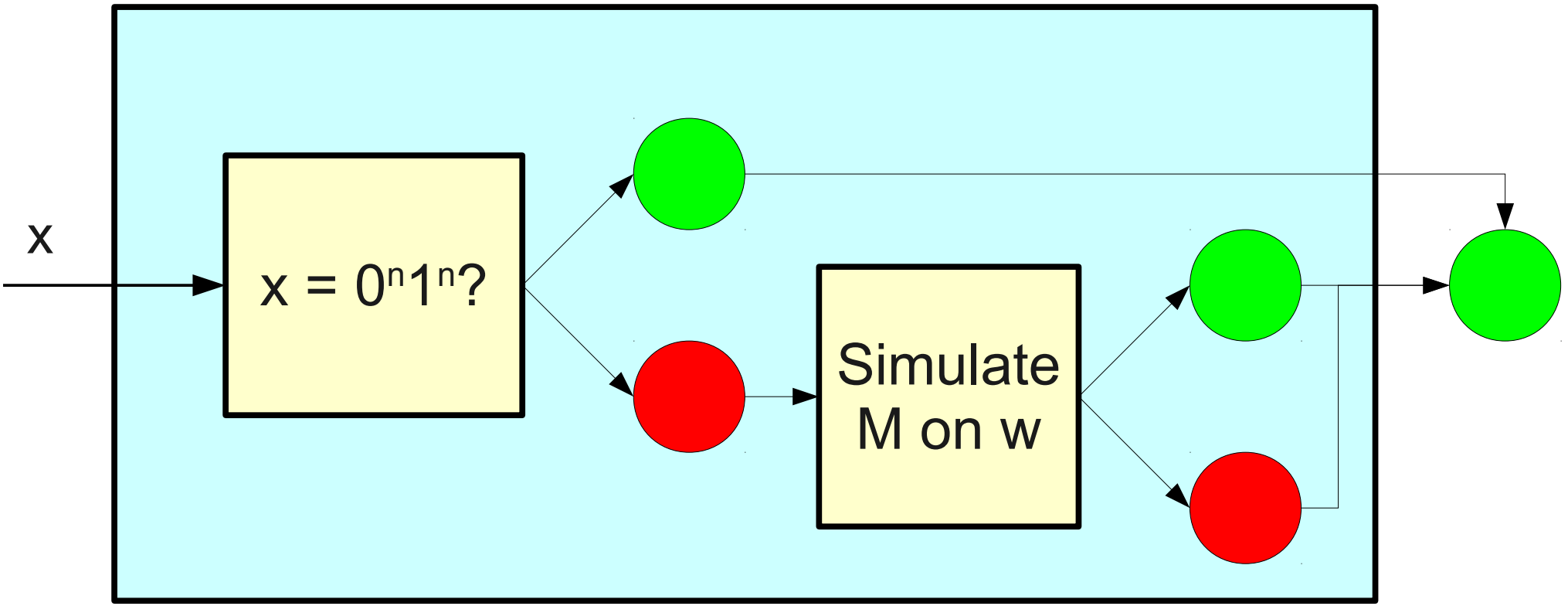


The Mysterious Machine



M' = “On input x:
If x = 0^n1^n, accept.
Otherwise, run M on w.
If M halts, accept.”

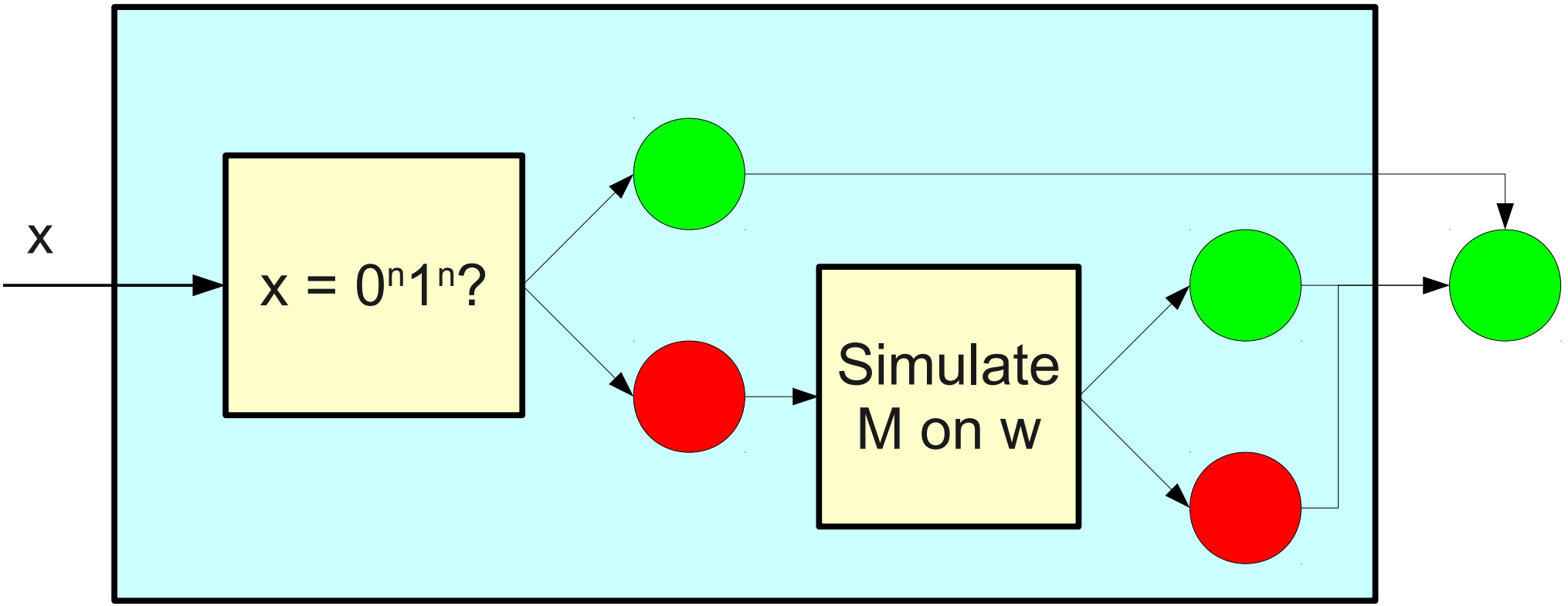
The Mysterious Machine



M' = “On input x:
If $x = 0^n1^n$, accept.
Otherwise, run M on w.
If M halts, accept.”

If M halts on w, M' accepts all strings.

The Mysterious Machine



M' = “On input x:
If $x = 0^n1^n$, accept.
Otherwise, run M on w.
If M halts, accept.”

If M halts on w, M' accepts all strings.

If M loops on w, M' only accepts strings of the form 0^n1^n.

Theorem: $REGULAR_{TM}$ is undecidable.

Theorem: $REGULAR_{TM}$ is undecidable.

Proof: By contradiction; assume D decides $REGULAR_{TM}$.

Theorem: $REGULAR_{TM}$ is undecidable.

Proof: By contradiction; assume D decides $REGULAR_{TM}$. Consider the following machine H:

Theorem: $REGULAR_{TM}$ is undecidable.

Proof: By contradiction; assume D decides $REGULAR_{TM}$. Consider the following machine H:

H = “On input ⟨M, w⟩:

Theorem: $REGULAR_{TM}$ is undecidable.

Proof: By contradiction; assume D decides $REGULAR_{TM}$. Consider the following machine H:

H = “On input ⟨M, w⟩:

Construct the machine M' = “On input x:

If x has the form 0^n1^n, accept.

Otherwise, run M on w.

If M halts on w, accept.”

Theorem: $REGULAR_{TM}$ is undecidable.

Proof: By contradiction; assume D decides $REGULAR_{TM}$. Consider the following machine H:

H = “On input ⟨M, w⟩:
    Construct the machine M' = “On input x:
        If x has the form 0^n1^n, accept.
        Otherwise, run M on w.
        If M halts on w, accept.”
    Run D on ⟨M'⟩

Theorem: $REGULAR_{TM}$ is undecidable.

Proof: By contradiction; assume D decides $REGULAR_{TM}$. Consider the following machine H:

H = “On input ⟨M, w⟩:

Construct the machine M' = “On input x:

If x has the form 0^n1^n, accept.

Otherwise, run M on w.

If M halts on w, accept.”

Run D on ⟨M'⟩

If D accepts, accept; if D rejects, reject.”

Theorem: $REGULAR_{TM}$ is undecidable.

Proof: By contradiction; assume D decides $REGULAR_{TM}$. Consider the following machine H:

H = "On input ⟨M, w⟩:
    Construct the machine M' = "On input x:
        If x has the form 0^n1^n, accept.
        Otherwise, run M on w.
        If M halts on w, accept."
    Run D on ⟨M'⟩
    If D accepts, accept; if D rejects, reject."

We claim that H is a decider and that L(H) = *HALT*.

Theorem: $REGULAR_{TM}$ is undecidable.

Proof: By contradiction; assume D decides $REGULAR_{TM}$. Consider the following machine H:

H = "On input ⟨M, w⟩:
    Construct the machine M' = "On input x:
        If x has the form 0^n1^n, accept.
        Otherwise, run M on w.
        If M halts on w, accept."
    Run D on ⟨M'⟩
    If D accepts, accept; if D rejects, reject."

We claim that H is a decider and that L(H) = *HALT*. To see that H is a decider, note that after H constructs M', H runs D on ⟨M'⟩.

Theorem: $REGULAR_{TM}$ is undecidable.

Proof: By contradiction; assume D decides $REGULAR_{TM}$. Consider the following machine H:

H = "On input ⟨M, w⟩:
    Construct the machine M' = "On input x:
        If x has the form 0^n1^n, accept.
        Otherwise, run M on w.
        If M halts on w, accept."
    Run D on ⟨M'⟩
    If D accepts, accept; if D rejects, reject."

We claim that H is a decider and that L(H) = *HALT*. To see that H is a decider, note that after H constructs M', H runs D on ⟨M'⟩. Since D is a decider, D always halts.

Theorem: $REGULAR_{TM}$ is undecidable.

Proof: By contradiction; assume D decides $REGULAR_{TM}$. Consider the following machine H:

H = "On input ⟨M, w⟩:
    Construct the machine M' = "On input x:
        If x has the form 0^n1^n, accept.
        Otherwise, run M on w.
        If M halts on w, accept."
    Run D on ⟨M'⟩
    If D accepts, accept; if D rejects, reject."

We claim that H is a decider and that L(H) = *HALT*. To see that H is a decider, note that after H constructs M', H runs D on ⟨M'⟩. Since D is a decider, D always halts. If D accepts, H accepts, and if D rejects, H rejects.

Theorem: $REGULAR_{TM}$ is undecidable.

Proof: By contradiction; assume D decides $REGULAR_{TM}$. Consider the following machine H:

H = "On input ⟨M, w⟩:
    Construct the machine M' = "On input x:
        If x has the form 0^n1^n, accept.
        Otherwise, run M on w.
        If M halts on w, accept."
    Run D on ⟨M'⟩
    If D accepts, accept; if D rejects, reject."

We claim that H is a decider and that L(H) = *HALT*. To see that H is a decider, note that after H constructs M', H runs D on ⟨M'⟩. Since D is a decider, D always halts. If D accepts, H accepts, and if D rejects, H rejects. Thus H halts on all inputs.

Theorem: $REGULAR_{TM}$ is undecidable.

Proof: By contradiction; assume D decides $REGULAR_{TM}$. Consider the following machine H:

H = "On input ⟨M, w⟩:
    Construct the machine M' = "On input x:
        If x has the form 0^n1^n, accept.
        Otherwise, run M on w.
        If M halts on w, accept."
    Run D on ⟨M'⟩
    If D accepts, accept; if D rejects, reject."

We claim that H is a decider and that L(H) = *HALT*. To see that H is a decider, note that after H constructs M', H runs D on ⟨M'⟩. Since D is a decider, D always halts. If D accepts, H accepts, and if D rejects, H rejects. Thus H halts on all inputs.

To see that L(H) = *HALT*, note that H accepts ⟨M, w⟩ iff D accepts ⟨M'⟩.

Theorem: $REGULAR_{TM}$ is undecidable.

Proof: By contradiction; assume D decides $REGULAR_{TM}$. Consider the following machine H:

H = "On input ⟨M, w⟩:
    Construct the machine M' = "On input x:
        If x has the form 0^n1^n, accept.
        Otherwise, run M on w.
        If M halts on w, accept."
    Run D on ⟨M'⟩
    If D accepts, accept; if D rejects, reject."

We claim that H is a decider and that L(H) = *HALT*. To see that H is a decider, note that after H constructs M', H runs D on ⟨M'⟩. Since D is a decider, D always halts. If D accepts, H accepts, and if D rejects, H rejects. Thus H halts on all inputs.

To see that L(H) = *HALT*, note that H accepts ⟨M, w⟩ iff D accepts ⟨M'⟩. Since D decides $REGULAR_{TM}$, D accepts ⟨M'⟩ iff L(M') is regular.

Theorem: $REGULAR_{TM}$ is undecidable.

Proof: By contradiction; assume D decides $REGULAR_{TM}$. Consider the following machine H:

H = "On input ⟨M, w⟩:
    Construct the machine M' = "On input x:
        If x has the form 0^n1^n, accept.
        Otherwise, run M on w.
        If M halts on w, accept."
    Run D on ⟨M'⟩
    If D accepts, accept; if D rejects, reject."

We claim that H is a decider and that L(H) = *HALT*. To see that H is a decider, note that after H constructs M', H runs D on ⟨M'⟩. Since D is a decider, D always halts. If D accepts, H accepts, and if D rejects, H rejects. Thus H halts on all inputs.

To see that L(H) = *HALT*, note that H accepts ⟨M, w⟩ iff D accepts ⟨M'⟩. Since D decides $REGULAR_{TM}$, D accepts ⟨M'⟩ iff L(M') is regular. We claim that L(M') is regular iff M halts on w.

Theorem: $REGULAR_{TM}$ is undecidable.

Proof: By contradiction; assume D decides $REGULAR_{TM}$. Consider the following machine H:

H = "On input ⟨M, w⟩:
    Construct the machine M' = "On input x:
        If x has the form 0^n1^n, accept.
        Otherwise, run M on w.
        If M halts on w, accept."
    Run D on ⟨M'⟩
    If D accepts, accept; if D rejects, reject."

We claim that H is a decider and that L(H) = *HALT*. To see that H is a decider, note that after H constructs M', H runs D on ⟨M'⟩. Since D is a decider, D always halts. If D accepts, H accepts, and if D rejects, H rejects. Thus H halts on all inputs.

To see that L(H) = *HALT*, note that H accepts ⟨M, w⟩ iff D accepts ⟨M'⟩. Since D decides $REGULAR_{TM}$, D accepts ⟨M'⟩ iff L(M') is regular. We claim that L(M') is regular iff M halts on w. To see this, note that if M halts on w, M' accepts all strings, either because the string has form 0^n1^n or because it accepts in the final step after M halts.

Theorem: $REGULAR_{TM}$ is undecidable.

Proof: By contradiction; assume D decides $REGULAR_{TM}$. Consider the following machine H:

H = "On input ⟨M, w⟩:
    Construct the machine M' = "On input x:
        If x has the form 0^n1^n, accept.
        Otherwise, run M on w.
        If M halts on w, accept."
    Run D on ⟨M'⟩
    If D accepts, accept; if D rejects, reject."

We claim that H is a decider and that L(H) = *HALT*. To see that H is a decider, note that after H constructs M', H runs D on ⟨M'⟩. Since D is a decider, D always halts. If D accepts, H accepts, and if D rejects, H rejects. Thus H halts on all inputs.

To see that L(H) = *HALT*, note that H accepts ⟨M, w⟩ iff D accepts ⟨M'⟩. Since D decides $REGULAR_{TM}$, D accepts ⟨M'⟩ iff L(M') is regular. We claim that L(M') is regular iff M halts on w. To see this, note that if M halts on w, M' accepts all strings, either because the string has form 0^n1^n or because it accepts in the final step after M halts. Thus L(M') = Σ*, which is regular.

Theorem: $REGULAR_{TM}$ is undecidable.

Proof: By contradiction; assume D decides $REGULAR_{TM}$. Consider the following machine H:

H = "On input ⟨M, w⟩:
    Construct the machine M' = "On input x:
        If x has the form 0^n1^n, accept.
        Otherwise, run M on w.
        If M halts on w, accept."
    Run D on ⟨M'⟩
    If D accepts, accept; if D rejects, reject."

We claim that H is a decider and that L(H) = *HALT*. To see that H is a decider, note that after H constructs M', H runs D on ⟨M'⟩. Since D is a decider, D always halts. If D accepts, H accepts, and if D rejects, H rejects. Thus H halts on all inputs.

To see that L(H) = *HALT*, note that H accepts ⟨M, w⟩ iff D accepts ⟨M'⟩. Since D decides $REGULAR_{TM}$, D accepts ⟨M'⟩ iff L(M') is regular. We claim that L(M') is regular iff M halts on w. To see this, note that if M halts on w, M' accepts all strings, either because the string has form 0^n1^n or because it accepts in the final step after M halts. Thus L(M') = Σ*, which is regular. If M does not halt on w, then M only accepts x if it has the form 0^n1^n.

Theorem: $REGULAR_{TM}$ is undecidable.

Proof: By contradiction; assume D decides $REGULAR_{TM}$. Consider the following machine H:

H = "On input ⟨M, w⟩:
    Construct the machine M' = "On input x:
        If x has the form 0^n1^n, accept.
        Otherwise, run M on w.
        If M halts on w, accept."
    Run D on ⟨M'⟩
    If D accepts, accept; if D rejects, reject."

We claim that H is a decider and that L(H) = *HALT*. To see that H is a decider, note that after H constructs M', H runs D on ⟨M'⟩. Since D is a decider, D always halts. If D accepts, H accepts, and if D rejects, H rejects. Thus H halts on all inputs.

To see that L(H) = *HALT*, note that H accepts ⟨M, w⟩ iff D accepts ⟨M'⟩. Since D decides $REGULAR_{TM}$, D accepts ⟨M'⟩ iff L(M') is regular. We claim that L(M') is regular iff M halts on w. To see this, note that if M halts on w, M' accepts all strings, either because the string has form 0^n1^n or because it accepts in the final step after M halts. Thus L(M') = Σ*, which is regular. If M does not halt on w, then M only accepts x if it has the form 0^n1^n. Thus L(M') = $\{0^n1^n \mid n \in \mathbb{N}\}$, which is not regular.

Theorem: $REGULAR_{TM}$ is undecidable.

Proof: By contradiction; assume D decides $REGULAR_{TM}$. Consider the following machine H:

H = "On input ⟨M, w⟩:
    Construct the machine M' = "On input x:
        If x has the form 0^n1^n, accept.
        Otherwise, run M on w.
        If M halts on w, accept."
    Run D on ⟨M'⟩
    If D accepts, accept; if D rejects, reject."

We claim that H is a decider and that L(H) = *HALT*. To see that H is a decider, note that after H constructs M', H runs D on ⟨M'⟩. Since D is a decider, D always halts. If D accepts, H accepts, and if D rejects, H rejects. Thus H halts on all inputs.

To see that L(H) = *HALT*, note that H accepts ⟨M, w⟩ iff D accepts ⟨M'⟩. Since D decides $REGULAR_{TM}$, D accepts ⟨M'⟩ iff L(M') is regular. We claim that L(M') is regular iff M halts on w. To see this, note that if M halts on w, M' accepts all strings, either because the string has form 0^n1^n or because it accepts in the final step after M halts. Thus L(M') = Σ*, which is regular. If M does not halt on w, then M only accepts x if it has the form 0^n1^n. Thus L(M') = $\{0^n1^n \mid n \in \mathbb{N}\}$, which is not regular. Thus H accepts ⟨M, w⟩ iff M halts on w iff ⟨M, w⟩ ∈ *HALT*, so L(H) = *HALT*.

Theorem: $REGULAR_{TM}$ is undecidable.

Proof: By contradiction; assume D decides $REGULAR_{TM}$. Consider the following machine H:

H = "On input ⟨M, w⟩:
    Construct the machine M' = "On input x:
        If x has the form 0^n1^n, accept.
        Otherwise, run M on w.
        If M halts on w, accept."
    Run D on ⟨M'⟩
    If D accepts, accept; if D rejects, reject."

We claim that H is a decider and that L(H) = *HALT*. To see that H is a decider, note that after H constructs M', H runs D on ⟨M'⟩. Since D is a decider, D always halts. If D accepts, H accepts, and if D rejects, H rejects. Thus H halts on all inputs.

To see that L(H) = *HALT*, note that H accepts ⟨M, w⟩ iff D accepts ⟨M'⟩. Since D decides $REGULAR_{TM}$, D accepts ⟨M'⟩ iff L(M') is regular. We claim that L(M') is regular iff M halts on w. To see this, note that if M halts on w, M' accepts all strings, either because the string has form 0^n1^n or because it accepts in the final step after M halts. Thus L(M') = Σ*, which is regular. If M does not halt on w, then M only accepts x if it has the form 0^n1^n. Thus L(M') = $\{0^n1^n \mid n \in \mathbb{N}\}$, which is not regular. Thus H accepts ⟨M, w⟩ iff M halts on w iff ⟨M, w⟩ ∈ *HALT*, so L(H) = *HALT*.

We have reached a contradiction, because we know that *HALT* is undecidable.

Theorem: $REGULAR_{TM}$ is undecidable.

Proof: By contradiction; assume D decides $REGULAR_{TM}$. Consider the following machine H:

H = "On input ⟨M, w⟩:
    Construct the machine M' = "On input x:
        If x has the form 0^n1^n, accept.
        Otherwise, run M on w.
        If M halts on w, accept."
    Run D on ⟨M'⟩
    If D accepts, accept; if D rejects, reject."

We claim that H is a decider and that L(H) = *HALT*. To see that H is a decider, note that after H constructs M', H runs D on ⟨M'⟩. Since D is a decider, D always halts. If D accepts, H accepts, and if D rejects, H rejects. Thus H halts on all inputs.

To see that L(H) = *HALT*, note that H accepts ⟨M, w⟩ iff D accepts ⟨M'⟩. Since D decides $REGULAR_{TM}$, D accepts ⟨M'⟩ iff L(M') is regular. We claim that L(M') is regular iff M halts on w. To see this, note that if M halts on w, M' accepts all strings, either because the string has form 0^n1^n or because it accepts in the final step after M halts. Thus L(M') = Σ*, which is regular. If M does not halt on w, then M only accepts x if it has the form 0^n1^n. Thus L(M') = $\{0^n1^n \mid n \in \mathbb{N}\}$, which is not regular. Thus H accepts ⟨M, w⟩ iff M halts on w iff ⟨M, w⟩ ∈ *HALT*, so L(H) = *HALT*.

We have reached a contradiction, because we know that *HALT* is undecidable. Thus our assumption was wrong and $REGULAR_{TM}$ is undecidable.

Theorem: $REGULAR_{TM}$ is undecidable.

Proof: By contradiction; assume D decides $REGULAR_{TM}$. Consider the following machine H:

H = "On input ⟨M, w⟩:
    Construct the machine M' = "On input x:
        If x has the form 0^n1^n, accept.
        Otherwise, run M on w.
        If M halts on w, accept."
    Run D on ⟨M'⟩
    If D accepts, accept; if D rejects, reject."

We claim that H is a decider and that L(H) = *HALT*. To see that H is a decider, note that after H constructs M', H runs D on ⟨M'⟩. Since D is a decider, D always halts. If D accepts, H accepts, and if D rejects, H rejects. Thus H halts on all inputs.

To see that L(H) = *HALT*, note that H accepts ⟨M, w⟩ iff D accepts ⟨M'⟩. Since D decides $REGULAR_{TM}$, D accepts ⟨M'⟩ iff L(M') is regular. We claim that L(M') is regular iff M halts on w. To see this, note that if M halts on w, M' accepts all strings, either because the string has form 0^n1^n or because it accepts in the final step after M halts. Thus L(M') = Σ*, which is regular. If M does not halt on w, then M only accepts x if it has the form 0^n1^n. Thus L(M') = $\{0^n1^n \mid n \in \mathbb{N}\}$, which is not regular. Thus H accepts ⟨M, w⟩ iff M halts on w iff ⟨M, w⟩ ∈ *HALT*, so L(H) = *HALT*.

We have reached a contradiction, because we know that *HALT* is undecidable. Thus our assumption was wrong and $REGULAR_{TM}$ is undecidable. ■

The Story So Far

- Consider the following problems:
  - Does M not accept ⟨M⟩?
  - Does M accept w?
  - Does M halt on all inputs?
  - Is L(M) regular?

The Story So Far

- Consider the following problems:
  - Does M not accept ⟨M⟩? **Undecidable**
  - Does M accept w?
  - Does M halt on all inputs?
  - Is L(M) regular?

The Story So Far

- Consider the following problems:
  - Does M not accept ⟨M⟩? **Undecidable**
  - Does M accept w? **Undecidable**
  - Does M halt on all inputs?
  - Is L(M) regular?

The Story So Far

- Consider the following problems:
  - Does M not accept ⟨M⟩? **Undecidable**
  - Does M accept w? **Undecidable**
  - Does M halt on all inputs? **Undecidable**
  - Is L(M) regular?

The Story So Far

- Consider the following problems:
  - Does M not accept ⟨M⟩? **Undecidable**
  - Does M accept w? **Undecidable**
  - Does M halt on all inputs? **Undecidable**
  - Is L(M) regular? **Undecidable**

The Story So Far

- Consider the following problems:
  - Does M not accept ⟨M⟩? **Undecidable**
  - Does M accept w? **Undecidable**
  - Does M halt on all inputs? **Undecidable**
  - Is L(M) regular? **Undecidable**
- There seems to be a trend here.
- What properties of Turing machines **are** decidable?

Rice's Theorem

Properties of RE Languages

- A **property of an RE language** is some trait that may apply to RE languages.
- For example:
  - Does L = Ø?
  - Is L regular?
  - Is L context-free?
  - Does L contain any string of length exactly 137?

Properties of RE Languages

- We can describe a property of an RE language as the set of RE languages with that property.
- If P is a property of RE languages, consider the language

$$L_P = \{ \langle M \rangle \mid M \text{ is a TM and } L(M) \in P \}$$

(The set of TMs that describe a language with property P.)

- Note that membership in L_P depends only on the **language** of a TM, not the **description** of that TM.

If $L(M_1) = L(M_2)$, then $\langle M_1 \rangle \in L_P$ iff $\langle M_2 \rangle \in L_P$

Properties of RE Languages

- The set

$$L_{even} = \{ \langle M \rangle \mid L(M) \text{ is finite, and } |L(M)| \text{ is even. } \}$$

  is a property of RE languages, because it depends **purely** on the language of the TM and not on the TM itself.
  - Specifically: If $L(M_1) = L(M_2)$, then $\langle M_1 \rangle \in L_{even}$ iff $\langle M_2 \rangle \in L_{even}$.

Properties of RE Languages

- The set

$$L_{even} = \{ \langle M \rangle \mid L(M) \text{ is finite, and } |L(M)| \text{ is even. } \}$$

  is a property of RE languages, because it depends **purely** on the language of the TM and not on the TM itself.
  - Specifically: If $L(M_1) = L(M_2)$, then $\langle M_1 \rangle \in L_{even}$ iff $\langle M_2 \rangle \in L_{even}$.
- The set

$$L_{evenQ} = \{ \langle M \rangle \mid M \text{ has an even number of states } \}$$

  is **not** a property of RE languages, because it does not depend purely on the language of the TM.
  - Specifically: If $L(M_1) = L(M_2)$, then it may be possible for $\langle M_1 \rangle \in L_{evenQ}$ but $\langle M_2 \rangle \notin L_{evenQ}$.

Trivial and Nontrivial Properties

- A property of RE languages is called **trivial** if all RE languages have the property or no RE languages have the property.
  - { ⟨M⟩ | L(M) is RE } is trivial.
  - { ⟨M⟩ | L(M) is not RE } is trivial.
- A property of RE languages is called **nontrivial** if there exists TMs M_1 and M_2 such that $\langle M_1 \rangle \in L_P$ but $\langle M_2 \rangle \notin L_2$.
  - { ⟨M⟩ | L(M) is infinite } is nontrivial.
  - { ⟨M⟩ | L(M) is regular } is nontrivial.
  - { ⟨M⟩ | L(M) is recursive } is nontrivial.

Rice's Theorem

Any nontrivial property of the RE languages is **undecidable**.

Using Rice's Theorem

- Can we apply Rice's theorem to this language?

$$L_{ne} = \{ \langle M \rangle \mid L(M) \neq \emptyset \}$$

Using Rice's Theorem

- Can we apply Rice's theorem to this language?

$$L_{ne} = \{ \langle M \rangle \mid L(M) \neq \emptyset \}$$

- We can apply Rice's theorem if two conditions hold:

Using Rice's Theorem

- Can we apply Rice's theorem to this language?

$$L_{ne} = \{ \langle M \rangle \mid L(M) \neq \emptyset \}$$

- We can apply Rice's theorem if two conditions hold:
- **L_{ne} is nontrivial**:

$$\exists M_1.\ \exists M_2.\ (\langle M_1 \rangle \in L_{ne} \wedge \langle M_2 \rangle \notin L_{ne})$$

Using Rice's Theorem

- Can we apply Rice's theorem to this language?

$$L_{ne} = \{ \langle M \rangle \mid L(M) \neq \emptyset \}$$

- We can apply Rice's theorem if two conditions hold:
- **L_{ne} is nontrivial**:

$$\exists M_1.\ \exists M_2.\ (\langle M_1 \rangle \in L_{ne} \wedge \langle M_2 \rangle \notin L_{ne})$$

Using Rice's Theorem

- Can we apply Rice's theorem to this language?

$$L_{ne} = \{ \langle M \rangle \mid L(M) \neq \emptyset \}$$

- We can apply Rice's theorem if two conditions hold:
- **L_{ne} is nontrivial**:

$$\exists M_1.\ \exists M_2.\ (\langle M_1 \rangle \in L_{ne} \wedge \langle M_2 \rangle \notin L_{ne})$$

- **L_{ne} is a property of RE languages**:

$$\text{If } L(M_1) = L(M_2), \text{ then } \langle M_1 \rangle \in L_{ne} \text{ iff } \langle M_2 \rangle \in L_{ne}$$

Using Rice's Theorem

- Can we apply Rice's theorem to this language?

$$L_{ne} = \{ \langle M \rangle \mid L(M) \neq \emptyset \}$$

- We can apply Rice's theorem if two conditions hold:
- **L_{ne} is nontrivial**:

$$\exists M_1.\ \exists M_2.\ (\langle M_1 \rangle \in L_{ne} \wedge \langle M_2 \rangle \notin L_{ne})$$

- **L_{ne} is a property of RE languages**:

$$\text{If } L(M_1) = L(M_2), \text{ then } \langle M_1 \rangle \in L_{ne} \text{ iff } \langle M_2 \rangle \in L_{ne}$$

Using Rice's Theorem

- Can we apply Rice's theorem to this language?

$$L_{ne} = \{ \langle M \rangle \mid L(M) \neq \emptyset \}$$

- We can apply Rice's theorem if two conditions hold:
- **L_{ne} is nontrivial**:

$$\exists M_1.\ \exists M_2.\ (\langle M_1 \rangle \in L_{ne} \wedge \langle M_2 \rangle \notin L_{ne})$$

- **L_{ne} is a property of RE languages**:

$$\text{If } L(M_1) = L(M_2), \text{ then } \langle M_1 \rangle \in L_{ne} \text{ iff } \langle M_2 \rangle \in L_{ne}$$

- **Rice's theorem applies; L_{ne} is undecidable.**

Using Rice's Theorem

- Can we apply Rice's theorem to this language?

  L_{es} = { ⟨M⟩ | M has an even number of states }

Using Rice's Theorem

- Can we apply Rice's theorem to this language?

  L_{es} = { ⟨M⟩ | M has an even number of states }

- We can apply Rice's theorem if two conditions hold:

Using Rice's Theorem

- Can we apply Rice's theorem to this language?

$$L_{es} = \{ \langle M \rangle \mid M \text{ has an even number of states} \}$$

- We can apply Rice's theorem if two conditions hold:
- **L_{es} is nontrivial**:

$$\exists M_1.\ \exists M_2.\ (\langle M_1 \rangle \in L_{es} \wedge \langle M_2 \rangle \notin L_{es})$$

Using Rice's Theorem

- Can we apply Rice's theorem to this language?

$$L_{es} = \{ \langle M \rangle \mid M \text{ has an even number of states} \}$$

- We can apply Rice's theorem if two conditions hold:
- **L_{es} is nontrivial**:

$$\exists M_1.\ \exists M_2.\ (\langle M_1 \rangle \in L_{es} \wedge \langle M_2 \rangle \notin L_{es})$$

Using Rice's Theorem

- Can we apply Rice's theorem to this language?

  L_{es} = { ⟨M⟩ | M has an even number of states }

- We can apply Rice's theorem if two conditions hold:
- **L_{es} is nontrivial**:

  $$\exists M_1.\ \exists M_2.\ (\langle M_1 \rangle \in L_{es} \wedge \langle M_2 \rangle \notin L_{es})$$

- **L_{es} is a property of RE languages**:

  If $L(M_1) = L(M_2)$, then $\langle M_1 \rangle \in L_{es}$ iff $\langle M_2 \rangle \in L_{es}$

Using Rice's Theorem

- Can we apply Rice's theorem to this language?

  L_{es} = { ⟨M⟩ | M has an even number of states }

- We can apply Rice's theorem if two conditions hold:
- **L_{es} is nontrivial**:

  $$\exists M_1.\ \exists M_2.\ (\langle M_1\rangle \in L_{es} \wedge \langle M_2\rangle \notin L_{es})$$

- **L_{es} is a property of RE languages**:

  If $L(M_1) = L(M_2)$, then $\langle M_1\rangle \in L_{es}$ iff $\langle M_2\rangle \in L_{es}$

Using Rice's Theorem

- Can we apply Rice's theorem to this language?

  L_{es} = { ⟨M⟩ | M has an even number of states }

- We can apply Rice's theorem if two conditions hold:
- **L_{es} is nontrivial**:

  $\exists M_1.\ \exists M_2.\ (\langle M_1 \rangle \in L_{es} \wedge \langle M_2 \rangle \notin L_{es})$

- **L_{es} is a property of RE languages**:

  If $L(M_1) = L(M_2)$, then $\langle M_1 \rangle \in L_{es}$ iff $\langle M_2 \rangle \in L_{es}$

- **Rice's theorem does not apply.**

Using Rice's Theorem

- Can we apply Rice's theorem to this language?

  L_{small} = { ⟨M⟩ | There is a five-state TM that accepts L(M) }

Using Rice's Theorem

- Can we apply Rice's theorem to this language?

  L_{small} = { ⟨M⟩ | There is a five-state TM that accepts L(M) }

- We can apply Rice's theorem if two conditions hold:
- **L_{small} is nontrivial**:

$$\exists M_1.\ \exists M_2.\ (\langle M_1 \rangle \in L_{small} \wedge \langle M_2 \rangle \notin L_{small})$$

- **L_{small} is a property of RE languages**:

$$\text{If } L(M_1) = L(M_2), \text{ then } \langle M_1 \rangle \in L_{small} \text{ iff } \langle M_2 \rangle \in L_{small}$$

Using Rice's Theorem

- Can we apply Rice's theorem to this language?

  L_{small} = { ⟨M⟩ | There is a five-state TM that accepts L(M) }

- We can apply Rice's theorem if two conditions hold:
- **L_{small} is nontrivial**:

$$\exists M_1.\ \exists M_2.\ (\langle M_1 \rangle \in L_{small} \wedge \langle M_2 \rangle \notin L_{small})$$

- **L_{small} is a property of RE languages**:

If $L(M_1) = L(M_2)$, then $\langle M_1 \rangle \in L_{small}$ iff $\langle M_2 \rangle \in L_{small}$

Using Rice's Theorem

- Can we apply Rice's theorem to this language?

  L_{small} = { ⟨M⟩ | There is a five-state TM that accepts L(M) }

- We can apply Rice's theorem if two conditions hold:
- **L_{small} is nontrivial**:

$$\exists M_1.\ \exists M_2.\ (\langle M_1 \rangle \in L_{small} \wedge \langle M_2 \rangle \notin L_{small})$$

- **L_{small} is a property of RE languages**:

$$\text{If } L(M_1) = L(M_2), \text{ then } \langle M_1 \rangle \in L_{small} \text{ iff } \langle M_2 \rangle \in L_{small}$$

Using Rice's Theorem

- Can we apply Rice's theorem to this language?

  L_{small} = { ⟨M⟩ | There is a five-state TM that accepts L(M) }

- We can apply Rice's theorem if two conditions hold:
- **L_{small} is nontrivial**:

  $$\exists M_1.\ \exists M_2.\ (\langle M_1 \rangle \in L_{small} \wedge \langle M_2 \rangle \notin L_{small})$$

- **L_{small} is a property of RE languages**:

  If $L(M_1) = L(M_2)$, then $\langle M_1 \rangle \in L_{small}$ iff $\langle M_2 \rangle \in L_{small}$

- **Rice's theorem applies; L_{small} is undecidable.**

Using Rice's Theorem

- All of the following problems are undecidable:
  - $L_{palindrome}$ = { ⟨M⟩ | every string in L(M) is a palindrome }
  - L_{allodd} = { ⟨M⟩ | every string in L(M) has odd length }
  - L_{CFL} = { ⟨M⟩ | L(M) is a context-free language }
  - L_{short} = { ⟨M⟩ | L(M) has no strings of length above 5 }
  - $L_{recursive}$ = { ⟨M⟩ | L(M) is recursive }
  - L_{empty} = { ⟨M⟩ | L(M) = Ø }

Proving Rice's Theorem

- The proof of Rice's theorem is generalization of the reductions we've seen so far.
- **General Idea**: If L_P is a nontrivial property of RE languages, show that we can reduce *HALT* to L_P.
- There are a few tricky details in the proof, so we'll take it slowly.

The General Proof Sketch

- Suppose that some nontrivial property of RE languages L_P is decidable.
- Construct a TM H that works as follows:

  On input ⟨M, w⟩:

  Construct TM M' such that ⟨M'⟩ ∈ L_P iff M halts on w.

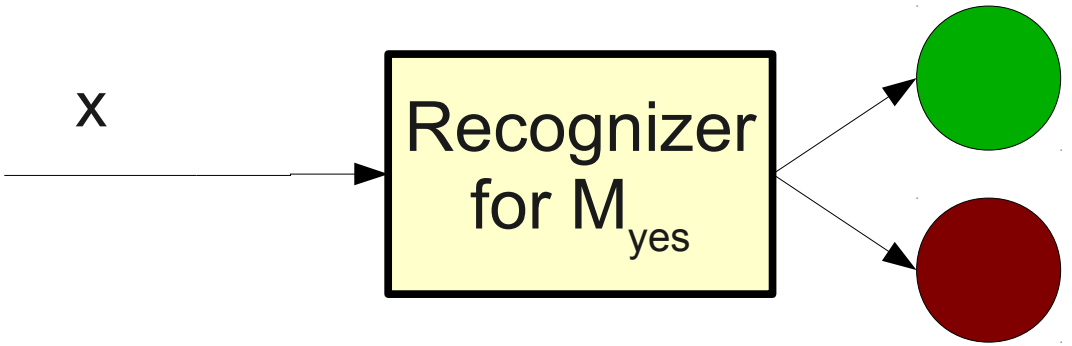
  Use the decider for L_P to decide whether ⟨M'⟩ ∈ L_P
- Conclude that H decides *HALT*, which is impossible.

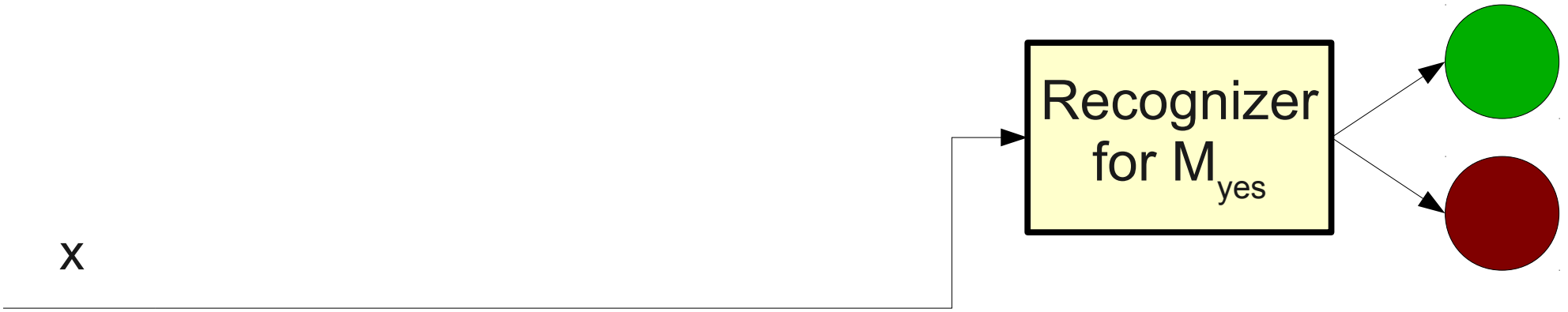
Key Idea One

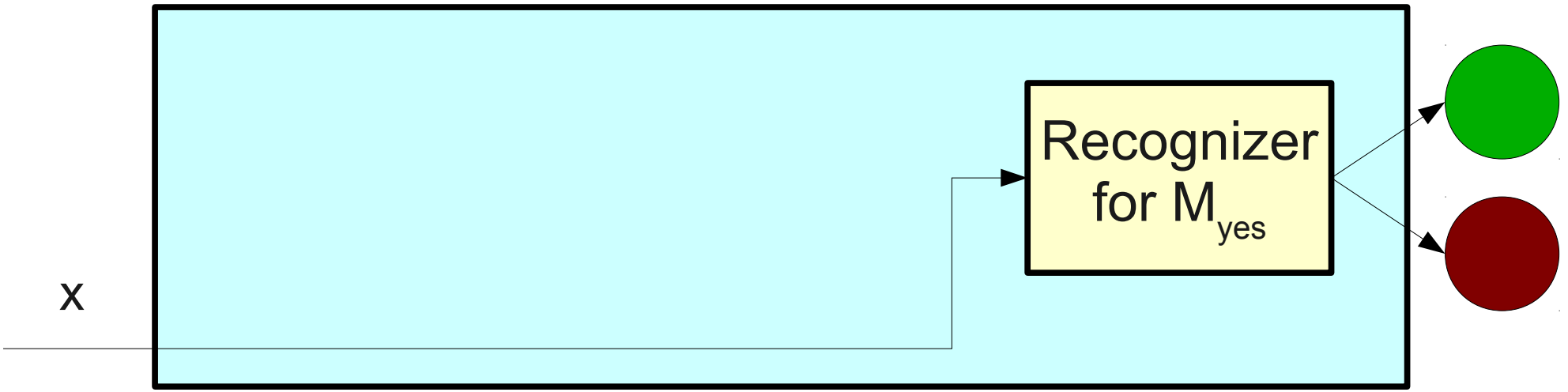
- Assume, for the sake of contradiction, that L_P is a decidable nontrivial property of RE languages.
- Since R is closed under complement, if we can show that either L_P is undecidable or that $\overline{L}_P$ is undecidable, we will have reached a contradiction.
- If $M_\varnothing$ is a TM that accepts the empty language, then either $\langle M_\varnothing \rangle \in L_P$ or $\langle M_\varnothing \rangle \notin L_P$.
  - If $\langle M_\varnothing \rangle \notin L_P$, we prove that L_P is undecidable.
  - If $\langle M_\varnothing \rangle \in L_P$, we prove that $\overline{L}_P$ is undecidable.

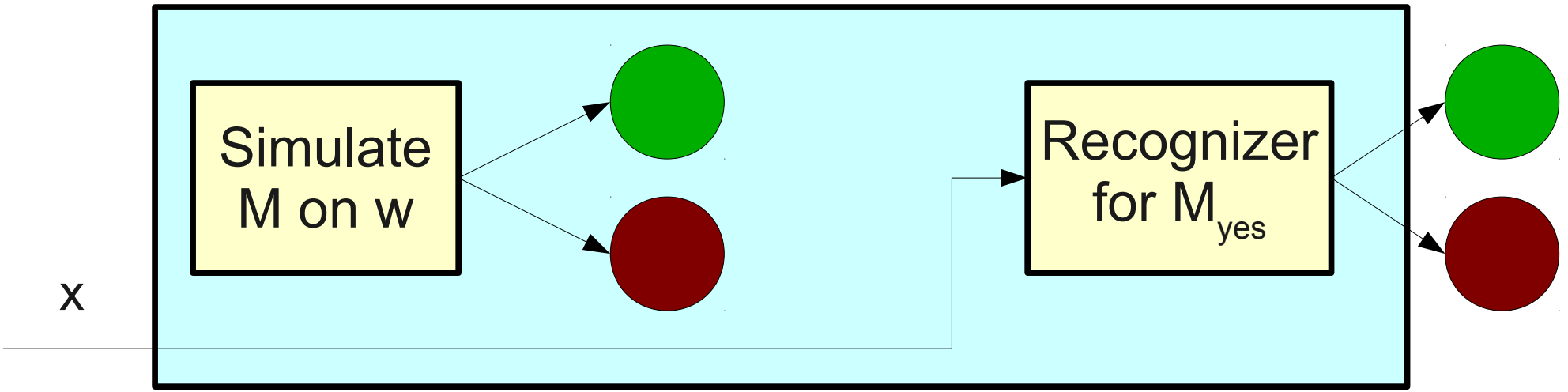
Key Idea Two

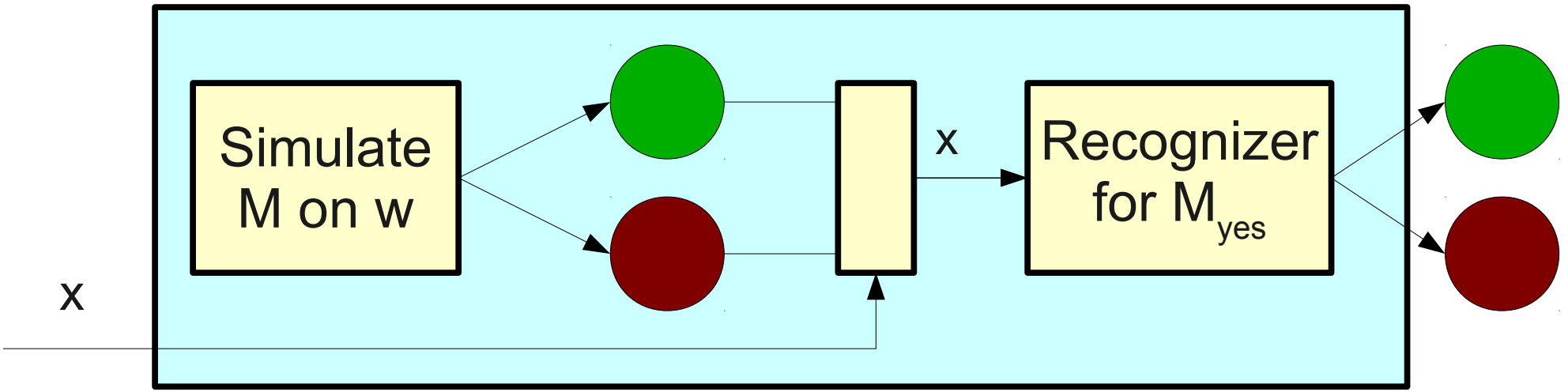
- Assume without loss of generality that $\langle M_\varnothing \rangle \notin L_P$.
- L_P is a *nontrivial property of RE languages*, so there must be some TM M_{yes} for which $\langle M_{yes} \rangle \in L_P$.
- L_P is a *nontrivial property of RE languages*, so $L(M_{yes}) \neq L(M_\varnothing)$.

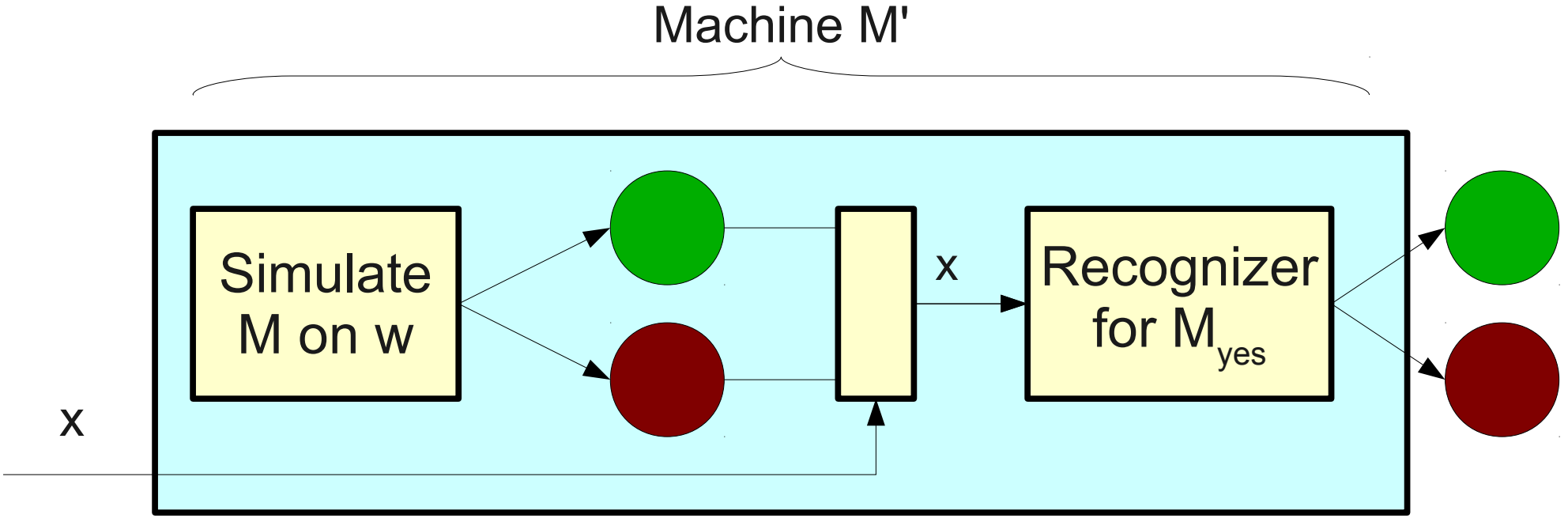

x
Recognizer
for M
yes

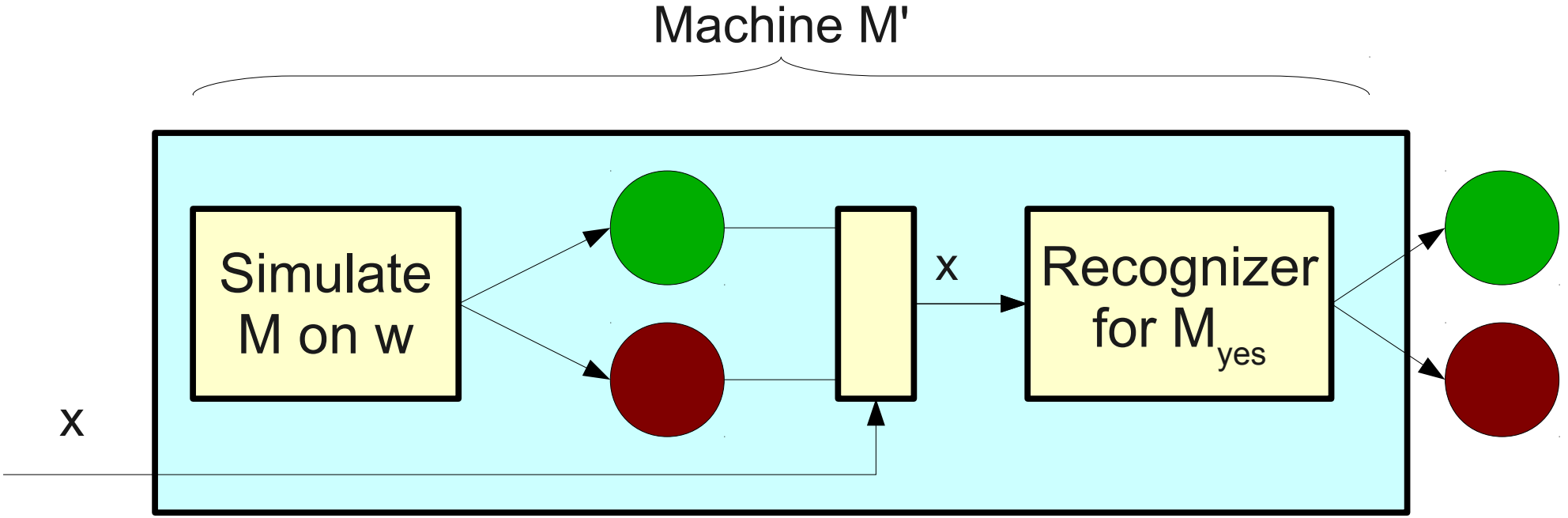

Recognizer
for M_{yes}
x


Recognizer
for M yes
x


Simulate
M on w
Recognizer
for M yes
x


Simulate
M on w
x
Recognizer
for M yes
x


Machine M'
Simulate
M on w
x
Recognizer
for M yes
x



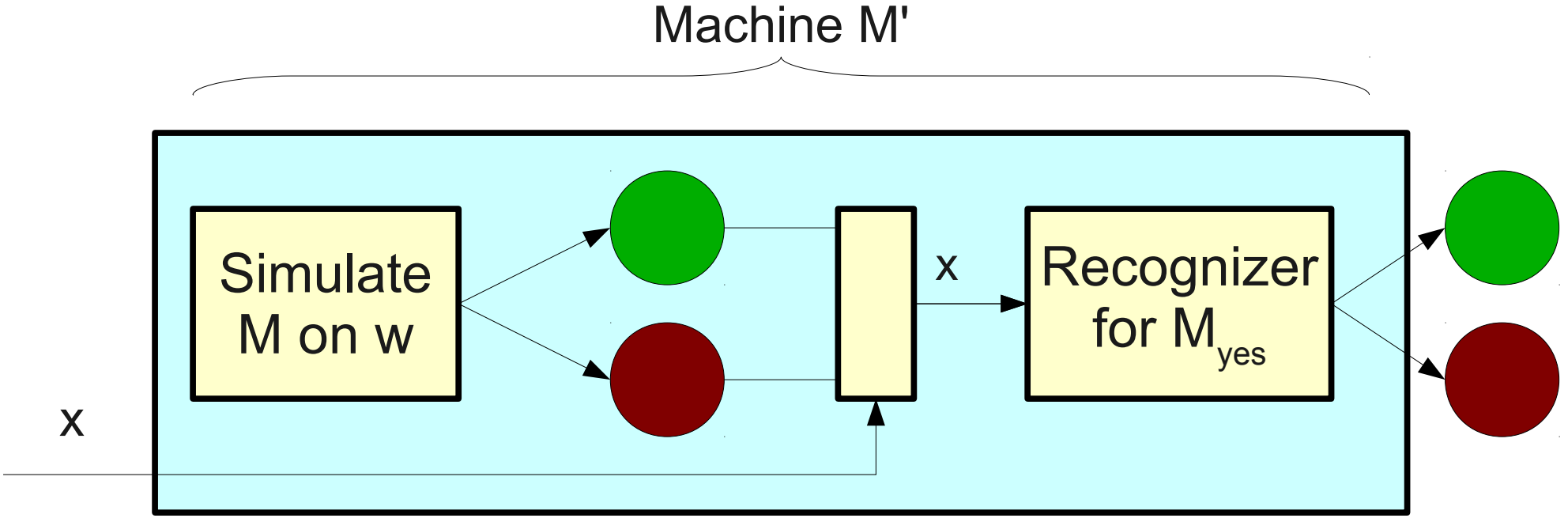
M' = “On input x:

Run M on w.

If M halts, run M_{yes} on x.

Accept if M_{yes} accepts x.

Reject if M_{yes} rejects x.”



M' = “On input x:

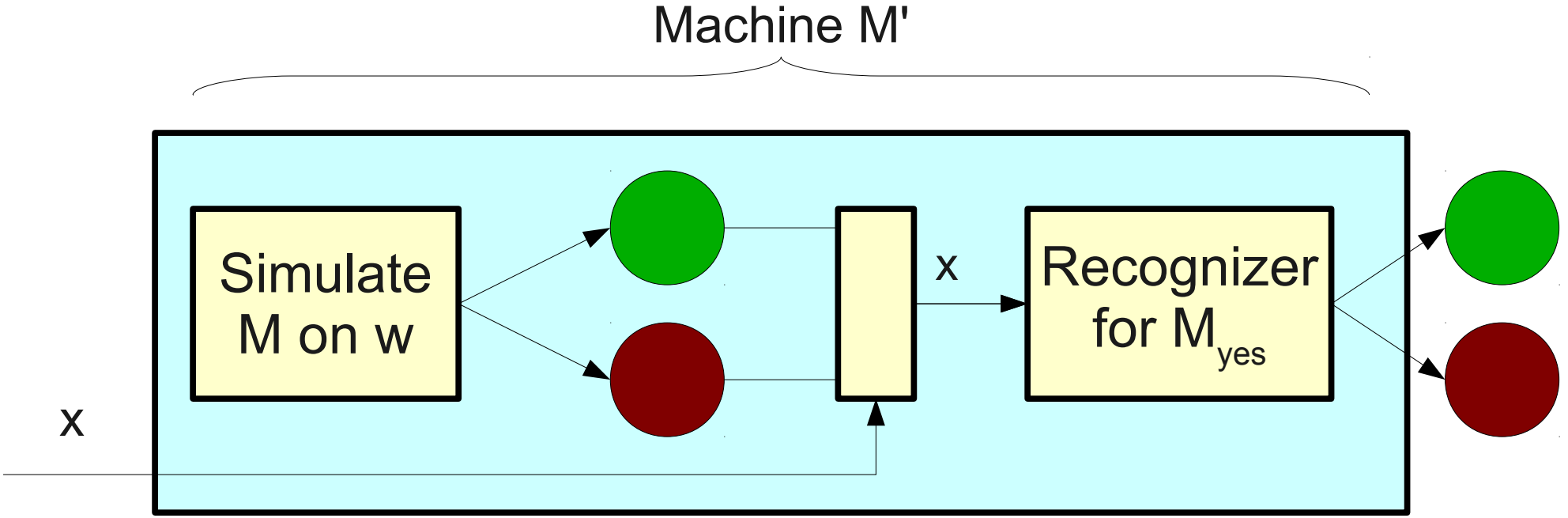
Run M on w.

If M halts, run M_{yes} on x.

Accept if M_{yes} accepts x.

Reject if M_{yes} rejects x.”

If M loops on w, what does M' do on x?



M' = “On input x:

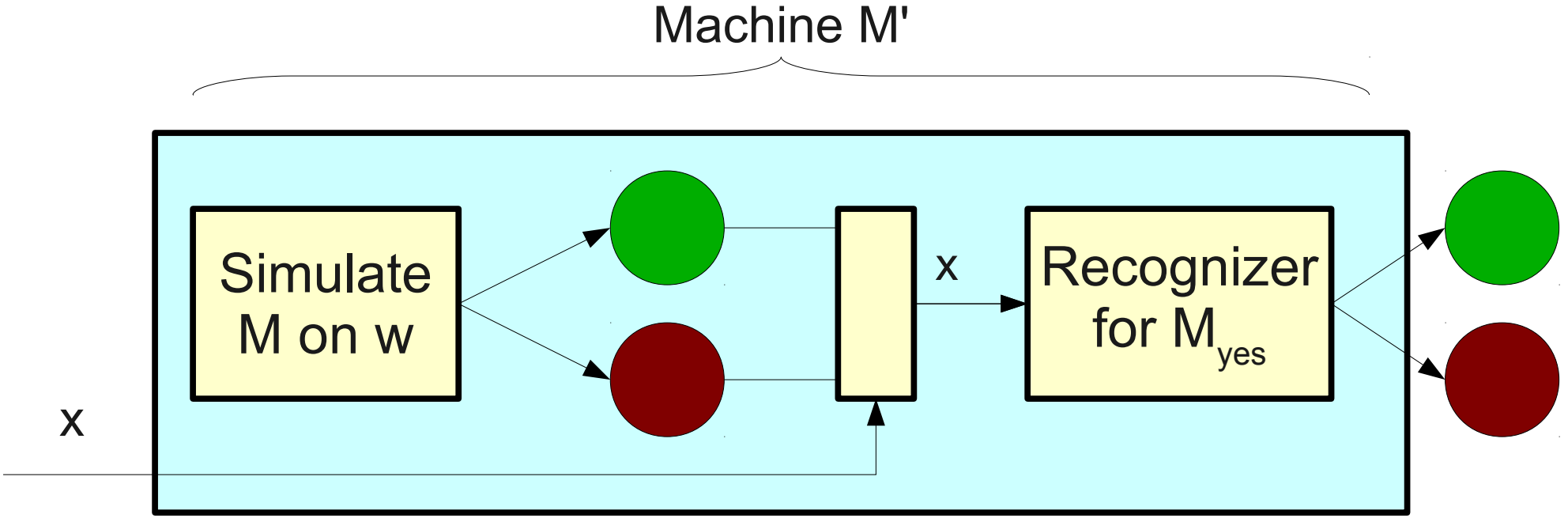
Run M on w.

If M halts, run M_{yes} on x.

Accept if M_{yes} accepts x.

Reject if M_{yes} rejects x.”

If M loops on w, what does M' do on x?



M' = "On input x:

Run M on w.

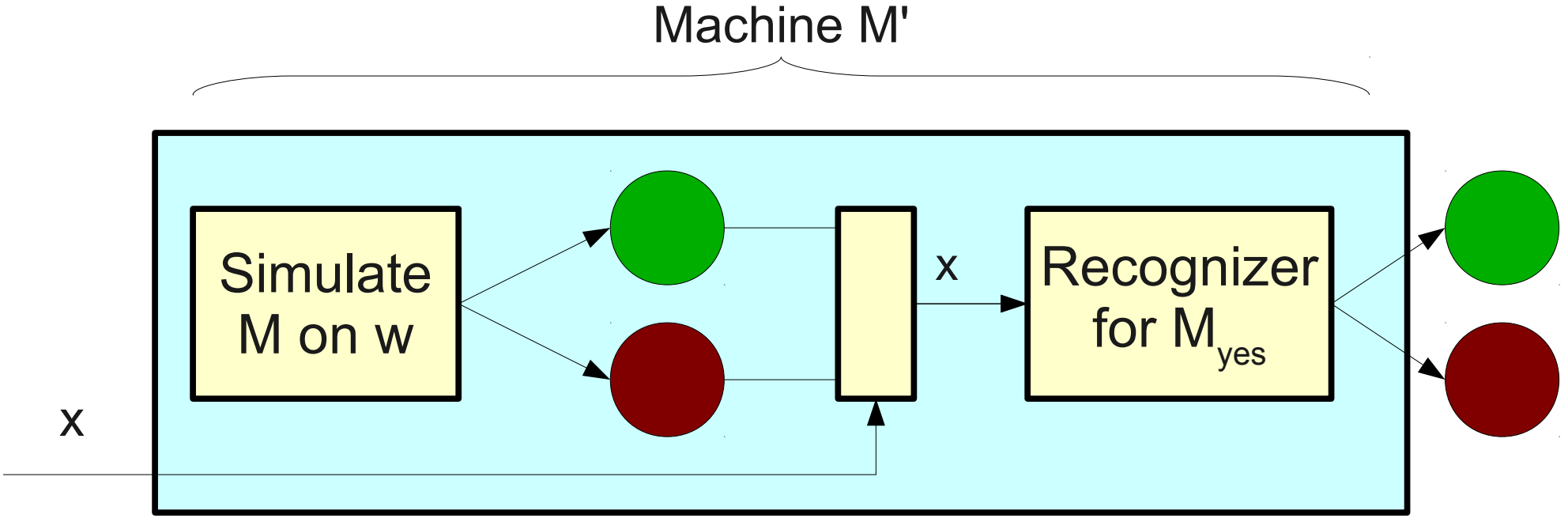
If M halts, run M_{yes} on x.

Accept if M_{yes} accepts x.

Reject if M_{yes} rejects x."

If M loops on w, what does M' do on x?

M' always loops.



M' = "On input x:

Run M on w.

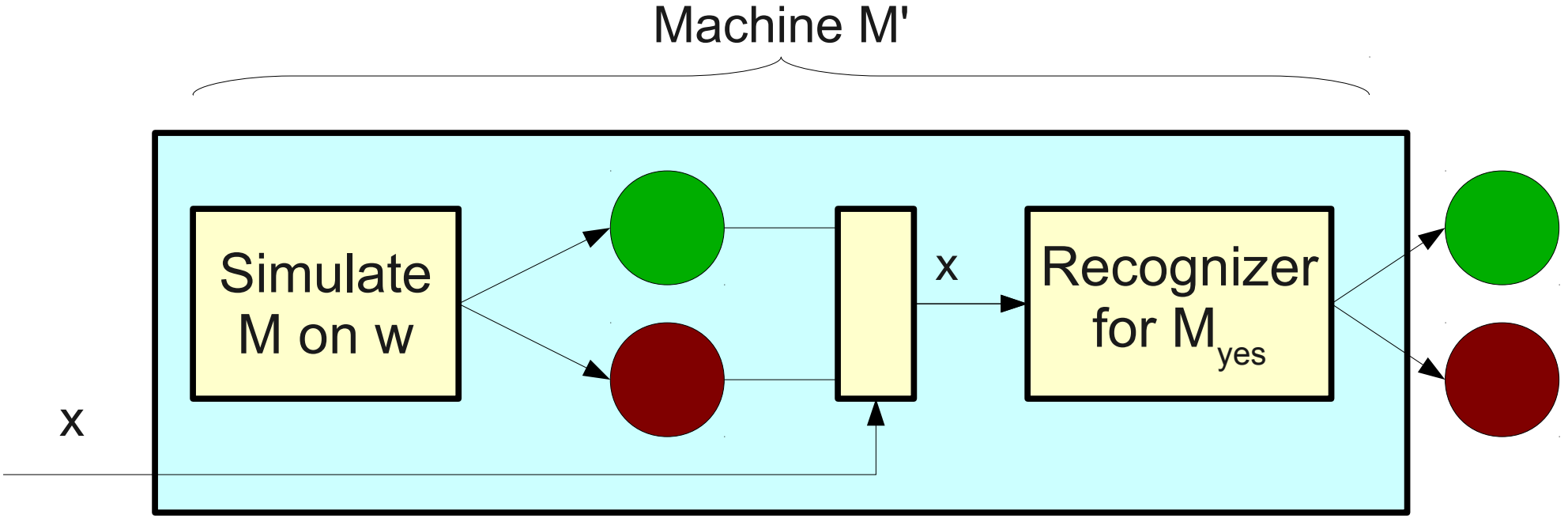
If M halts, run M_{yes} on x.

Accept if M_{yes} accepts x.

Reject if M_{yes} rejects x."

If M loops on w, what does M' do on x?

$L(M') = \emptyset$



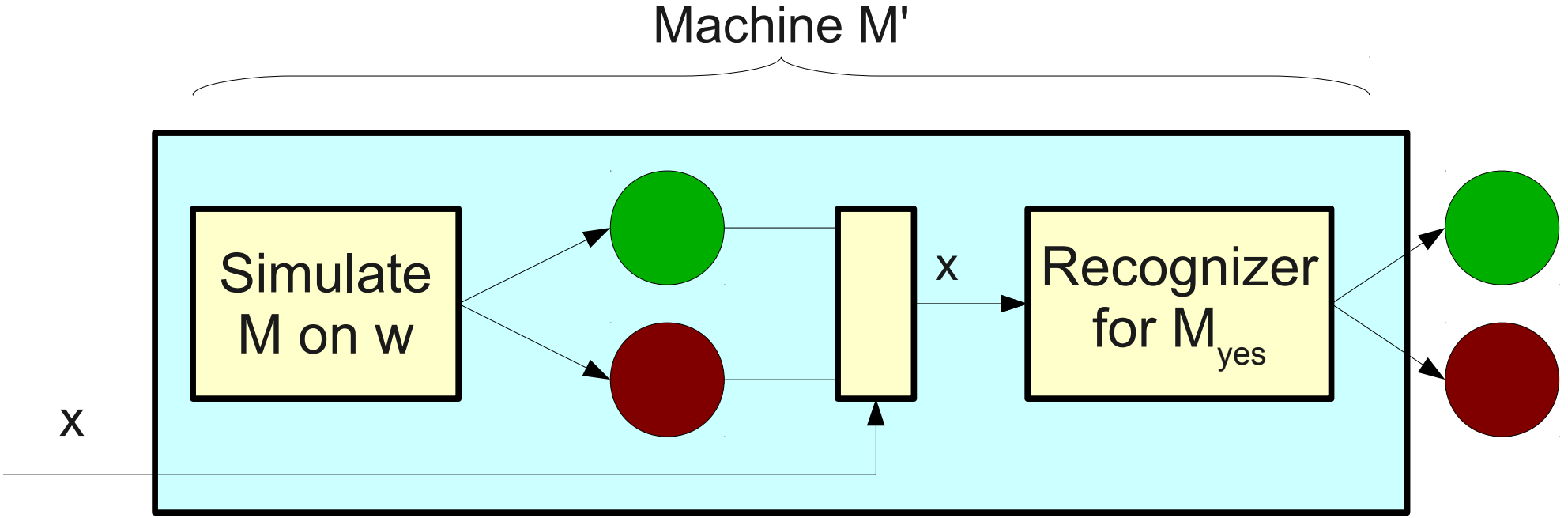
M' = “On input x:

Run M on w.

If M halts, run M_{yes} on x.

Accept if M_{yes} accepts x.

Reject if M_{yes} rejects x.”



M' = “On input x:

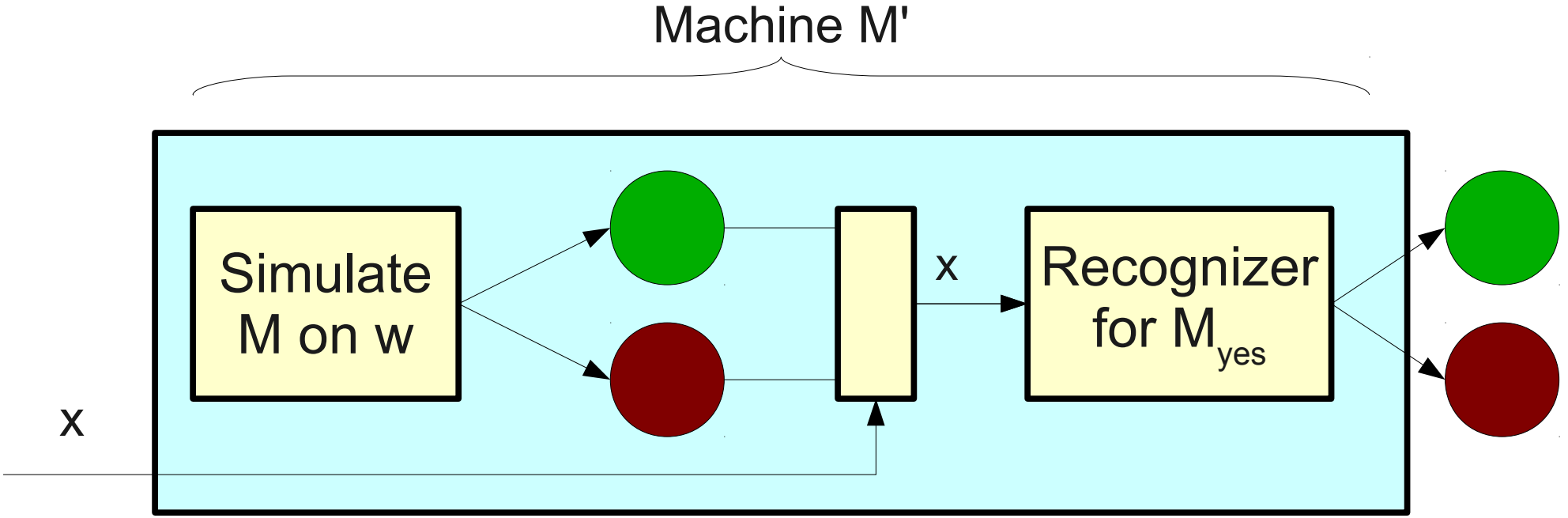
Run M on w.

If M halts, run M_{yes} on x.

Accept if M_{yes} accepts x.

Reject if M_{yes} rejects x.”

If M halts on w, what does M' do on x?



M' = "On input x:

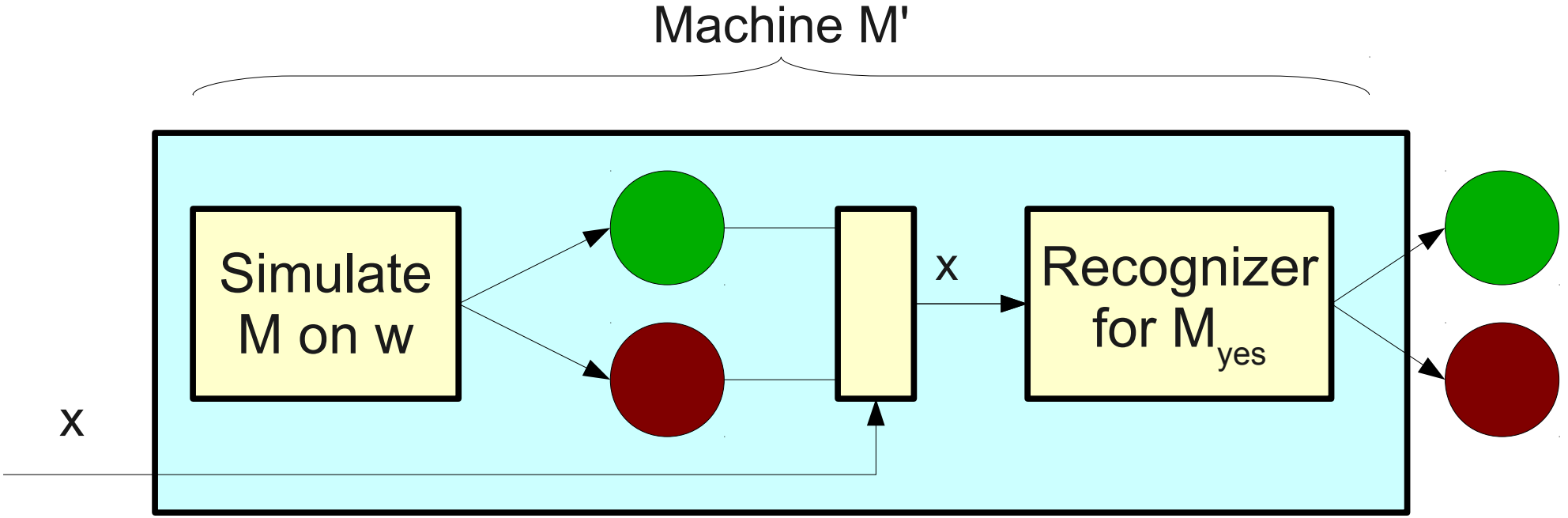
Run M on w.

If M halts, run M_{yes} on x.

Accept if M_{yes} accepts x.

Reject if M_{yes} rejects x."

If M halts on w, what does M' do on x?



M' = "On input x:

Run M on w.

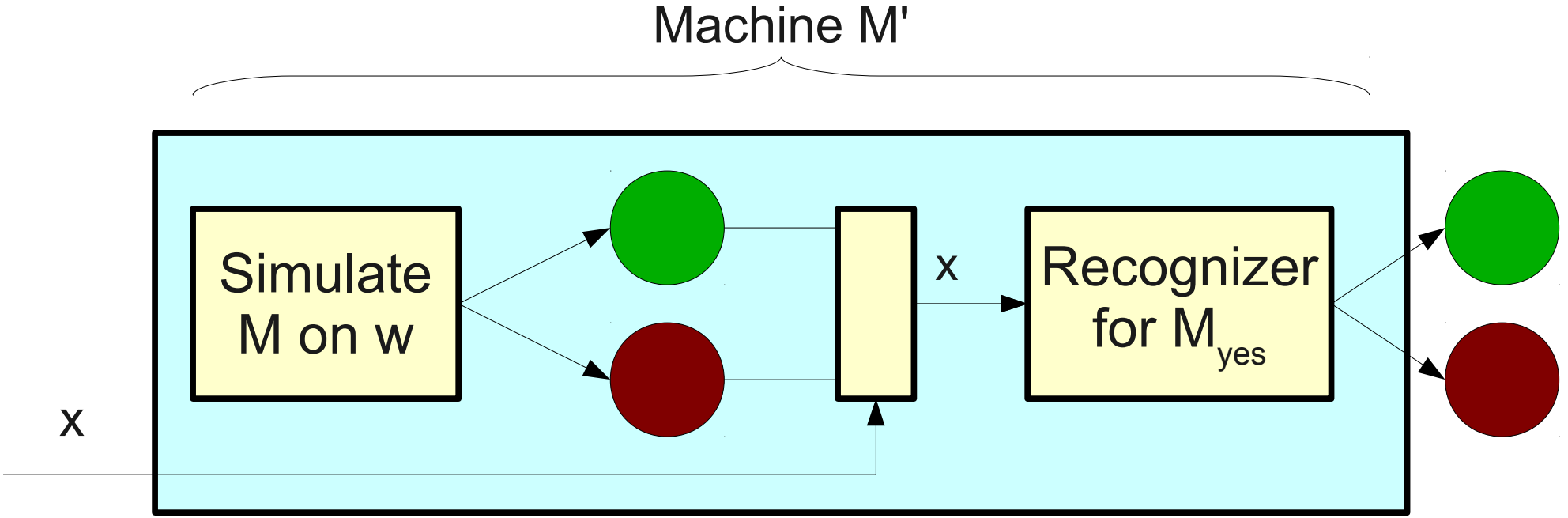
If M halts, run M_{yes} on x.

Accept if M_{yes} accepts x.

Reject if M_{yes} rejects x."

If M halts on w, what does M' do on x?

M' accepts x iff M_{yes} accepts x



M' = "On input x:

Run M on w.

If M halts, run M_{yes} on x.

Accept if M_{yes} accepts x.

Reject if M_{yes} rejects x."

If M halts on w, what does M' do on x?

$L(M') = L(M_{yes})$

Key Idea Three

M' = “On input x:

Run M on w.

If M halts, run M_{yes} on x.

Accept if M_{yes} accepts x.

Reject if M_{yes} rejects x.”

If M halts on w, $L(M') = L(M_{yes})$

If M loops on w, $L(M') = \varnothing$

Key Idea Three

M' = “On input x:

Run M on w.

If M halts, run M_{yes} on x.

Accept if M_{yes} accepts x.

Reject if M_{yes} rejects x.”

If M halts on w, $L(M') = L(M_{yes})$

If M loops on w, **$L(M') = \emptyset$**

Key Idea Three

M' = “On input x:

Run M on w.

If M halts, run M_{yes} on x.

Accept if M_{yes} accepts x.

Reject if M_{yes} rejects x.”

If M halts on w, $L(M') = L(M_{yes})$

If M loops on w, **$L(M') = L(M_{\emptyset})$**

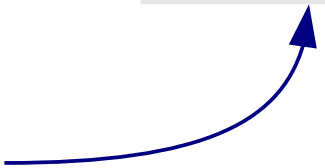
Key Idea Three

M' = “On input x:
Run M on w.
If M halts, run M_{yes} on x.
Accept if M_{yes} accepts x.
Reject if M_{yes} rejects x.”

If M halts on w, $L(M') = L(M_{yes})$

If M loops on w, **$L(M') = L(M_{\emptyset})$**

Remember that $M_{\emptyset}$ is a machine that never accepts, so $L(M_{\emptyset}) = \emptyset$



Key Idea Three

M' = “On input x:

Run M on w.

If M halts, run M_{yes} on x.

Accept if M_{yes} accepts x.

Reject if M_{yes} rejects x.”

If M halts on w, $L(M') = L(M_{yes})$

If M loops on w, $L(M') = L(M_{\varnothing})$

Key Idea Three

M' = “On input x:

Run M on w.

If M halts, run M_{yes} on x.

Accept if M_{yes} accepts x.

Reject if M_{yes} rejects x.”

If M halts on w, **$L(M') = L(M_{yes})$**

If M loops on w, $L(M') = L(M_{\varnothing})$

Key Idea Three

M' = “On input x:

Run M on w.

If M halts, run M_{yes} on x.

Accept if M_{yes} accepts x.

Reject if M_{yes} rejects x.”

If M halts on w, $\langle M' \rangle \in L_P$

If M loops on w, $L(M') = L(M_\varnothing)$

Key Idea Three

M' = "On input x:

Run M on w.

If M halts, run M_{yes} on x.

Accept if M_{yes} accepts x.

Reject if M_{yes} rejects x."

Since we know that $L(M') = L(M_{yes})$ and we know that $\langle M_{yes} \rangle \in L_P$ we know that $\langle M' \rangle \in L_P$.

If M halts on w, $\langle M' \rangle \in L_P$

If M loops on w, $L(M') = L(M_\varnothing)$

Key Idea Three

M' = “On input x:

Run M on w.

If M halts, run M_{yes} on x.

Accept if M_{yes} accepts x.

Reject if M_{yes} rejects x.”

If M halts on w, $\langle M' \rangle \in L_P$

If M loops on w, $L(M') = L(M_\varnothing)$

Key Idea Three

M' = “On input x:

Run M on w.

If M halts, run M_{yes} on x.

Accept if M_{yes} accepts x.

Reject if M_{yes} rejects x.”

If M halts on w, $\langle M' \rangle \in L_P$

If M loops on w, **$L(M') = L(M_{\emptyset})$**

Key Idea Three

M' = “On input x:

Run M on w.

If M halts, run M_{yes} on x.

Accept if M_{yes} accepts x.

Reject if M_{yes} rejects x.”

If M halts on w, $\langle M' \rangle \in L_P$

If M loops on w, $\langle M' \rangle \notin L_P$

Key Idea Three

M' = "On input x:

Run M on w.

If M halts, run M_{yes} on x.

Accept if M_{yes} accepts x.

Reject if M_{yes} rejects x."

Since we know that

$L(M') = L(M_\emptyset)$

and we know that

$\langle M_\emptyset \rangle \notin L_P$

we know that

$\langle M' \rangle \notin L_P$.

If M halts on w, $\langle M' \rangle \in L_P$

If M loops on w, **$\langle M' \rangle \notin L_P$**

Key Idea Three

M' = “On input x:

Run M on w.

If M halts, run M_{yes} on x.

Accept if M_{yes} accepts x.

Reject if M_{yes} rejects x.”

If M halts on w, $\langle M' \rangle \in L_P$

If M loops on w, $\langle M' \rangle \notin L_P$

Key Idea Three

M' = "On input x:

Run M on w.

If M halts, run M_{yes} on x.

Accept if M_{yes} accepts x.

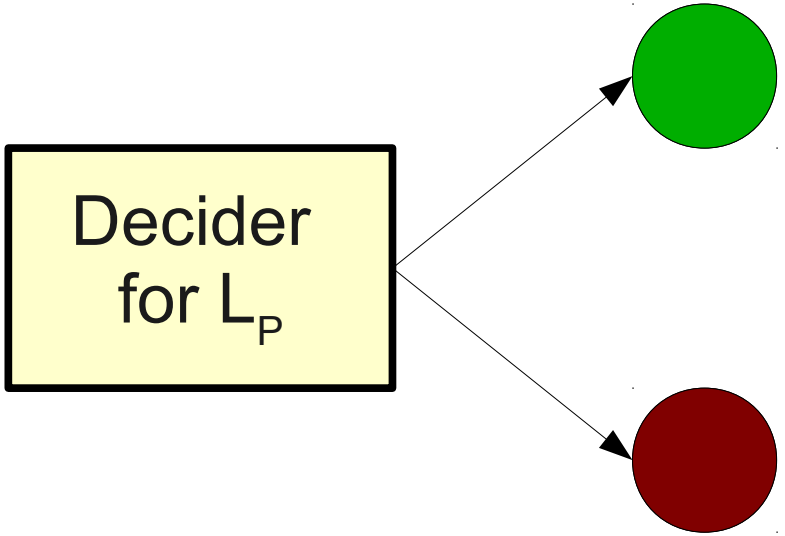
Reject if M_{yes} rejects x."

This is the key step of the construction. If we can decide whether $\langle M' \rangle \in L_P$, we would be deciding whether M halts on w.

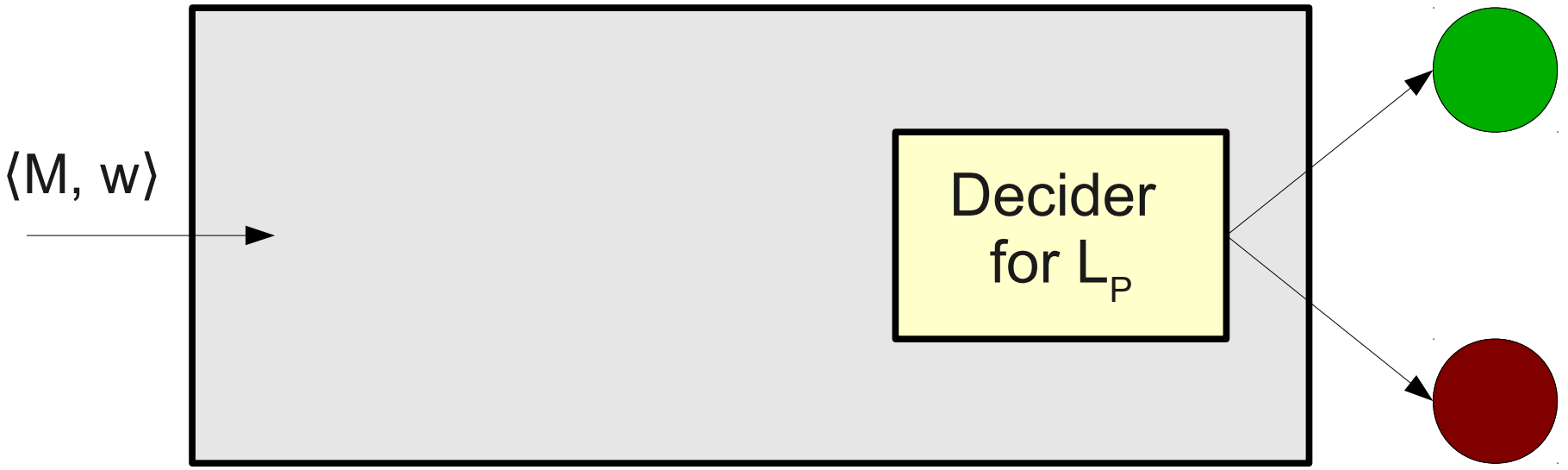
If M halts on w, $\langle M' \rangle \in L_P$

If M loops on w, $\langle M' \rangle \notin L_P$

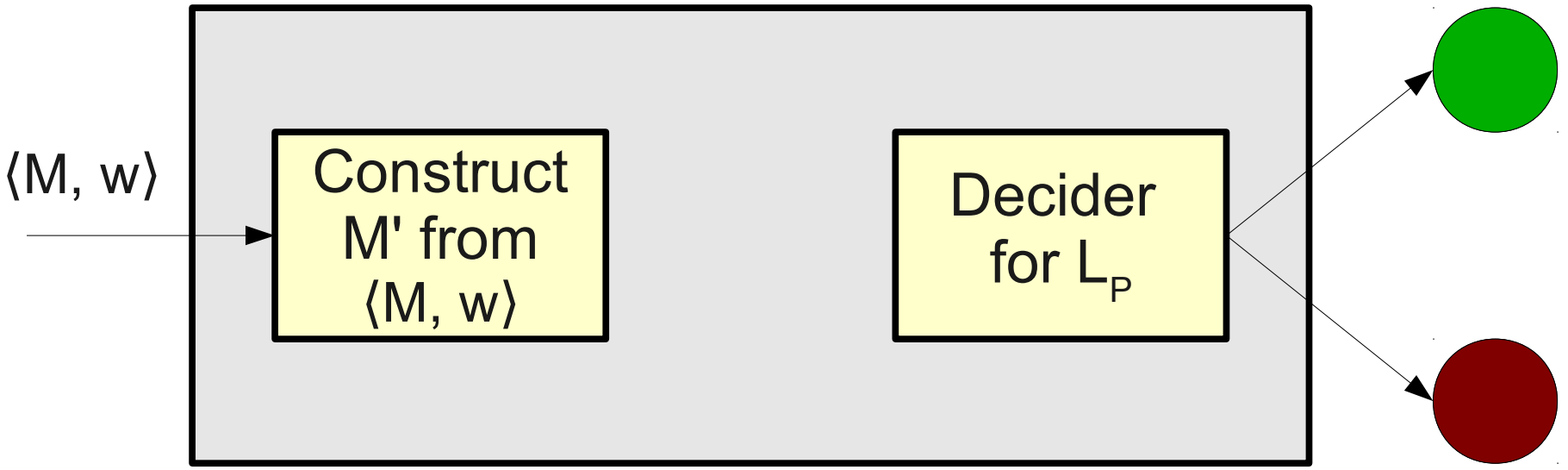
The Complete Construction



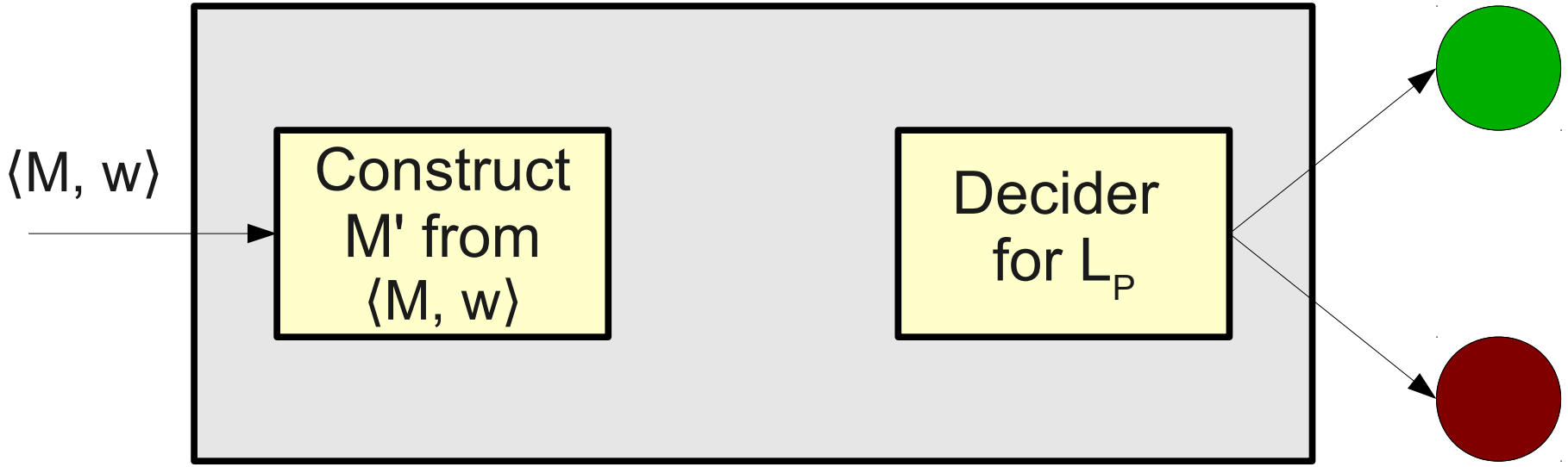
The Complete Construction

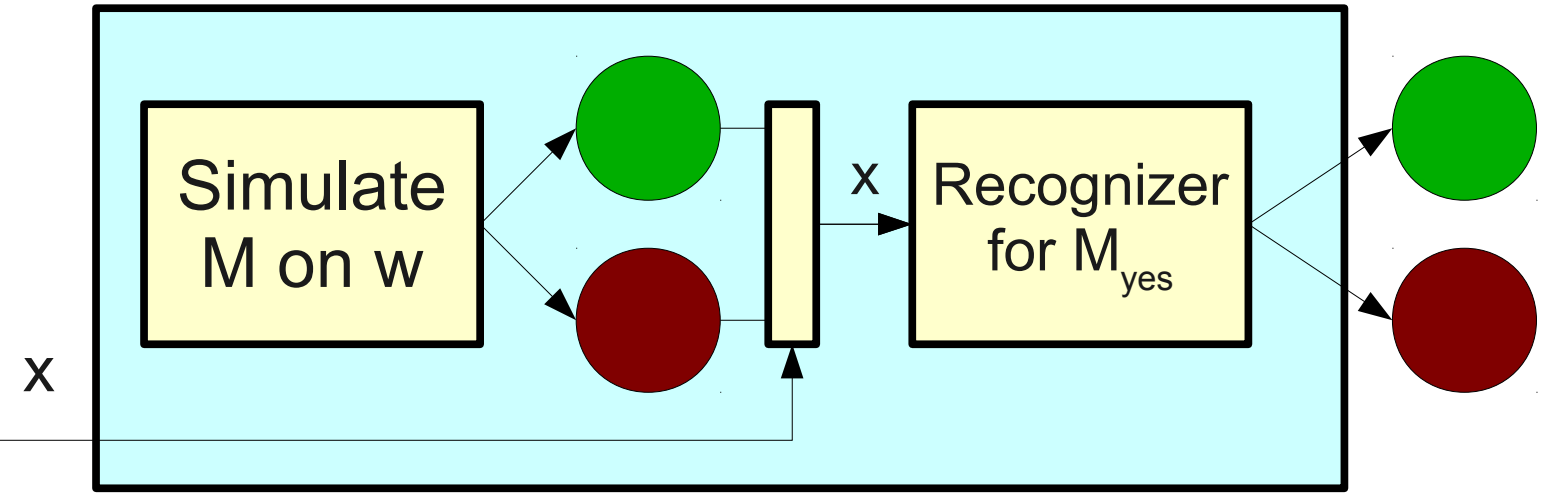


The Complete Construction

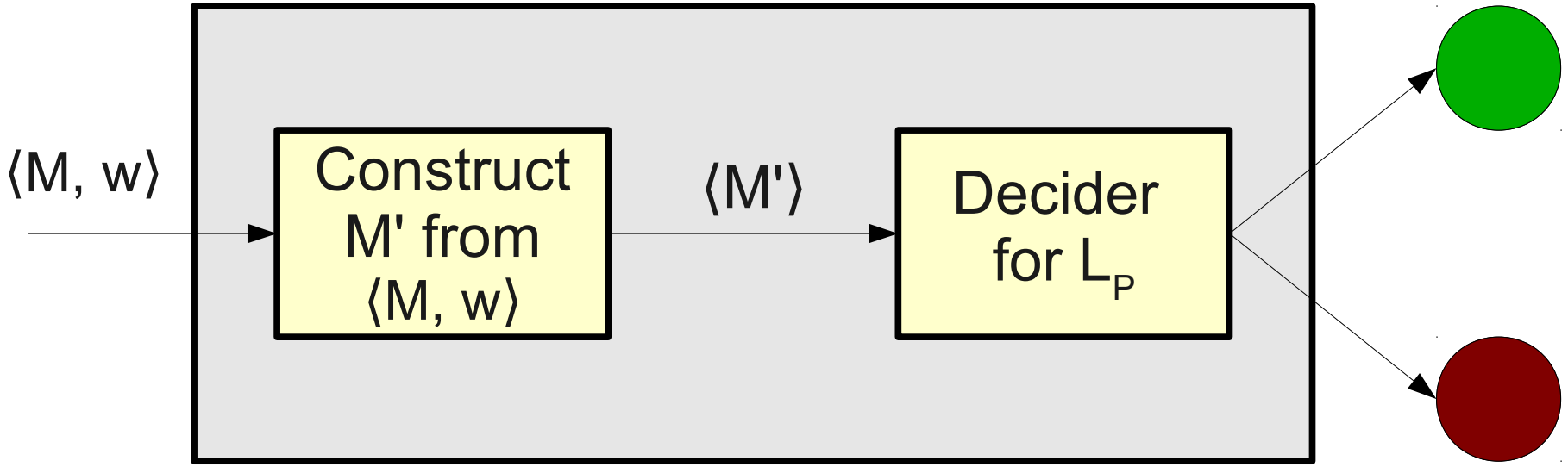


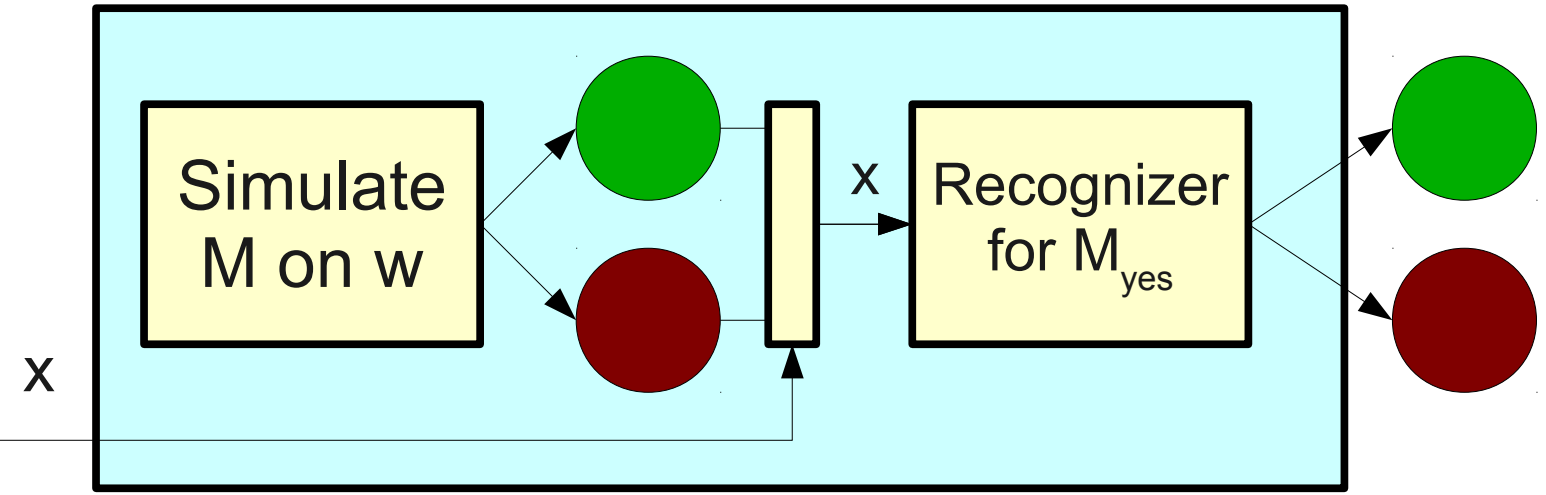
The Complete Construction



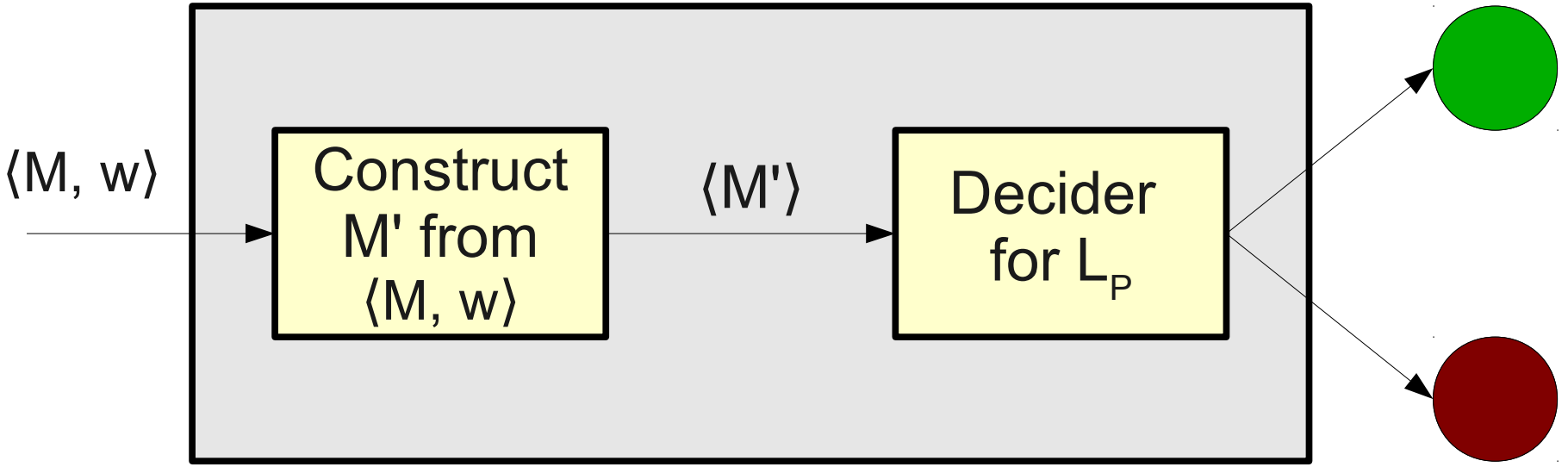


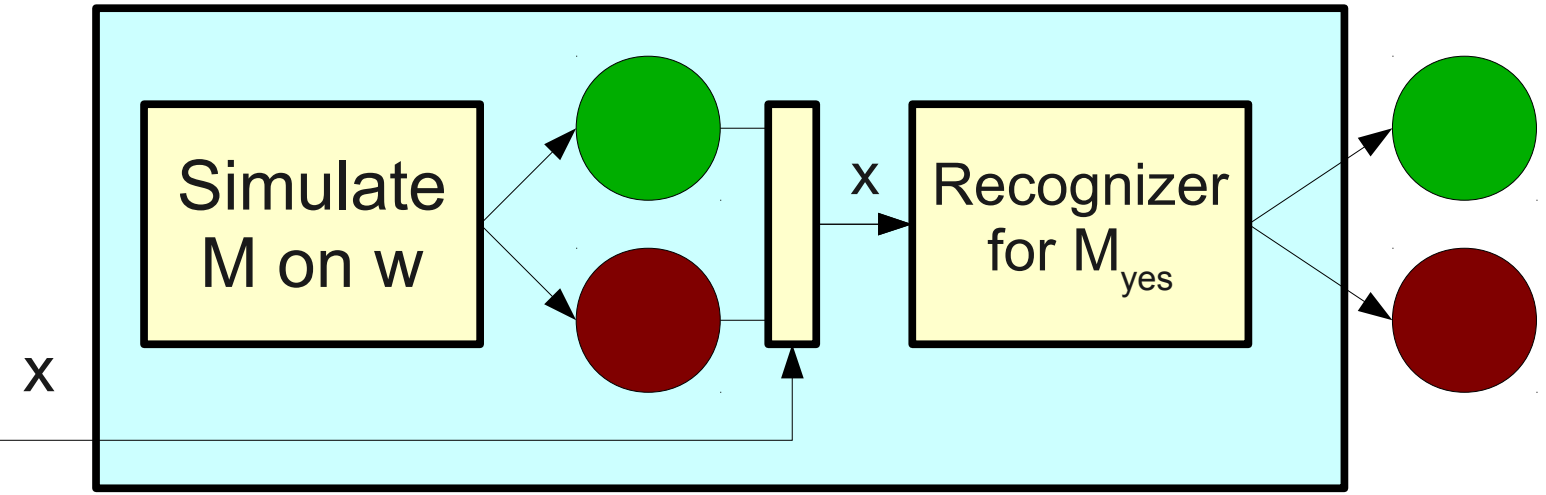
The Complete Construction





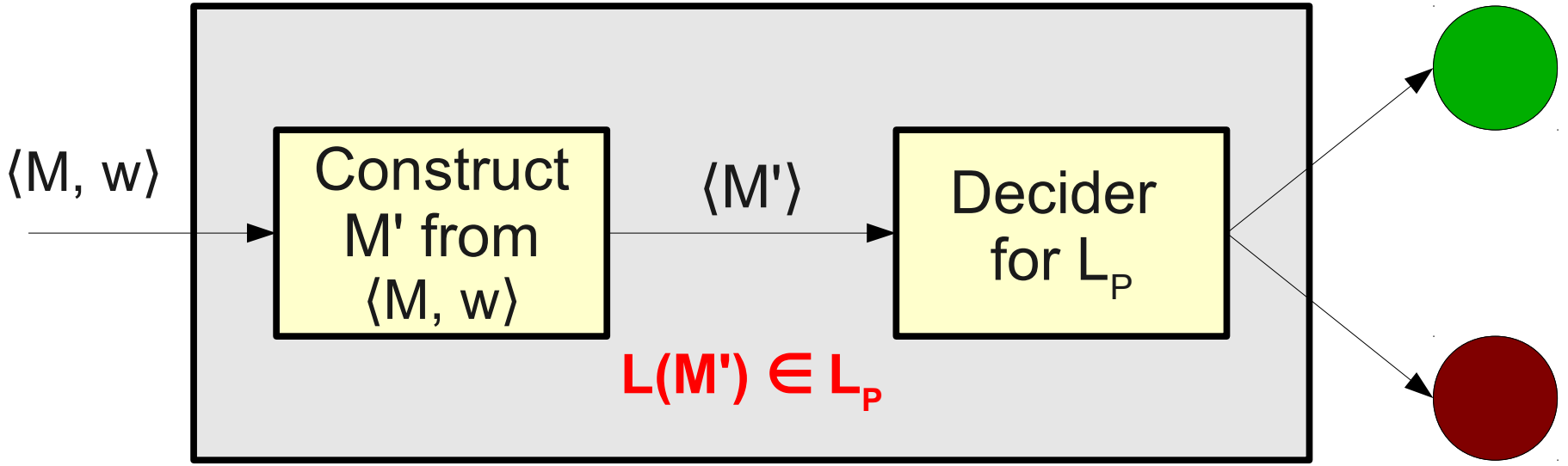
The Complete Construction

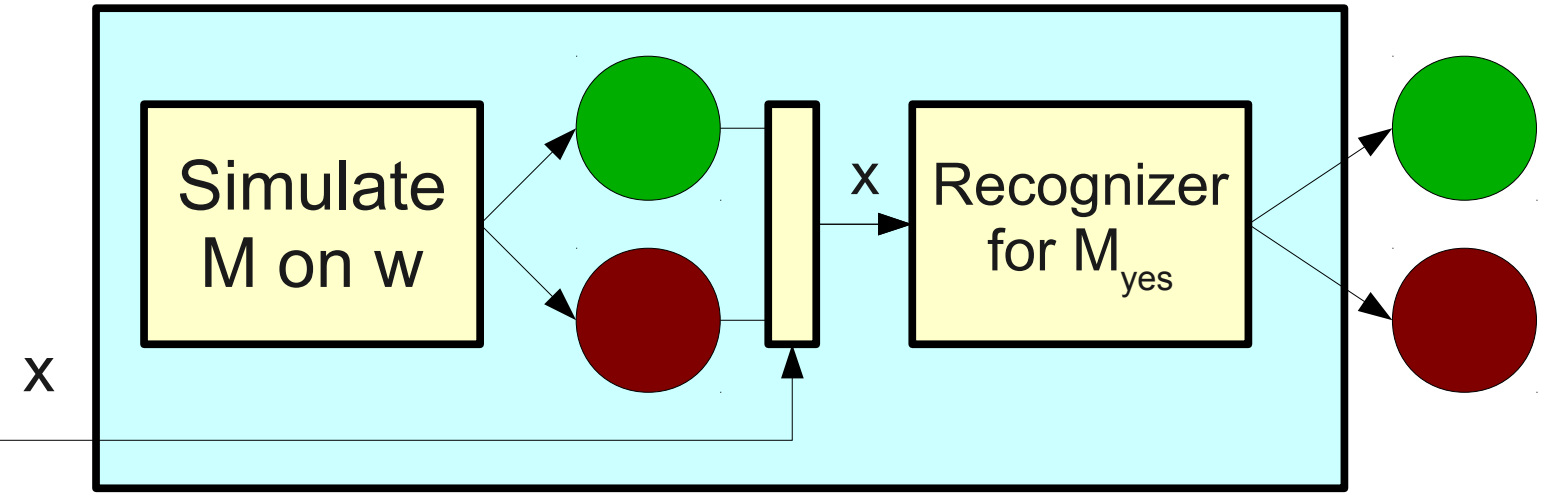




What happens if M halts on w?

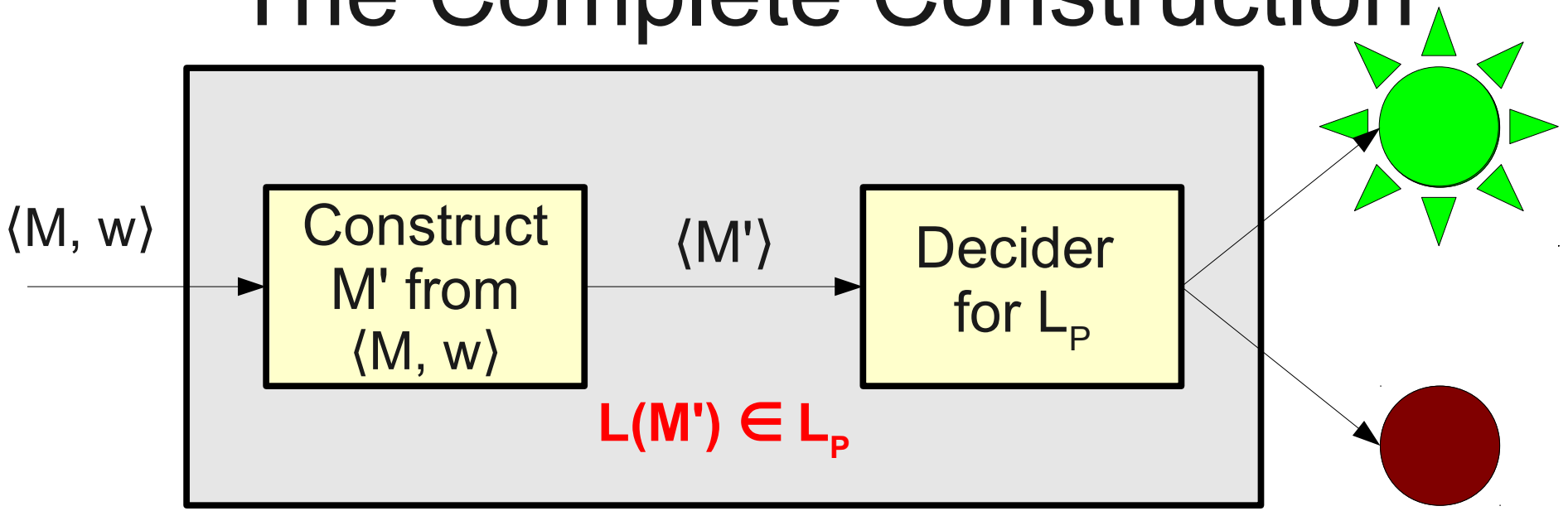
The Complete Construction

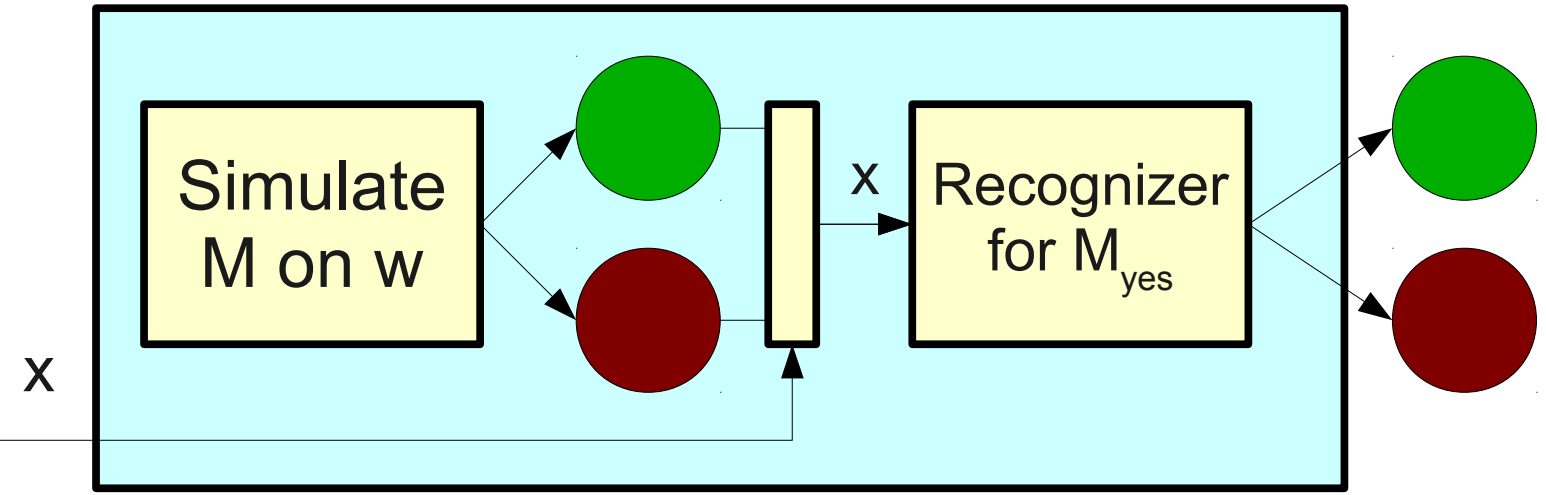




What happens if M halts on w?

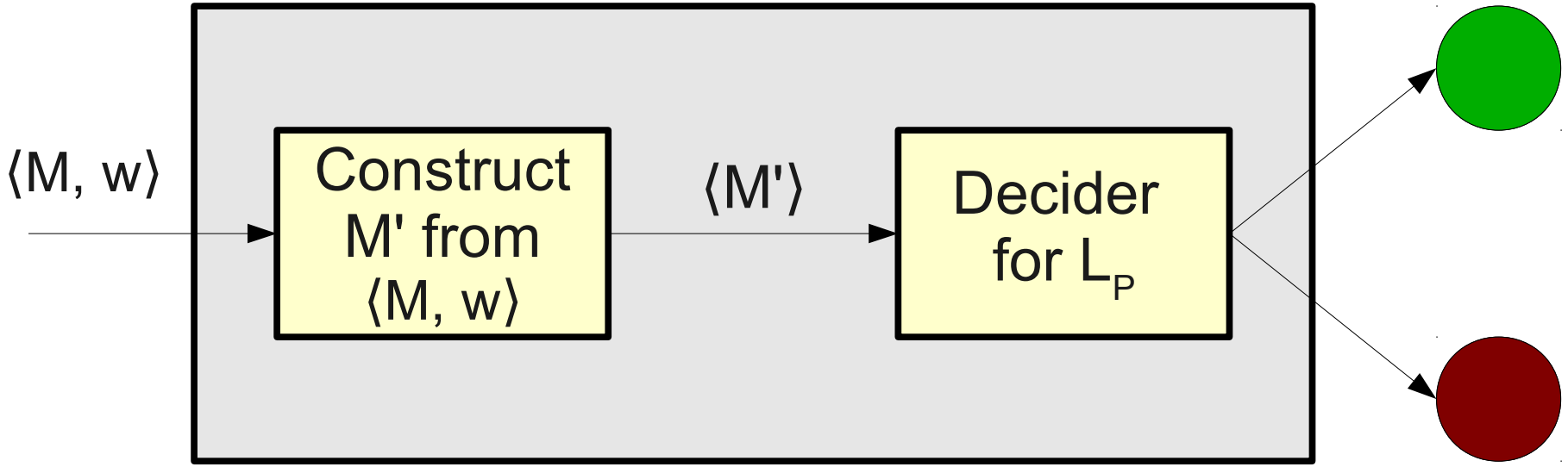
The Complete Construction

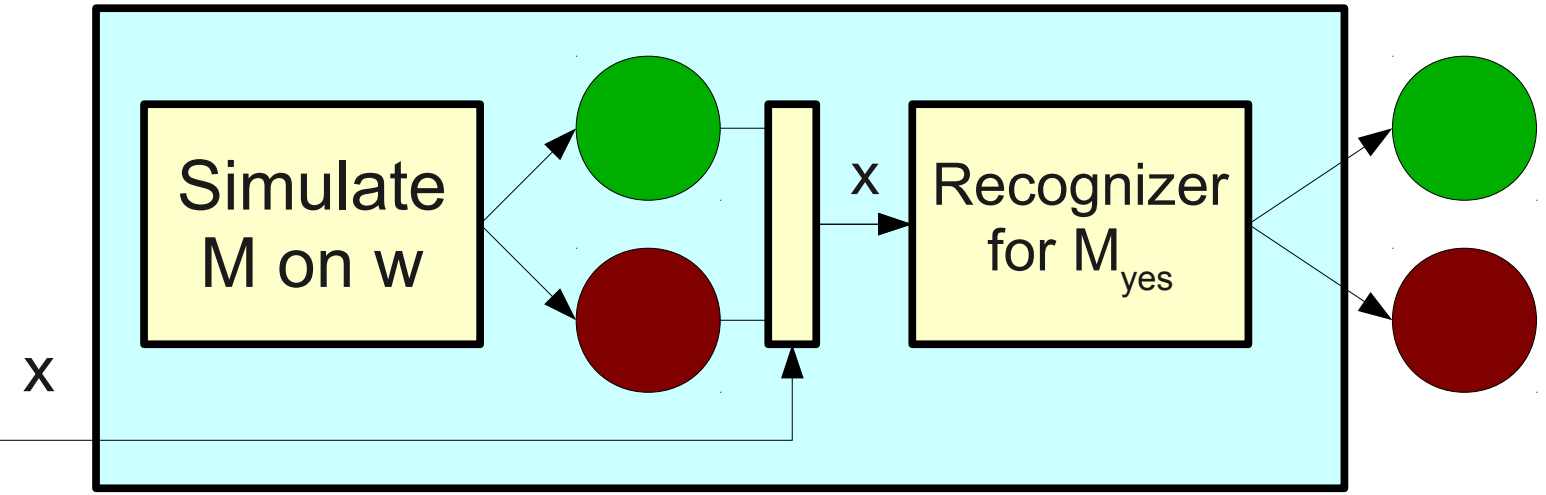




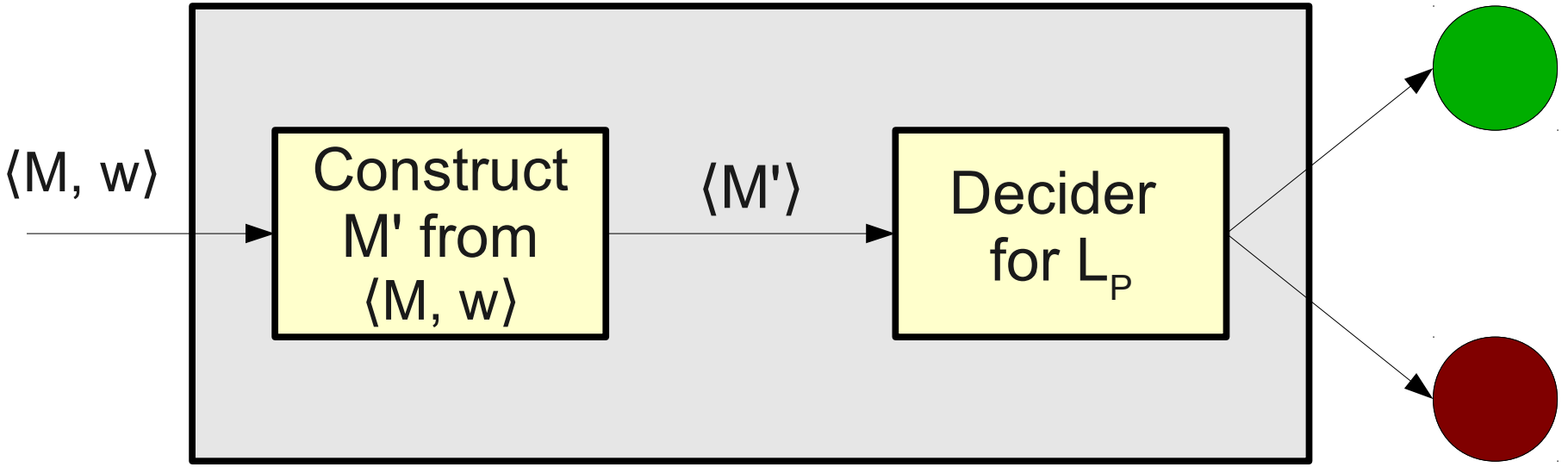
What happens if M halts on w?

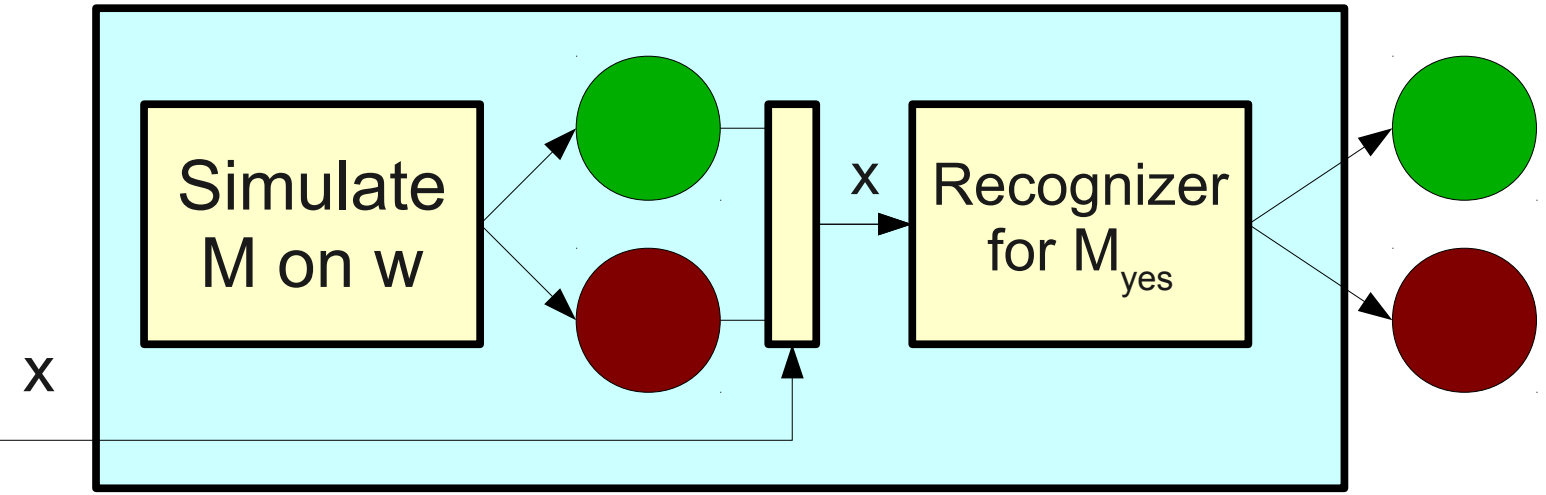
The Complete Construction





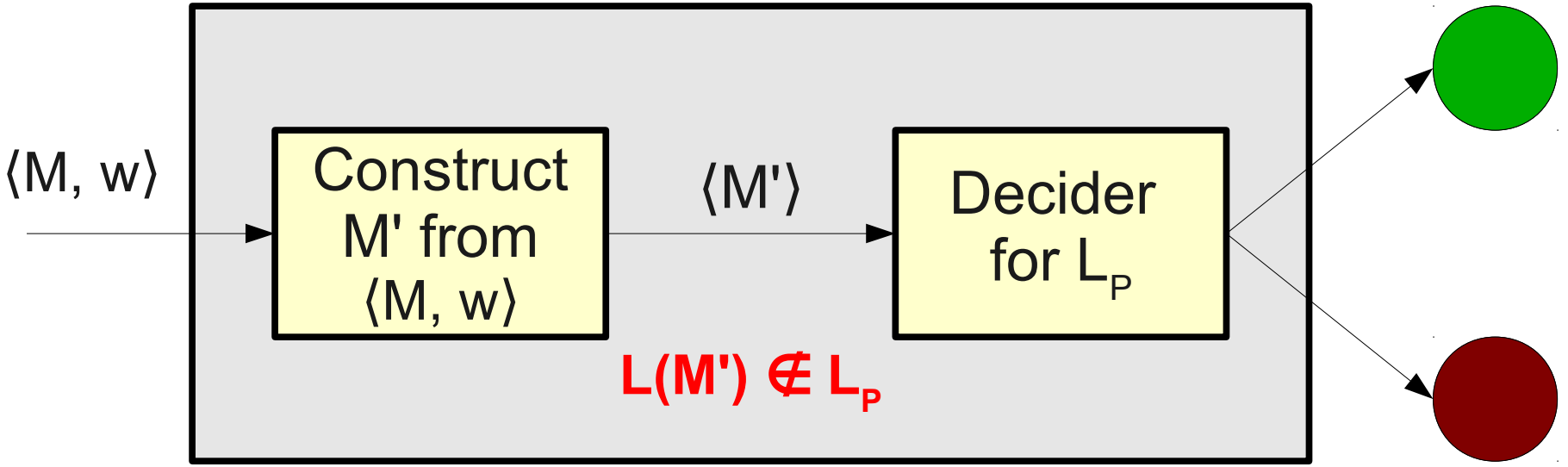
The Complete Construction

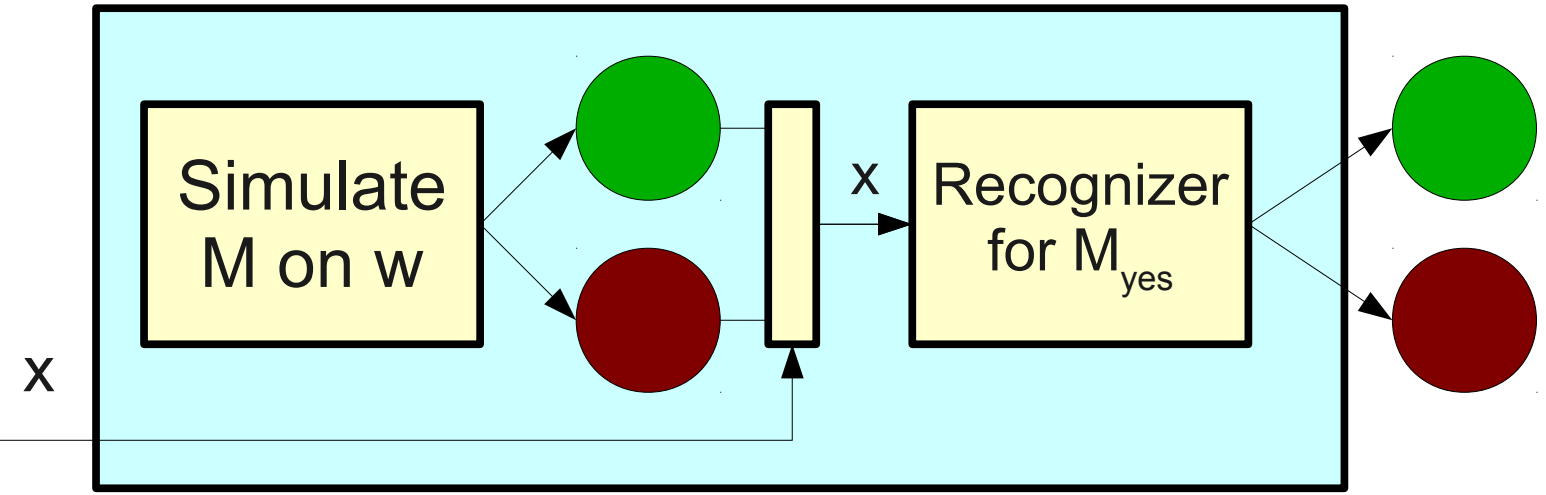




What happens if M loops on w?

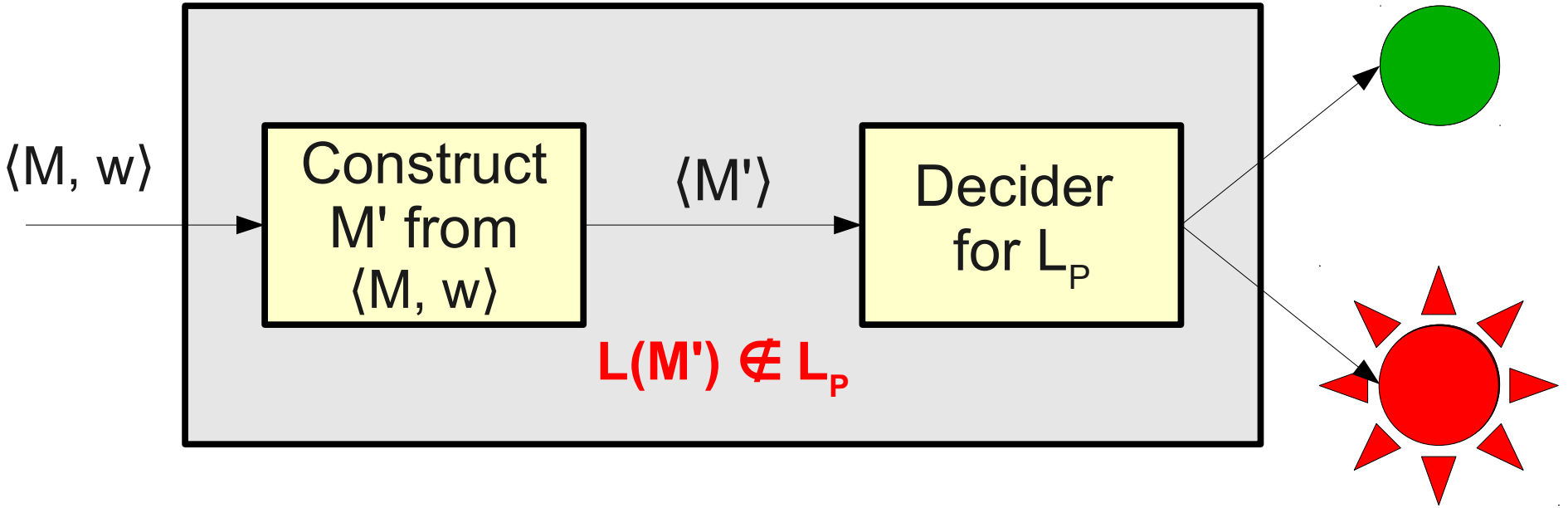
The Complete Construction

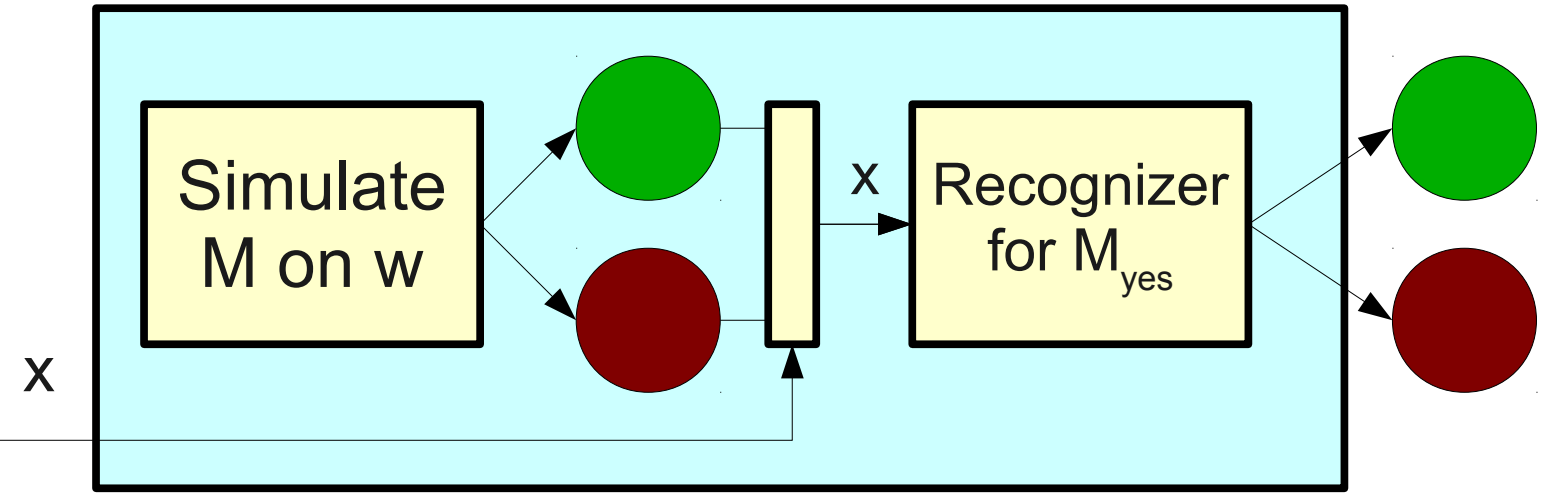




What happens if M loops on w?

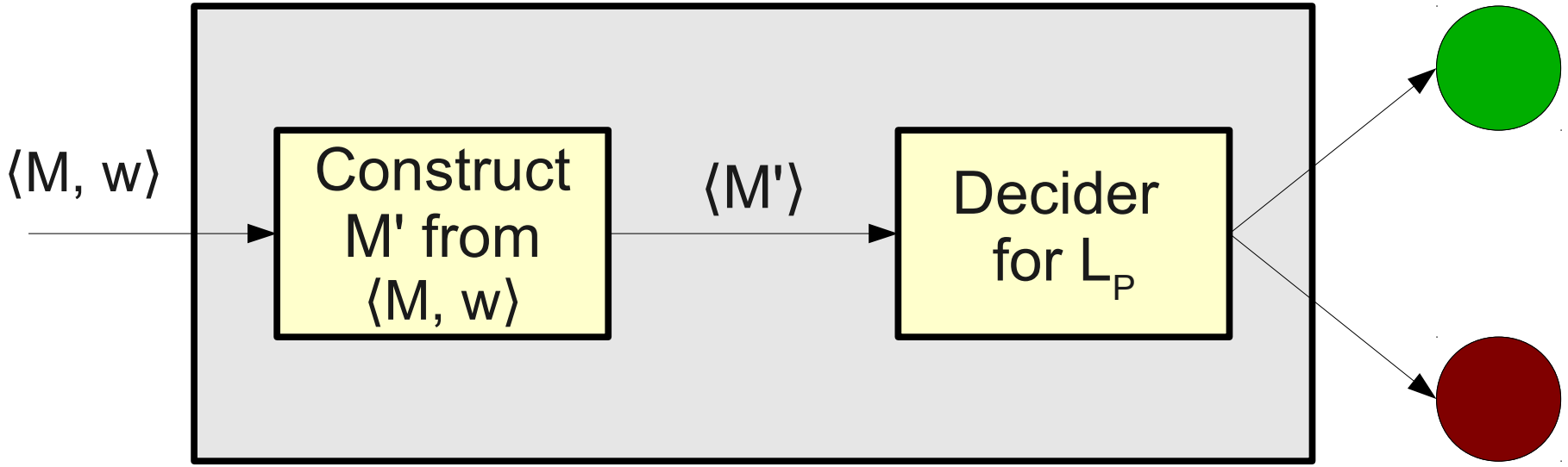
The Complete Construction

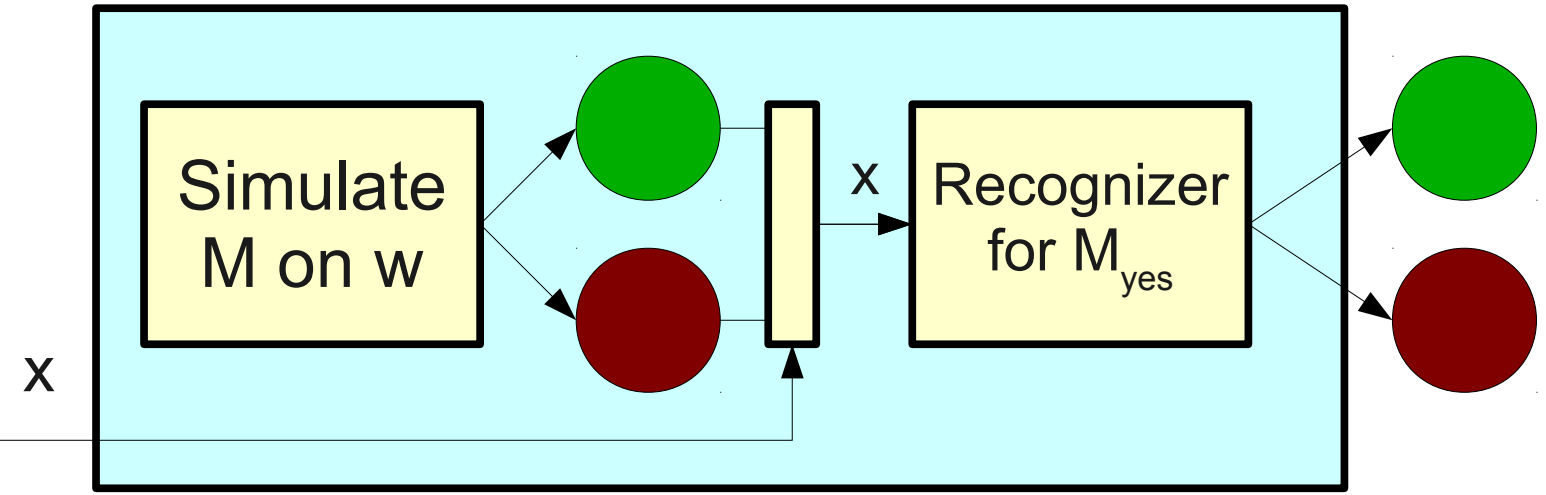




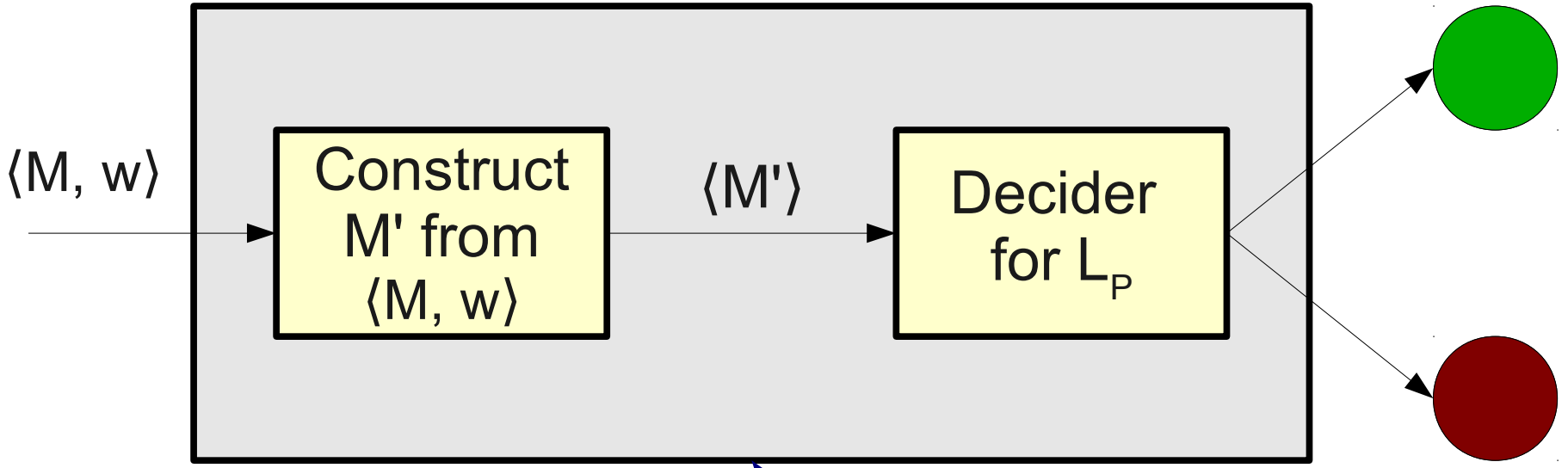
What happens if M loops on w?

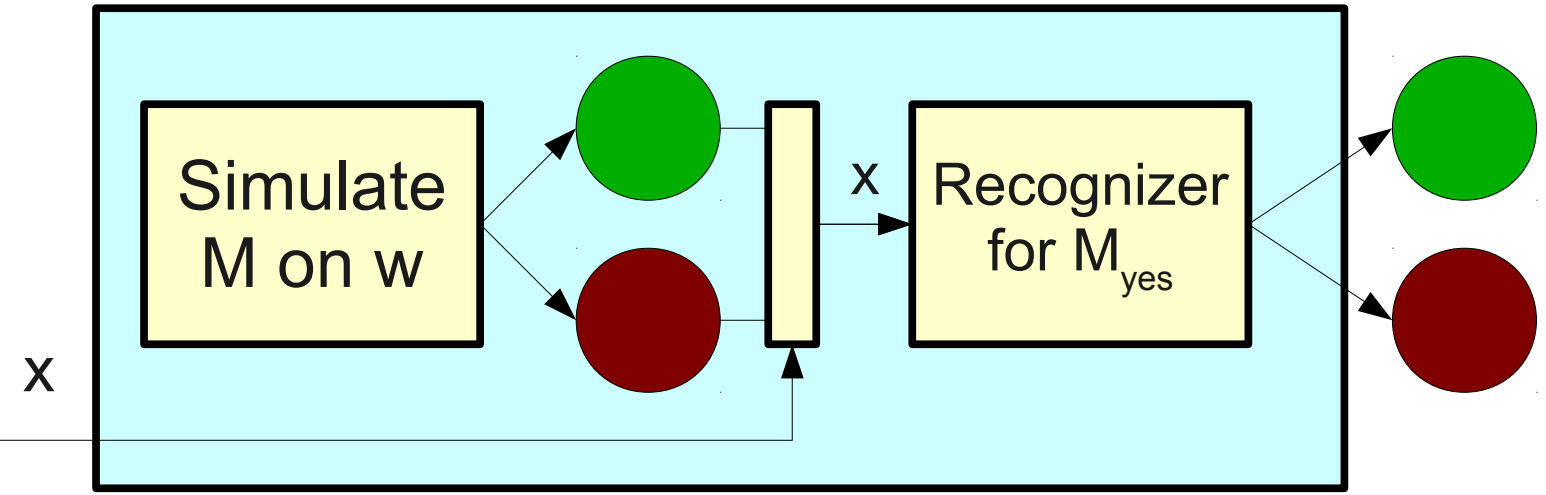
The Complete Construction





The Complete Construction





This machine decides HALT!

Theorem (Rice): Any nontrivial property of the RE languages is undecidable.
Proof: Let $M_{\emptyset}$ be a TM that accepts the empty language. Then either $\langle M_{\emptyset} \rangle \notin L_P$ or $\langle M_{\emptyset} \rangle \in L_P$. In the former case, we prove that L_P is undecidable directly. In the latter case, we prove that $\overline{L}_P$ is undecidable; this means that L_P is undecidable as well. So assume without loss of generality that $\langle M_{\emptyset} \rangle \notin L_P$. Assume for the sake of contradiction that L_P is a decidable property of the RE languages.

Since L_P is a nontrivial property of the RE languages and $\langle M_{\emptyset} \rangle \notin L_P$, there must be some TM M_{yes} such that $\langle M_{yes} \rangle \in L_P$. Moreover, $L(M_{yes}) \neq L(M_{\emptyset})$, because otherwise if $L(M_{yes}) = L(M_{\emptyset})$, we would have that either $\langle M_{\emptyset} \rangle \in L_P$ and $\langle M_{yes} \rangle \in L_P$ or $\langle M_{\emptyset} \rangle \notin L_P$ and $\langle M_{yes} \rangle \notin L_P$, both of which are false.

Because L_P is decidable, let D be a decider for L_P. Then consider the following TM:

H = "On input $\langle M, w \rangle$:
    Construct the TM M' = "On input x:
        Run M on w, and if M halts, run M_{yes} on x.
        Accept if M_{yes} accepts, reject if M_{yes} rejects."
    Run D on $\langle M' \rangle$ and accept if D accepts and reject if D rejects."

We claim that H is a decider for *HALT*. If we can show this, then we have reached a contradiction because *HALT* is undecidable. Thus our assumption was wrong, so no nontrivial property of the RE languages is decidable.

To see that H is a decider, note that since D is a decider, after we construct M' and run D on $\langle M' \rangle$, D always halts, so H always halts. Thus H halts on all inputs.

To see that L(H) = *HALT*, note that if M loops on w, then L(M') = Ø, and so D rejects $\langle M' \rangle$ because $L(M') = \emptyset = L(M_{\emptyset})$ and $\langle M_{\emptyset} \rangle \notin L_P$. Otherwise, if M halts on w, then M' accepts x iff M_{yes} accepts x, so $L(M') = L(M_{yes})$, and since $\langle M_{yes} \rangle \in L_P$, D accepts $\langle M' \rangle$. Thus H accepts $\langle M, w \rangle$ iff D accepts $\langle M' \rangle$ iff M halts on w iff $\langle M, w \rangle \in$ *HALT*, so L(H) = *HALT*. ■

Summary

- Reductions from the halting problem can be used to prove that certain languages are undecidable.
- Rice's theorem can be used to immediately prove that large classes of languages are also undecidable.

Next Time

- **Complexity Theory**
  - How hard is it to solve certain types of problems?
  - Of the decidable problems, which ones could be computed efficiently?
- **P and NP**
  - What problems can be *solved* efficiently?
  - What problems can be *checked* efficiently?