Principles of Abstract Interpretation
MIT press
Ch. **9**, Undecidability and Rice theorem

Patrick Cousot

pcousot.github.io

PrAbsInt@gmail.com github.com/PrAbsInt/

These slides are available at
http://github.com/PrAbsInt/slides/slides/slides-09--undecidability-Rice-theorem-PrAbsInt.pdf

Chapter 9

Ch. **9**, Undecidability and Rice theorem

Undecidability and Rice theorem are mathematical results showing that computers cannot fully automatically prove non-trivial properties of computer program executions (like termination)

en.wikipedia.org/wiki/Undecidable_problem
en.wikipedia.org/wiki/List_of_undecidable_problems
en.wikipedia.org/wiki/Rice's_theorem

Undecidability

Decidability, semi-decidability, and undecidability

A question (say on the semantics of a program) with boolean answer is

- *Decidable* if and only if there exists an algorithm (taking the program as input and) effectively computing the answer to the question in finite time[1];
- *Semi-decidable* if and only if there exists an algorithm (taking the program as input and) effectively computing the answer to the question in finite time when the answer is true and may be not terminating when it is false;
- *Undecidable* if and only if there exists no algorithm (taking the program as input and) effectively computing the answer to the question in finite time
- Any algorithm providing correct answers will therefore answer true, false, I don't know, or will not terminate;

`en.wikipedia.org/wiki/Decidability_(logic)`
`en.wikipedia.org/wiki/Decidability_(logic)#Semidecidability`
`en.wikipedia.org/wiki/Undecidable_problem`

[1] A further question about decidable problems is that of their computational complexity *i.e.* their inherent difficulty (`en.wikipedia.org/wiki/Computational_complexity_theory`)

Undecidability results

- The Turing Theorem 9.1 states that the halting/termination problem is algorithmically undecidable.
- Rice Theorem 9.12 states that the problem of whether a program has a given non-trivial semantic property is algorithmically undecidable.
- Our proofs are informal sketches in the style of [Jones, 1997; Reus, 2016][2]

[2] see [Rogers, 1987; Sipser, 1997] for rigorous proofs.

Turing theorem: termination is undecidable

Turing-completeness

- A programming language is *Turing-complete* if and only if it is possible to write an interpreter for this language in this language.
- For example, a language without iteration or recursion is not Turing-complete since it cannot read arbitrary long programs.
- The language of Chapters **4** and **7** is Turing complete[3].
- A programming language is *deterministic* if any input (*i.e.* given initial values of variables) has a unique outcome (including the possibility of non-termination).
- The language of Chapters **4** and **7** is deterministic.

[3] since the initial values of variables are not sequence of characters but integers, an interpreter written in this language would have to encode programs as naturals *e.g.* the natural obtained by concatenating the fixed size character codes of the sequence of characters constituting the program

Undecidability of the termination problem

The *halting/termination problem* is to write a program `termination` which

takes as input

- an arbitrary program `P` (represented as a string or file or an integer encoding)
- its input data `D` (also represented as a string or file or an integer encoding)

and such that

- `termination(P,D)` always terminates for *all* programs `P` and input data `D`
- returns
  - $\mathtt{tt}$ if the execution of the program `P` on the data `D` does terminate
  - $\mathtt{ff}$ if the execution of the program `P` on the data `D` does not terminate.

Turing proof

- The halting/termination problem is undecidable
- The proof of the halting/termination problem undecidability is due to A. Turing [Turing, 1937, 1938].
- Similar arguments had been used to prove undecidability of other problems *e.g.*
  - in number theory by A. Church [Church, 1936]
  - the incompleteness theorems in logic by K. Gödel [Gödel, 1931]

  but the originality of A. Turing is that it explicitly refers to the halting problem of a (Turing) machine modeling computers.

en.wikipedia.org/wiki/Alonzo_Church en.wikipedia.org/wiki/Kurt_Gödel
en.wikipedia.org/wiki/Alan_Turing
en.wikipedia.org/wiki/Halting_problem

Turing theorem I

Theorem 9.1 The halting/termination problem is undecidable for a deterministic Turing-complete language.

Proof sketch

- The proof by reductio ad absurdum.
- Let `interpret` be an interpreter for the language such that `interpret(P,D)` is the result of running the program `P` on the input data `D`.
- Assume `termination` does exist.
- Define the deterministic program

```
contradiction ≜ function P = if (termination(P,P)) { while (tt) ; }
```

Turing theorem II

- Run the program `contradiction` on its own text *i.e.*

```
contradiction(contradiction)
```

- More precisely, that is computed by running the interpreter `interpret(contradiction,contradiction)`.
- Then `termination(contradiction,contradiction)` is computed by running the interpreter `interpret(termination, (contradiction,contradiction))` and always terminates.
  - if `contradiction(contradiction)` does terminate then
    - `termination(contradiction, contradiction)` terminates and returns true,
    - This is a contradiction since the iteration `while (tt) ;` does not terminate; □

Turing theorem III

- or, `contradiction(contradiction)` does not terminate and then
  - `termination(contradiction, contradiction)` terminates and returns false
  - this is a contradiction since the iteration `while (tt) ;` is not entered and the program terminates. □

Examples of algorithmically undecidable problems

Other decision problems for programs can be proved undecidable by reduction to the halting/termination problem

- Constancy analysis: decide whether a program `P` assigns a value to a variable `x` different from its initial value
- Absence of runtime error: decide the absence of runtime error in an execution of a program `P`
- Sign analysis: decide whether a numerical variable has a given sign
- Type checking: decide whether a variable has a given type.
- Type inference: it follows that the more difficult type inference problem of inferring the type of a variable is also undecidable.

Undecidability proof by reduction to the halting/termination problem

Theorem 9.1 Constancy analysis is undecidable.

Proof sketch

- By contradiction, assume that constancy analysis is decidable.
- Given a program `P`, consider a fresh variable `x` not in `P`
- Consider the witness program

$$\texttt{P}' = \texttt{x = 0 ; P; x = 1 ;}.$$

- $\texttt{P}'$ assigns a value different from the initial value 0 to the variable `x` if and only if `P` terminates.
- So if the constancy problem were decidable, termination would also be decidable
- In contradiction with Turing Theorem 9.1. □

Rice's theorem generalizes undecidability to any non-trivial program property.

Failures of algorithms "solving" undecidable problems

- Let `A(P,d)` be an algorithm to solve an undecidable problem on program `P` for input data `d`.
- The `A(P,d)` algorithm is sound when it correctly answers *true*, *false*, or *fails i.e.* answers *I don't know* or does not terminate.

Corollary 9.6 A sound algorithm `A(P,d)` solving an undecidable problem must fail on infinitely many input data `d`.

Proof sketch

- By reductio ad absurdum, assume `A(P,d)` fails on finitely many input data $\mathtt{d}_i$, $i \in \Delta$ only.
- Then ask mathematicians to solve these finitely many problems and provide an `Answer(P,`$\mathtt{d}_i$`)` $\in \{true, false\}$.
- Formally check the correctness of their answers for each of $\mathtt{d}_i$, $i \in \Delta$, with a proof assistant or theorem prover.
- Consider the witness program:

```
A'(P,d) = if d ∈ {dᵢ | i ∈ Δ} then Answer(P,dᵢ)
                               else A(P,d)
```

- `A'(P,d)` always terminates and never fails to solve the undecidable problem,
- This is in contradiction with Turing Theorem 9.1. □

Semi-decidability

A problem with boolean answer is *semi-decidable* if and only if there exists an algorithm that terminate with *true* when the answer is true and might not terminate when the answer is false.

Example (Termination is semidecidable)

- Simply run the program and answer *true* upon termination.
- If the program does not terminate then there is no answer which is allowed by semi-decidability. □

Emil L. Post's semi-decidability theorem [Post, 1944]

Theorem 9.8 Decidable(P) $\Leftrightarrow$ (Semi-decidable(P) and Semi-decidable(not P)).

Proof sketch
- For ($\Rightarrow$), we have either Decidable(P) $\Rightarrow$ Semi-decidable(P), or not Decidable(P) $\Rightarrow$ Decidable(not P) $\Rightarrow$ Semi-decidable(not P).
- For ($\Leftarrow$),
  - alternatively execute one step of the algorithms for Semi-decidable(P) and Semi-decidable(not P)
  - stop when one has an answer

□

en.wikipedia.org/wiki/Emil_Leon_Post

Non-termination is not semidecidable

Theorem 9.9 The non-termination question: "Will execution of program P on data d never terminate?" is not semi-decidable (hence not decidable) for a deterministic Turing-complete language.

Proof sketch By reductio ad absurdum,
Semi-decidable (Termination) and Semi-decidable (not Termination) would imply that Decidable (Termination). □

Rice theorem

Rice theorem

Rice theorem [Rice, 1953] states that all *non-trivial semantic properties* of programs are *undecidable*.

en.wikipedia.org/wiki/Henry_Gordon_Rice
en.wikipedia.org/wiki/Rice's_theorem

Semantics

- The functional semantics $\mathcal{S}^{\mathrm{pf}}[\![\mathtt{S}]\!]$ of program components $\mathtt{S} \in \mathbb{S}$ considered in Rice theorem is a partial function on naturals

$$\mathcal{S}^{\mathrm{pf}}[\![\mathtt{S}]\!] \in \mathbb{N} \rightharpoonup \mathbb{N}$$

- The program component $\mathtt{S}$ is assumed to have one input (say in variable $\mathtt{input}_{\mathtt{S}}$) and one output (say in variable $\mathtt{output}_{\mathtt{S}}$)
- The semantics $\mathcal{S}^{\mathrm{pf}}[\![\mathtt{S}]\!]$ states that if $\mathtt{S}$ is executed with $\mathtt{input}_{\mathtt{S}}$ equal to a natural value v then
  - Either the program terminates with a final natural value $\mathcal{S}^{\mathrm{pf}}[\![\mathtt{S}]\!]v$ of $\mathtt{output}_{\mathtt{S}}$
  - Or, $\mathcal{S}^{\mathrm{pf}}[\![\mathtt{S}]\!]v$ is undefined meaning that the program execution does not terminate

The functional semantics abstracts the maximal trace semantics $\mathcal{S}^{+\infty}$

- For $v \in \mathbb{V} = \mathbb{N}$, define $\pi_v \triangleq \ell \xrightarrow{\texttt{input}_S = v} \texttt{at}[\![S]\!]$ where $\ell \notin \texttt{labx}[\![S]\!]$
- If $\pi \in \mathcal{S}^+[\![S]\!](\pi_v)$
  then $\mathcal{S}^{\texttt{pf}}[\![S]\!](v) \triangleq \varrho(\pi)\texttt{output}_S$
- Otherwise $\pi \in \mathcal{S}^\infty[\![S]\!](\pi_v) \in \mathbb{T}^\infty$ and then
  $\mathcal{S}^{\texttt{pf}}[\![S]\!](v)$ is undefined
- No other possibility (the language is deterministic)

- This abstraction yields a Galois connection

$$\mathcal{S}^{\texttt{pf}} = \alpha(\mathcal{S}^{+\infty}) = v \mapsto \langle\!| \mathcal{S}^{+\infty}[\![S]\!](\pi_v) \cap \mathbb{T}^+ = \{\pi\} \,?\, \varrho(\pi)\texttt{output}_S : \text{undefined} |\!\rangle$$

- Rice theorem holds for $\mathcal{S}^{\texttt{pf}}$ hence also for the trace semantics $\mathcal{S}^{+\infty}$

Program properties (syntactic characterization)

- A *program property* is the set of all program components which have this property
- So program properties belong to $\wp(\mathbb{S})$.
- The trivial properties are
  - the empty set $\varnothing$ of program components
  - the set $\mathbb{S}$ of all program components

Program properties (semantic characterization)

- A program property $P \in \wp(\mathbb{S})$ is a *semantic property* if and only if

$$\mathsf{s} \in P \Leftrightarrow \mathcal{S}^{\mathsf{pf}}[\![\mathsf{s}]\!] \in \gamma_{\mathcal{S}^{\mathsf{pf}}}(P)$$

where the semantic meaning $\gamma_{\mathcal{S}^{\mathsf{pf}}}(P)$ of a property $P \in \wp(\mathbb{S})$ is

$$\gamma_{\mathcal{S}^{\mathsf{pf}}}(P) \triangleq \{\mathcal{S}^{\mathsf{pf}}[\![\mathsf{s}']\!] \mid \mathsf{s}' \in P\} \in \wp(\mathbb{N} \nrightarrow \mathbb{N}).$$

- Only the $\Leftarrow$ direction is not trivial.
- For example "s has 42 minus operations -" is not a functional semantic property
- Given a statement s having 42 minus operations - we can certainly find another program with a different number of minus operations - but with the same functional semantics.

Extensional program properties

- Rice theorem considers only program properties $P \in \wp(\mathbb{S})$ which are *extensional*
- A property $P \in \wp(\mathbb{S})$ is extensional if and only if

$$\forall \mathtt{s}_1, \mathtt{s}_2 \in \mathbb{S} \,.\, \boldsymbol{\mathcal{S}}^{\mathsf{pf}}[\![\mathtt{s}_1]\!] = \boldsymbol{\mathcal{S}}^{\mathsf{pf}}[\![\mathtt{s}_2]\!] \Rightarrow (\mathtt{s}_1 \in P \Leftrightarrow \mathtt{s}_2 \in P).$$

Extensional and functional semantic program properties are equivalent I

The extensional properties are exactly the functional semantics properties:

Lemma 9.10 A property $P \in \wp(\mathbb{S})$ is extensional if and only if P is a semantic property. □

The proof is given in an appendix of these slides.

Termination and non-termination are non-trivial and extensional I

Lemma 9.11 The termination and non-termination properties (for a given input n) are non-trivial and extensional.

Proof
- Expressed with the function semantics $\mathcal{S}^{\mathsf{pf}}[\![\mathsf{S}]\!] \in \mathbb{N} \rightharpoonup \mathbb{N}$, termination for the given input n is $n \in \mathrm{dom}(\mathcal{S}^{\mathsf{pf}}[\![\mathsf{S}]\!])$
- This is the semantic property $\{f \in \mathbb{N} \rightharpoonup \mathbb{N} \mid n \in \mathrm{dom}(f)\}$
- By Lemma 9.10, this property is extensional.
- Similarly for non-termination, we consider $\{f \in \mathbb{N} \rightharpoonup \mathbb{N} \mid n \notin \mathrm{dom}(f)\}$.
- There are programs that terminates on given data and others that don't so the termination and non-termination properties are non-trivial. □

Rice theorem

Rice theorem: undecidability of semantic property verification

Theorem 9.12 Let P be a non-trivial and extensional/functional semantics property of a program s in a deterministic Turing-complete language. Then the boolean question $\mathsf{s} \in P$? is undecidable.

In short, an extensional/functional semantic property P is decidable if and only if it is trivial.

The proof is given in an appendix of these slides.

Conclusion

- By Rice Theorem 9.12, checking that a program semantics has any non-trivial semantic property is undecidable.
- This means that when a sound algorithm automatically checks for a program properties, the algorithm may not terminate, or if it always terminates then
  - either there are restrictions on the considered programs (like finite states only *i.e.* model-checking),
  - a human interaction is necessary (*e.g.* theorem proving, SMT solvers),
  - or the result may be true, false, or undetermined *i.e.* "I don't know".
- This is an indication that the verification and static program analysis problems are very difficult, even with approximate solutions [Landi, 1992]
- But they are not impossible, though without limit on their perfection!

Bibliography I

Church, Alonzo (Apr. 1936). "An Unsolvable Problem of Elementary Number Theory". *American Journal of Mathematics* 58.2, pp. 345–363.

Gödel, Kurt (1931). "Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme, I". *Monatshefte für Mathematik und Physik* 38.1, pp. 173–198.

Jones, Neil D. (1997). *Computability and complexity - from a programming perspective*. Foundations of computing series. MIT Press.

Landi, William (1992). "Undecidability of Static Analysis". *LOPLAS* 1.4, pp. 323–337.

Post, Emil (1944). "Recursively enumerable sets of positive integers and their decision problems". *Bull. Amer. Math. Soc.* 50, pp. 284–316.

Reus, Bernhard (2016). *Limits of Computation - From a Programming Perspective*. Undergraduate Topics in Computer Science. Springer.

Bibliography II

Rice, Henry Gordon (1953). "Classes of Recursively Enumerable Sets and Their Decision Problems". *Trans. Amer. Math. Soc.* 74.1, pp. 358–366.

Rogers, Hartley (1987). *Theory of recursive functions and effective computability (Reprint from 1967)*. MIT Press.

Sipser, Michael (1997). *Introduction to the theory of computation*. PWS Publishing Company.

Turing, Alan (1937). "On computable numbers, with an application to the Entscheidungsproblem". *Proceedings of the London Mathematical Society*. 2nd ser. 42, pp. 230–265.

— (1938). "On Computable Numbers, with an Application to the Entscheidungsproblem. A Correction". *Proceedings of the London Mathematical Society*. 2nd ser. 43, pp. 544–546.

Proof of the equivalence of extensional and functional semantic program properties

Extensional and functional semantic program properties I

- A program property $P \in \wp(\mathbb{S})$ is *extensional* if and only if

$$\forall \mathtt{S}_1, \mathtt{S}_2 \in \mathbb{S} \,.\, \mathcal{S}^{\mathrm{pf}}[\![\mathtt{S}_1]\!] = \mathcal{S}^{\mathrm{pf}}[\![\mathtt{S}_2]\!] \Rightarrow (\mathtt{S}_1 \in P \Leftrightarrow \mathtt{S}_2 \in P).$$

- The *semantic meaning* $\gamma_{\mathcal{S}^{\mathrm{pf}}}(P)$ of a property $P \in \wp(\mathbb{S})$ is

$$\gamma_{\mathcal{S}^{\mathrm{pf}}}(P) \triangleq \{\mathcal{S}^{\mathrm{pf}}[\![\mathtt{S}']\!] \mid \mathtt{S}' \in P\} \in \wp(\mathbb{N} \rightharpoonup \mathbb{N}).$$

- A program property $P \in \wp(\mathbb{S})$ is a *functional semantic property* if and only if

$$\mathtt{S} \in P \Leftrightarrow \mathcal{S}^{\mathrm{pf}}[\![\mathtt{S}]\!] \in \gamma_{\mathcal{S}^{\mathrm{pf}}}(P)$$

($\Rightarrow$ is trivial)

Extensional and functional semantic program properties are equivalent I

The extensional properties are exactly the functional semantics properties:

Lemma 9.10 A property $P \in \wp(\mathbb{S})$ is extensional if and only if P is a semantic property. □

Proof ($\Rightarrow$)

- Assume that property $P \in \wp(\mathbb{S})$ is extensional.
- Assume that $\mathcal{S}^{\mathrm{pf}}[\![\mathtt{s}]\!] \in \gamma_{\mathcal{S}^{\mathrm{pf}}}(P) = \{\mathcal{S}^{\mathrm{pf}}[\![\mathtt{s}']\!] \mid \mathtt{s}' \in P\}$
- Then there exists $\mathtt{s}' \in P$ such that $\mathcal{S}^{\mathrm{pf}}[\![\mathtt{s}]\!] = \mathcal{S}^{\mathrm{pf}}[\![\mathtt{s}']\!]$
- So, by extensionality, $\mathtt{s} \in P$.
- Conversely, if $\mathtt{s} \in P$ then $\mathcal{S}^{\mathrm{pf}}[\![\mathtt{s}]\!] \in \{\mathcal{S}^{\mathrm{pf}}[\![\mathtt{s}']\!] \mid \mathtt{s}' \in P\} \triangleq \gamma_{\mathcal{S}^{\mathrm{pf}}}(P)$.
- We conclude that $\mathtt{s} \in P \Leftrightarrow \mathcal{S}^{\mathrm{pf}}[\![\mathtt{s}]\!] \in \gamma_{\mathcal{S}^{\mathrm{pf}}}(P)$
- By definition, P is a functional semantic property. □

Extensional and functional semantic program properties are equivalent II

($\Leftarrow$)

- Assume that P is a functional semantic property
- Therefore $\forall \mathtt{s} \in \mathbb{S} \,.\, \mathcal{S}^{\mathrm{pf}}[\![\mathtt{s}]\!] \in \gamma_{\mathcal{S}^{\mathrm{pf}}}(P) \Leftrightarrow \mathtt{s} \in P$.
- Let $\mathtt{s}_1, \mathtt{s}_2 \in \mathbb{S}$ such that $\mathcal{S}^{\mathrm{pf}}[\![\mathtt{s}_1]\!] = \mathcal{S}^{\mathrm{pf}}[\![\mathtt{s}_2]\!]$.
- Then $(\mathtt{s}_1 \in P \Leftrightarrow \mathtt{s}_2 \in P)$ is equivalent to $(\mathcal{S}^{\mathrm{pf}}[\![\mathtt{s}_1]\!] \in \gamma_{\mathcal{S}^{\mathrm{pf}}}(P) \Leftrightarrow \mathcal{S}^{\mathrm{pf}}[\![\mathtt{s}_2]\!] \in \gamma_{\mathcal{S}^{\mathrm{pf}}}(P)$
- This is true since $\mathcal{S}^{\mathrm{pf}}[\![\mathtt{s}_1]\!] = \mathcal{S}^{\mathrm{pf}}[\![\mathtt{s}_2]\!]$.
- It follows that P is extensional. □

Proof sketch of Rice theorem

Rice theorem: undecidability of semantic property verification

Theorem 9.12 Let P be a non-trivial and extensional/functional semantics property of a program $\mathtt{s}$ in a deterministic Turing-complete language. Then the boolean question $\mathtt{s} \in P\,?$ is undecidable.

In short, an extensional/functional semantic property P is decidable if and only if it is trivial.

Proof sketch of Theorem 9.12

- We let $\texttt{input}_{\texttt{S}}$ be the variable in which statement $\texttt{S}$ finds its input data v
- We let $\texttt{output}_{\texttt{S}}$ be the variable in which statement $\texttt{S}$ writes its final result ξ in case of termination.
- The functional semantics of statement $\texttt{S}$ is therefore
  - $v \notin \mathrm{dom}(\mathcal{S}^{\mathrm{pf}}[\![\texttt{S}]\!])$ if $\texttt{S}$ does not terminate
  - $\mathcal{S}^{\mathrm{pf}}[\![\texttt{S}]\!]v = \xi$ in case of termination.
- Formally the termination question is $v \in \mathrm{dom}(\mathcal{S}^{\mathrm{pf}}[\![\texttt{S}]\!])$?
- The proof is by contradiction, assuming that $\forall \texttt{S} \in \mathbb{S} \,.\, \texttt{S} \in P\,?$ is decidable.
- Under this hypothesis, there would be a terminating algorithm $\texttt{in}_P(\texttt{S})$ returning $\texttt{tt}$ if $\texttt{S} \in P$ and $\texttt{ff}$ otherwise.

- Because P is not trivial there are programs $\mathtt{S}_t \in P$ and $\mathtt{S}_f \notin P$.
- Moreover $\mathtt{S}_d \triangleq \mathtt{while\ (tt)\ ;}$ never terminates on any input $\mathtt{input}_{\mathtt{S}_d}$ so $\mathrm{dom}(\mathcal{S}^{\mathrm{pf}}[\![\mathtt{S}]\!]) = \varnothing$.
- Consider the witness program (the value of expression $\mathtt{A}_v$ is v)

$$\begin{array}{rcl}\mathtt{S}_w & \triangleq & \mathtt{if\ (in}_P(\mathtt{S}_d)\mathtt{)\ input}_{\mathtt{S}} = \mathtt{A}_v\ \mathtt{;}\quad \mathtt{S}\quad \mathtt{input}_{\mathtt{S}_f} = \mathtt{input}_{\mathtt{S}_w}\ \mathtt{;}\quad \mathtt{S}_f\quad \mathtt{output}_{\mathtt{S}_w} = \mathtt{output}_{\mathtt{S}_f}\ \mathtt{;} \\ & & \mathtt{else\ input}_{\mathtt{S}} = \mathtt{A}_v\ \mathtt{;}\quad \mathtt{S}\quad \mathtt{input}_{\mathtt{S}_t} = \mathtt{input}_{\mathtt{S}_w}\ \mathtt{;}\quad \mathtt{S}_t\quad \mathtt{output}_{\mathtt{S}_w} = \mathtt{output}_{\mathtt{S}_t}\ \mathtt{;}\end{array}$$

- The termination program is

$$\mathtt{termination}(\mathtt{S}, v) \triangleq \mathtt{if\ (in}_P(\mathtt{S}_d)\mathtt{)}\ \neg\mathtt{in}_P(\mathtt{S}_w)\ \mathtt{else\ in}_P(\mathtt{S}_w)$$

- This program always terminates by hypothesis that P is decided by $\mathtt{in}_P$ which therefore always terminates.

- It remains to prove that $\texttt{termination}$ is correct. There are two cases.

(1) Assume that the divergent program $\mathtt{S}_d \in P$ has property P so $\texttt{in}_P(\mathtt{S}_d) = \mathtt{tt}$.
In that first case $\mathtt{S}_w$ behaves as

$$\texttt{input}_{\mathtt{S}} = \mathtt{A}_v \texttt{ ; } \quad \mathtt{S} \quad \texttt{input}_{\mathtt{S}_f} = \texttt{input}_{\mathtt{S}_w} \texttt{ ; } \quad \mathtt{S}_f \quad \texttt{output}_{\mathtt{S}_w} = \texttt{output}_{\mathtt{S}_f} \texttt{ ;}$$

There are two subcases.

(a) $v \notin \mathrm{dom}(\mathscr{S}^{\mathrm{pf}}[\![\mathtt{S}]\!])$ if and only if $\mathtt{S}$ does not terminate on input v. In that case
- $\mathtt{S}_w$ does not terminate on any input (in $\texttt{input}_{\mathtt{S}_w}$ which is not used).
- So $\mathtt{S}_w$ behaves as $\mathtt{S}_d \in P$ *i.e.* $\mathscr{S}^{\mathrm{pf}}[\![\mathtt{S}_w]\!] = \mathscr{S}^{\mathrm{pf}}[\![\mathtt{S}_d]\!]$
- $\mathtt{S}_d \in P$ so $\mathtt{S}_w \in P$ by extentionality.

(b) $v \in \mathrm{dom}(\mathscr{S}^{\mathrm{pf}}[\![\mathtt{S}]\!])$ if and only if $\mathtt{S}$ does terminate on input v. In that case,
- $\mathtt{S}_w$ behaves as $\mathtt{S}_f \notin P$ *i.e.* $\mathscr{S}^{\mathrm{pf}}[\![\mathtt{S}_w]\!] = \mathscr{S}^{\mathrm{pf}}[\![\mathtt{S}_f]\!]$ so $\mathtt{S}_w \notin P$ by extentionality.

In this first case $\mathtt{S}_d \in P$, $\mathtt{S}$ terminates on input v if and only if $\mathtt{S}_w \notin P$, so $\texttt{termination}(\mathtt{S}, v) = \neg \texttt{in}_P(\mathtt{S}_w)$ is correct.

(2) Otherwise, the divergent program $\mathtt{S}_d \notin P$ does not have property P.
In that second case $\mathtt{S}_w$ behaves as

$$\mathtt{input_S = A_v\ ;}\quad \mathtt{S}\quad \mathtt{input_{S_t} = input_{S_w}\ ;}\quad \mathtt{S_t}\quad \mathtt{output_{S_w} = output_{S_t}\ ;}$$

Again, there are two subcases.

(a) $v \notin \mathrm{dom}(\mathcal{S}^{\mathrm{pf}}[\![\mathtt{S}]\!])$ if and only if $\mathtt{S}$ does not terminate on input v.

- In that case $\mathtt{S}_w$ does not terminate on any input (in $\mathtt{input}_{\mathtt{S}_w}$ which is not used).
- So $\mathtt{S}_w$ behaves as $\mathtt{S}_d$ *i.e.* $\mathcal{S}^{\mathrm{pf}}[\![\mathtt{S}_w]\!] = \mathcal{S}^{\mathrm{pf}}[\![\mathtt{S}_d]\!]$
- $\mathtt{S}_d \notin P$ so $\mathtt{S}_w \notin P$ by extentionality.

(b) $v \in \mathrm{dom}(\mathcal{S}^{\mathrm{pf}}[\![\mathtt{S}]\!])$ if and only if $\mathtt{S}$ does terminate on input v.

- In that case $\mathtt{S}_w$ behaves as $\mathtt{S}_t$ *i.e.* $\mathcal{S}^{\mathrm{pf}}[\![\mathtt{S}_w]\!] = \mathcal{S}^{\mathrm{pf}}[\![\mathtt{S}_t]\!]$
- So $\mathtt{S}_t \in P$ implies $\mathtt{S}_w \in P$ by extentionality.

In this second case $\mathtt{S}_d \notin P$, $\mathtt{S}$ terminates on input v if and only if $\mathtt{S}_w \in P$, so $\mathtt{termination}(\mathtt{S}, v) = \mathtt{in}_P(\mathtt{S}_w)$ is correct.

We have shown that $\texttt{termination}(\texttt{S}, \nu)$ solves the termination problem in contradiction with Turing Theorem 9.1. □

Home work

- Read Ch. **9** "Undecidability and Rice theorem" of

Principles of Abstract Interpretation
Patrick Cousot
MIT Press

The End, Thank you