Course 2: Non-Terminating Algorithms in Coq

Dominique Larchey-Wendling
https://github.com/DmxLarchey/PC19

LORIA (Nancy), TU & WPI (Vienna), CNRS

PC'19, Herrsching, September 20, 2019

Introduction

- From termination to non-termination
  - how to deal with partiality?
  - $\mathbb{D}_\varphi$ = strict subset of X
  - nesting postponed to next course

Introduction

- From termination to non-termination
  - how to deal with partiality?
  - $\mathbb{D}_\varphi$ = strict subset of X
  - nesting postponed to next course
- Inductive domain predicates $\mathbb{D}_\varphi$
  - intuitive idea of its inductive structure
  - respects the rec. calls of φ
  - representation as `Acc` or `bar` predicates

Dominique Larchey-Wendling
https://github.com/DmxLarchey/PC19

Introduction

- From termination to non-termination
  - how to deal with partiality?
  - $\mathbb{D}_\varphi$ = strict subset of X
  - nesting postponed to next course
- Inductive domain predicates $\mathbb{D}_\varphi$
  - intuitive idea of its inductive structure
  - respects the rec. calls of φ
  - representation as `Acc` or `bar` predicates
- Constructive Hilbert's ϵ
  - $(\exists n,\ P\ n) \to \{n \mid P\ n\}$
  - unbounded (decidable) minimization
- Cycle detection T&H (via `bar`)
  - both non- and tail recursive
- Depth First Search (via `Acc`)
  - tail rec, hard termination charac.

Inductive Domain Predicates

- Fully specified terms
  - From a given OCaml algo. $\varphi : \alpha \to \beta$
  - with $\varphi = \mathrm{EXTR}(t_\varphi) : \mathrm{EXTR}(X_\alpha) \to \mathrm{EXTR}(X_\beta)$

$$\begin{array}{l|l}
\mathbb{D}_\varphi : X_\alpha \to \texttt{Prop} & \text{Domain} \\
\mathbb{G}_\varphi : X_\alpha \to X_\beta \to \texttt{Prop} & \text{Specification} \\
t_\varphi \; : \forall x : X_\alpha,\, \mathbb{D}_\varphi\ x \to \{y : X_\beta \mid \mathbb{G}_\varphi\ x\ y\} & \text{Implementation}
\end{array}$$

Dominique Larchey-Wendling
https://github.com/DmxLarchey/PC19

Inductive Domain Predicates

- Fully specified terms
  - From a given OCaml algo. $\varphi : \alpha \to \beta$
  - with $\varphi = \text{EXTR}(t_\varphi) : \text{EXTR}(X_\alpha) \to \text{EXTR}(X_\beta)$

$\mathbb{D}_\varphi : X_\alpha \to \texttt{Prop}$	Domain
$\mathbb{G}_\varphi : X_\alpha \to X_\beta \to \texttt{Prop}$	Specification
$t_\varphi : \forall x : X_\alpha,\ \mathbb{D}_\varphi\ x \to \{y : X_\beta \mid \mathbb{G}_\varphi\ x\ y\}$	Implementation

- An inductive structure on $D_\varphi : X_\alpha \to \texttt{Prop}$

Inductive Domain Predicates

- Fully specified terms
  - From a given OCaml algo. $\varphi : \alpha \to \beta$
  - with $\varphi = \text{EXTR}(t_\varphi) : \text{EXTR}(X_\alpha) \to \text{EXTR}(X_\beta)$

$\mathbb{D}_\varphi : X_\alpha \to \texttt{Prop}$	Domain
$\mathbb{G}_\varphi : X_\alpha \to X_\beta \to \texttt{Prop}$	Specification
$t_\varphi : \forall x : X_\alpha,\ \mathbb{D}_\varphi\ x \to \{y : X_\beta \mid \mathbb{G}_\varphi\ x\ y\}$	Implementation

- An inductive structure on $D_\varphi : X_\alpha \to \texttt{Prop}$
  - $\mathbb{D}_\varphi\ x = \exists y, \mathbb{G}_\varphi\ x\ y$ is inductive but...
  - does not respect the rec. calls of φ

Inductive Domain Predicates

- Fully specified terms
  - From a given OCaml algo. $\varphi : \alpha \to \beta$
  - with $\varphi = \text{EXTR}(t_\varphi) : \text{EXTR}(X_\alpha) \to \text{EXTR}(X_\beta)$

$\mathbb{D}_\varphi : X_\alpha \to \texttt{Prop}$	Domain
$\mathbb{G}_\varphi : X_\alpha \to X_\beta \to \texttt{Prop}$	Specification
$t_\varphi : \forall x : X_\alpha,\ \mathbb{D}_\varphi\ x \to \{y : X_\beta \mid \mathbb{G}_\varphi\ x\ y\}$	Implementation

- An inductive structure on $D_\varphi : X_\alpha \to \texttt{Prop}$
  - $\mathbb{D}_\varphi\ x = \exists y, \mathbb{G}_\varphi\ x\ y$ is inductive but...
  - does not respect the rec. calls of φ
  - let $m_\varphi(x) :=$ number of rec. calls to compute $\varphi\ x$
  - $x \in \mathbb{D}_\varphi$ iff $m_\varphi(x) < \infty$ iff $\texttt{Acc}\ R_m\ x$, WF with

$$R_m\ x\ y \quad \text{iff} \quad m_\varphi(x) < m_\varphi(y)$$

Inductive Domain Predicates

- Fully specified terms
  - From a given OCaml algo. $\varphi : \alpha \to \beta$
  - with $\varphi = \text{EXTR}(t_\varphi) : \text{EXTR}(X_\alpha) \to \text{EXTR}(X_\beta)$

$\mathbb{D}_\varphi : X_\alpha \to \texttt{Prop}$	Domain
$\mathbb{G}_\varphi : X_\alpha \to X_\beta \to \texttt{Prop}$	Specification
$t_\varphi : \forall x : X_\alpha,\ \mathbb{D}_\varphi\ x \to \{y : X_\beta \mid \mathbb{G}_\varphi\ x\ y\}$	Implementation

- An inductive structure on $D_\varphi : X_\alpha \to \texttt{Prop}$
  - $\mathbb{D}_\varphi\ x = \exists y, \mathbb{G}_\varphi\ x\ y$ is inductive but...
  - does not respect the rec. calls of φ
  - let $m_\varphi(x) :=$ number of rec. calls to compute $\varphi\ x$
  - $x \in \mathbb{D}_\varphi$ iff $m_\varphi(x) < \infty$ iff $\texttt{Acc}\ R_m\ x$, WF with

$$R_m\ x\ y \quad \text{iff} \quad m_\varphi(x) < m_\varphi(y)$$

  - but m_φ cannot be computed ($\mathbb{D}_\varphi$ not decidable)

Inductive Domain Predicates

- Fully specified terms
  - From a given OCaml algo. $\varphi : \alpha \to \beta$
  - with $\varphi = \text{EXTR}(t_\varphi) : \text{EXTR}(X_\alpha) \to \text{EXTR}(X_\beta)$

$\mathbb{D}_\varphi : X_\alpha \to \texttt{Prop}$	Domain
$\mathbb{G}_\varphi : X_\alpha \to X_\beta \to \texttt{Prop}$	Specification
$t_\varphi : \forall x : X_\alpha,\ \mathbb{D}_\varphi\ x \to \{y : X_\beta \mid \mathbb{G}_\varphi\ x\ y\}$	Implementation

- An inductive structure on $D_\varphi : X_\alpha \to \texttt{Prop}$
  - $\mathbb{D}_\varphi\ x = \exists y, \mathbb{G}_\varphi\ x\ y$ is inductive but...
  - does not respect the rec. calls of φ
  - let $m_\varphi(x) :=$ number of rec. calls to compute $\varphi\ x$
  - $x \in \mathbb{D}_\varphi$ iff $m_\varphi(x) < \infty$ iff $\texttt{Acc}\ R_m\ x$, WF with

$$R_m\ x\ y \quad \text{iff} \quad m_\varphi(x) < m_\varphi(y)$$

  - but m_φ cannot be computed ($\mathbb{D}_\varphi$ not decidable)
- Capture $\texttt{Acc}\ R_m$ by (another) $\texttt{Acc}$ (or bar) predicate

Constructive Hilbert's Epsilon

- Hypothesis for Coq's Decidable Minimization
  - $P : \mathtt{nat} \to \mathtt{Prop}$, $P_{\mathrm{dec}} : \forall n, \{P\ n\} + \{\neg P\ n\}$
- Reification of $\exists P$ into ΣP over $\mathtt{nat} \to \mathtt{Prop}$

$$\boxed{\mathtt{constructive\_epsilon} : (\exists n,\ P\ \ n) \to \{n : \mathtt{nat} \mid P\ \ n\}}$$

Constructive Hilbert's Epsilon

- Hypothesis for Coq's Decidable Minimization
  - $P : \mathtt{nat} \to \mathtt{Prop},\ P_{\mathrm{dec}} : \forall n,\ \{P\ n\} + \{\neg P\ n\}$
- Reification of $\exists P$ into ΣP over $\mathtt{nat} \to \mathtt{Prop}$

$$\boxed{\mathtt{constructive\_epsilon} : (\exists n,\ P\ \ n) \to \{n : \mathtt{nat} \mid P\ \ n\}}$$

- Decidable Unbounded Minimization

$$\boxed{\mathtt{unb\_dec\_min} : \exists P \to \{m \mid P\ \ m \wedge \forall i, P\ \ i \to m \leqslant i\}}$$

Constructive Hilbert's Epsilon

- Hypothesis for Coq's Decidable Minimization
  - $P : \texttt{nat} \to \texttt{Prop},\ P_{\mathrm{dec}} : \forall n,\ \{P\ n\} + \{\neg P\ n\}$
- Reification of $\exists P$ into ΣP over $\texttt{nat} \to \texttt{Prop}$

$$\boxed{\texttt{constructive\_epsilon} : (\exists n,\ P\ \ n) \to \{n : \texttt{nat} \mid P\ \ n\}}$$

- Decidable Unbounded Minimization

$$\boxed{\texttt{unb\_dec\_min} : \exists P \to \{m \mid P\ \ m \wedge \forall i, P\ \ i \to m \leqslant i\}}$$

- OCaml Unbounded Minimization

```
let unb_dec_min P_dec =
  let rec loop n = if P_dec n then n else loop (S n)
  in loop 0
```

Extraction of Unbounded Minimization

$$\boxed{\texttt{let rec loop}\ n = \texttt{if}\ P\ n\ \texttt{then}\ n\ \texttt{else loop}\ (\mathsf{S}\ n)}$$

- domain of $\mathbb{D}_{\texttt{loop}}\ n = \exists i, n \leqslant i \wedge P\ i$

Extraction of Unbounded Minimization

$$\boxed{\texttt{let rec loop}\ n = \texttt{if}\ P\ n\ \texttt{then}\ n\ \texttt{else loop}\ (\texttt{S}\ n)}$$

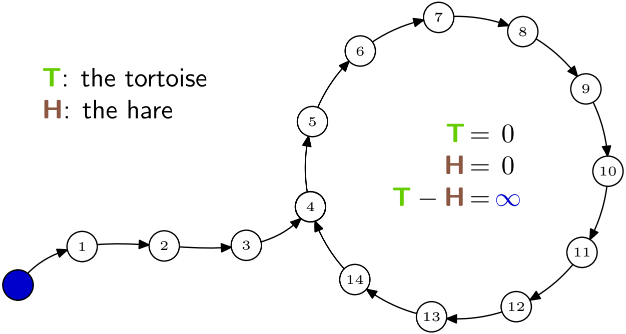
- domain of $\mathbb{D}_{\texttt{loop}}\ n = \exists i, n \leqslant i \wedge P\ i$
- $\texttt{bar} : \texttt{nat} \to \texttt{Prop}$ inductive structure on $\mathbb{D}_{\texttt{loop}}$

Two rules	$\mathbb{D}_{\texttt{loop}}$	bar
$\dfrac{P\ n}{\texttt{bar}\ n}$ $\dfrac{\texttt{bar}\ (\texttt{S}\ n)}{\texttt{bar}\ n}$	$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\times}{P\ i}}{P(i-1)\ ?}}{\cdots}}{P(\texttt{S}\ n)\ ?}}{P\ n\ ?}$	$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{P\ i}{\texttt{bar}\ i}}{\texttt{bar}\ (i-1)}}{\cdots}}{\texttt{bar}\ (\texttt{S}\ n)}}{\texttt{bar}\ n}$

The Tortoise and the Hare (T&H)

- Detecting a cycle in a (functional) graph
- Attributed to R.W. Floyd (by D.E. Knuth)

T: the tortoise
H: the hare

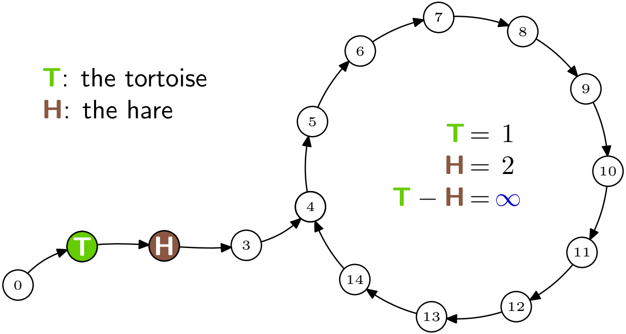


- Loop forever when no cycle (undecidable)

The Tortoise and the Hare (T&H)

- Detecting a cycle in a (functional) graph
- Attributed to R.W. Floyd (by D.E. Knuth)

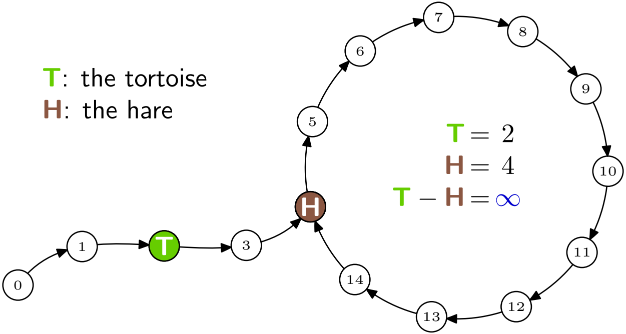
T: the tortoise
H: the hare



- Loop forever when no cycle (undecidable)

The Tortoise and the Hare (T&H)

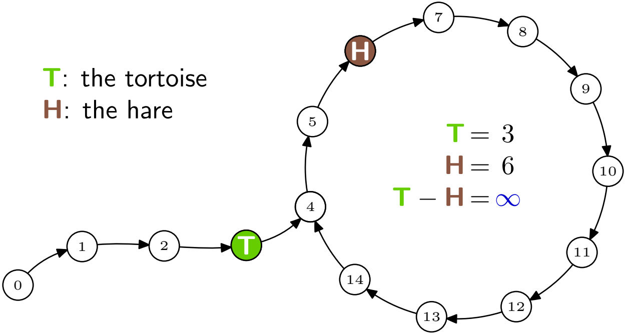
- Detecting a cycle in a (functional) graph
- Attributed to R.W. Floyd (by D.E. Knuth)



- Loop forever when no cycle (undecidable)

The Tortoise and the Hare (T&H)

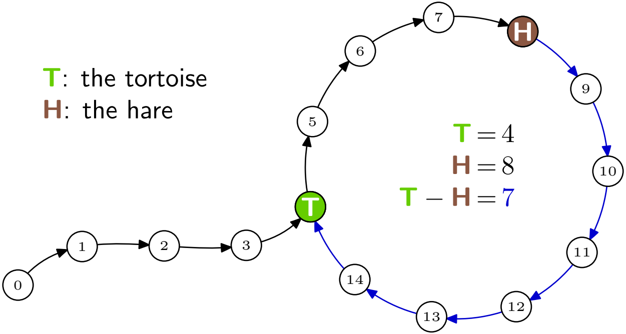
- Detecting a cycle in a (functional) graph
- Attributed to R.W. Floyd (by D.E. Knuth)



- Loop forever when no cycle (undecidable)

The Tortoise and the Hare (T&H)

- Detecting a cycle in a (functional) graph
- Attributed to R.W. Floyd (by D.E. Knuth)

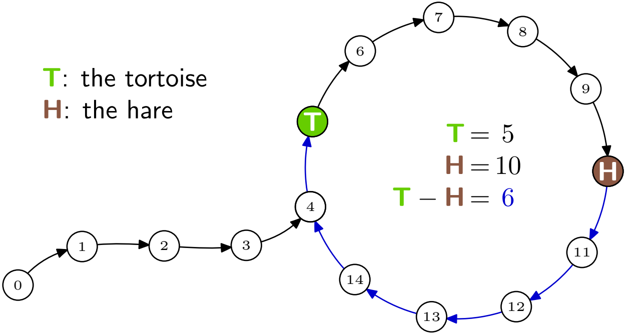


- Loop forever when no cycle (undecidable)

The Tortoise and the Hare (T&H)

- Detecting a cycle in a (functional) graph
- Attributed to R.W. Floyd (by D.E. Knuth)

T: the tortoise
H: the hare



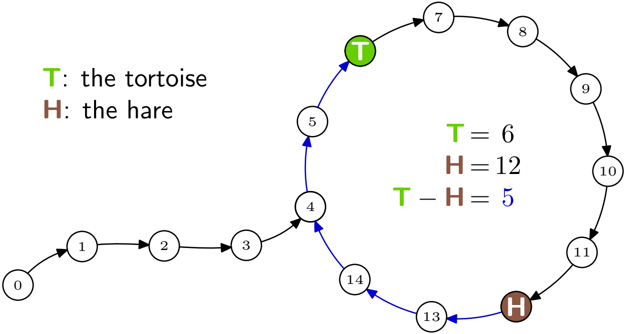
- Loop forever when no cycle (undecidable)

Dominique Larchey-Wendling
https://github.com/DmxLarchey/PC19

The Tortoise and the Hare (T&H)

- Detecting a cycle in a (functional) graph
- Attributed to R.W. Floyd (by D.E. Knuth)

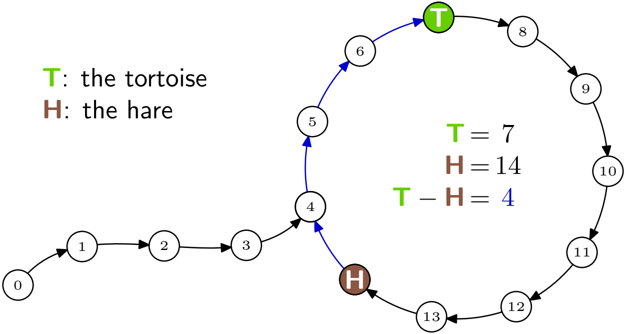
T: the tortoise
H: the hare



- Loop forever when no cycle (undecidable)

The Tortoise and the Hare (T&H)

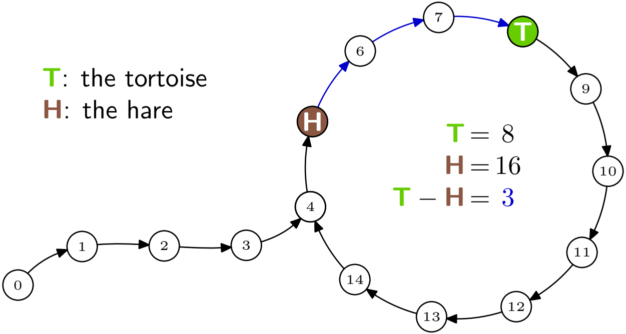
- Detecting a cycle in a (functional) graph
- Attributed to R.W. Floyd (by D.E. Knuth)



- Loop forever when no cycle (undecidable)

The Tortoise and the Hare (T&H)

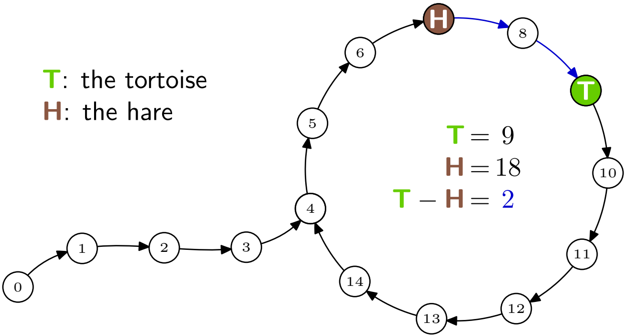
- Detecting a cycle in a (functional) graph
- Attributed to R.W. Floyd (by D.E. Knuth)



- Loop forever when no cycle (undecidable)

The Tortoise and the Hare (T&H)

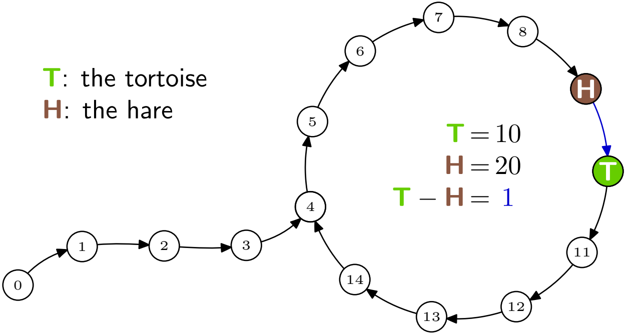
- Detecting a cycle in a (functional) graph
- Attributed to R.W. Floyd (by D.E. Knuth)



- Loop forever when no cycle (undecidable)

The Tortoise and the Hare (T&H)

- Detecting a cycle in a (functional) graph
- Attributed to R.W. Floyd (by D.E. Knuth)



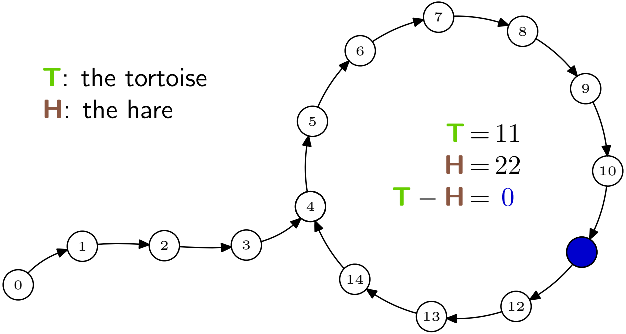
- Loop forever when no cycle (undecidable)

Dominique Larchey-Wendling
https://github.com/DmxLarchey/PC19

The Tortoise and the Hare (T&H)

- Detecting a cycle in a (functional) graph
- Attributed to R.W. Floyd (by D.E. Knuth)

T: the tortoise
H: the hare



- Loop forever when no cycle (undecidable)

Dominique Larchey-Wendling
https://github.com/DmxLarchey/PC19

A Coq specification of T&H

- Input of T&H
  - A type X and $f : X \to X$, starting point $x_0 : X$
  - eq. decider: $=^?_X : \forall x\, y : X, \{x = y\} + \{x \neq y\}$
  - satisfying cyclicity: e.g. $\exists k > 0, f^k\, x_0 = f^{2k}\, x_0$

A Coq specification of T&H

- Input of T&H
  - A type X and $f : X \to X$, starting point $x_0 : X$
  - eq. decider: $=^?_X : \forall x\, y : X, \{x = y\} + \{x \neq y\}$
  - satisfying cyclicity: e.g. $\exists k > 0, f^k\, x_0 = f^{2k}\, x_0$
- Functional specification of T&H
  - a meeting point for T&H
  - outputs $\tau > 0$ s.t. $f^\tau\, x_0 = f^{2\tau}\, x_0$

A Coq specification of T&H

- Input of T&H
  - A type X and $f : X \to X$, starting point $x_0 : X$
  - eq. decider: $=^?_X : \forall x\, y : X, \{x = y\} + \{x \neq y\}$
  - satisfying cyclicity: e.g. $\exists k > 0, f^k\, x_0 = f^{2k}\, x_0$
- Functional specification of T&H
  - a meeting point for T&H
  - outputs $\tau > 0$ s.t. $f^\tau\, x_0 = f^{2\tau}\, x_0$
- `th_spec` :

$$(\exists k, 0 < k \wedge f^k\, x_0 = f^{2k}\, x_0) \to \{\tau \mid 0 < \tau \wedge f^\tau\, x_0 = f^{2\tau}\, x_0\}$$

A Coq specification of T&H

- Input of T&H
  - A type X and $f : X \to X$, starting point $x_0 : X$
  - eq. decider: $=^?_X : \forall x\, y : X, \{x = y\} + \{x \neq y\}$
  - satisfying cyclicity: e.g. $\exists k > 0, f^k\, x_0 = f^{2k}\, x_0$
- Functional specification of T&H
  - a meeting point for T&H
  - outputs $\tau > 0$ s.t. $f^\tau\, x_0 = f^{2\tau}\, x_0$
- `th_spec` :

$$(\exists k, 0 < k \wedge f^k\, x_0 = f^{2k}\, x_0) \to \{\tau \mid 0 < \tau \wedge f^\tau\, x_0 = f^{2\tau}\, x_0\}$$

- Operational specification of T&H
  - efficently compute the sequence $(f^i\, x_0, f^{2i}\, x_0)$
  - from $i = 1, 2, \ldots$ until τ (i.e. $f^\tau\, x_0 = f^{2\tau}\, x_0$)

T&H with `constructive_epsilon` ?

- decidable unbounded minimization
- `constructive_epsilon` $(Q : \mathtt{nat} \to \mathtt{Prop})$:

$$\big(\forall n, \{Q\ n\} + \{\neg Q\ n\}\big) \to \exists Q \to \Sigma Q$$

- $Q\ \ n := 0 < n \wedge f^{\,n}\, x_0 = f^{\,2n}\, x_0$ is decidable

T&H with `constructive_epsilon` ?

- decidable unbounded minimization
- `constructive_epsilon` $(Q : \texttt{nat} \to \texttt{Prop})$:

$$\big(\forall n, \{Q\ n\} + \{\neg Q\ n\}\big) \to \exists Q \to \Sigma Q$$

- $Q\ n := 0 < n \wedge f^n\, x_0 = f^{2n}\, x_0$ is decidable
- Extraction gives

$$
\begin{array}{l}
\texttt{let th\_eps}\ f\ x_0 = \\
\quad \texttt{let rec loop}\ n = \\
\quad\quad \texttt{if}\ (0 < n)\ \text{and}\ (f^n\, x_0 = f^{2n}\, x_0) \\
\quad\quad \texttt{then}\ n \\
\quad\quad \texttt{else loop}\ (1 + n) \\
\quad \texttt{in loop}\ 0
\end{array}
$$

- wrong operational behavior

Standard T&H Ocaml implementations

- Non-tail recursive:

$$
\begin{array}{l}
\texttt{let rec th\_rec } f\ t\ h = \\
\quad \texttt{if } t = h \texttt{ then } 0 \\
\quad \texttt{else } 1 + \texttt{th\_rec } f\ (f\ t)\ (f\ (f\ h)) \\
\texttt{let th } f\ x_0 = 1 + \texttt{th\_rec } f\ (f\ x_0)\ (f\ (f\ x_0))
\end{array}
$$

Standard T&H Ocaml implementations

- Non-tail recursive:

$$\begin{array}{l}\texttt{let rec th\_rec}\ f\ t\ h =\\ \quad \texttt{if}\ t = h\ \texttt{then}\ 0\\ \quad \texttt{else}\ 1 + \texttt{th\_rec}\ f\ (f\ t)\ (f\ (f\ h))\\ \texttt{let th}\ f\ x_0 = 1 + \texttt{th\_rec}\ f\ (f\ x_0)\ (f\ (f\ x_0))\end{array}$$

- Tail-recursive:

$$\begin{array}{l}\texttt{let th\_tail}\ f\ x_0 =\\ \quad \texttt{let rec loop}\ n\ t\ h =\\ \quad\quad \texttt{if}\ t = h\ \texttt{then}\ n\\ \quad\quad \texttt{else loop}\ (1+n)\ (f\ t)\ (f\ (f\ h))\\ \quad \texttt{in loop}\ 1\ (f\ x_0)\ (f\ (f\ x_0))\end{array}$$

T&H domain as a bar predicate

- represent n such that $f^n\, t = f^{2n}\, h$ with

T&H domain as a bar predicate

- represent n such that $f^n\, t = f^{2n}\, h$ with
- a (proof of) a predicate $\texttt{bar}\ (t\ h : X) : \texttt{Prop}$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\times}{0}}{1}}{\dots}}{n-1}}{n} \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{f^n\, t = f^{2n}\, h}{\texttt{bar}\ (f^n\, t)\ (f^{2n}\, h)}}{\texttt{bar}\ (f^{n-1}\, t)\ (f^{2n-2}\, h)}}{\dots}}{\texttt{bar}\ (f^1\, t)\ (f^2\, h)}}{\texttt{bar}\ t\ h}$$

T&H domain as a bar predicate

- represent n such that $f^n\,t = f^{2n}\,h$ with
- a (proof of) a predicate $\texttt{bar}\ (t\ h : X) : \texttt{Prop}$

$$
\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\times}{0}}{1}}{\dots}}{n-1}}{n}
\qquad\qquad
\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{f^n\,t = f^{2n}\,h}{\texttt{bar}\ (f^n\,t)\ (f^{2n}\,h)}}{\texttt{bar}\ (f^{n-1}\,t)\ (f^{2n-2}\,h)}}{\dots}}{\texttt{bar}\ (f^1\,t)\ (f^2\,h)}}{\texttt{bar}\ t\ h}
$$

- n is informative, $\texttt{bar}\ t\ h$ is non-informative
- $\texttt{bar}\ t\ h$ used for termination, not computation

An inductive domain for T&H (th_rec)

- a bar inductive description of $\exists k, f^k\, t = f^{2k}\, h$

$$\frac{x = y}{\texttt{bar}\ x\ y} \qquad \frac{\texttt{bar}\ (f\ x)\ (f\ (f\ y))}{\texttt{bar}\ x\ y}$$

- show: $(\exists k, f^k\, t = f^{2k}\, h) \to \texttt{bar}\ t\ h$
- define $\texttt{th\_rec} : \forall t\, h,\ \texttt{bar}\ t\ h \to \{k \mid f^k\, t = f^{2k}\, h\}$
- by structural induction on the proof $H : \texttt{bar}\ t\ h$

$$
\begin{array}{l}
\texttt{Fixpoint th\_rec}\ t\ h\ (H : \texttt{bar}\ t\ h) : \{k \mid f^k\, t = f^{2k}\, h\} := \\
\quad \texttt{match}\ t =^?_X h\ \texttt{with} \\
\qquad \mid \texttt{lft}\ E \Rightarrow \texttt{exist}\ \_\ 0\ \mathbb{G}^?_1 \\
\qquad \mid \texttt{rt}\ C \ \ \Rightarrow \texttt{let}\ (k, H_k) := \texttt{th\_rec}\ (f\ t)\ (f^2\ h)\ \mathbb{G}^?_2 \\
\qquad\qquad\qquad\quad \texttt{in exist}\ \_\ (\texttt{S}\ k)\ \mathbb{G}^?_3 \\
\quad \texttt{end.}
\end{array}
$$

A non-tail recursive T&H (th)

- from bar $t\ h \iff \exists k, f^k\ t = f^{2k}\ h$:

$$\text{bar}\ (f\ x_0)\ (f^2\ x_0) \iff \exists k, 0 < k \wedge f^k\ x_0 = f^{2k}\ x_0$$

- for $x_0 : X$ and $H_0 : \exists \tau, 0 < \tau \wedge f^\tau\ x_0 = f^{2\tau}\ x_0$
- we get th as an instance of th_rec:

$$\begin{array}{lll}\texttt{Definition th}\ x_0\ H_0 : \{\tau \mid 0 < \tau \wedge f^\tau\ x_0 = f^{2\tau}\ x_0\} := & & \\ \quad \texttt{let} & (k, H_k) := \texttt{th\_rec}\ (f\ x_0)\ (f^2\ x_0)\ \mathbb{G}_1^? & \\ \quad \texttt{in} & \texttt{exist \_}\ (\texttt{S}\ k)\ \mathbb{G}_2^? & \end{array}$$

- $\mathbb{G}_1^?$ is proof of bar $(f\ x_0)\ (f^2\ x_0)$
- $\mathrm{EXTR}(\texttt{th})$ is the non-tail recursive T&H

A tail-recursive T&H (th_tail)

- Define a local fixpoint

$$
\begin{array}{l}
\texttt{fix loop}\ i\ t\ h\ (H : \mathsf{bar}\ t\ h) : \{k \mid\ \ \textbf{???}\ \ \} := \\
\quad \texttt{match}\ t =^?_X h\ \texttt{with} \\
\qquad \mid \texttt{left}\ E\ \ \mapsto \texttt{exist}\ \_\ i\ \mathbb{G}^?_1 \\
\qquad \mid \texttt{right}\ C \mapsto \texttt{let}\ (k, H_k) := \texttt{loop}\ (\mathsf{S}\ i)\ (f\ t)\ (f^2\ h)\ \mathbb{G}^?_2 \\
\qquad\qquad\qquad\quad \texttt{in}\ \ \texttt{exist}\ \_\ \mathtt{k}\ \mathbb{G}^?_3 \\
\quad \texttt{end}.
\end{array}
$$

- and instanciate

$$
\begin{array}{l}
\texttt{Definition th\_tail}\ x_0\ H_0 : \{\tau \mid 0 < \tau \wedge f^\tau\ x_0 = f^{2\tau}\ x_0\} := \\
\quad \texttt{let}\ (k, H_k) := \texttt{loop}\ 1\ (f\ x_0)\ (f^2\ x_0)\ \mathbb{G}^?_1\ \texttt{in exist}\ \_\ k\ \mathbb{G}^?_2
\end{array}
$$

- $\mathrm{EXTR}(\texttt{th\_tail})$ tail-recursive Ocaml code

Depth First Search

$$
\begin{array}{l}
\texttt{let rec } x \in_1 v = \\
\quad \texttt{match } v \texttt{ with} \\
\qquad \mid [\,] \qquad \to \texttt{false} \\
\qquad \mid y :: w \to y = x \texttt{ or } x \in_1 w
\end{array}
$$

$$
\begin{array}{l}
\texttt{let rec dfs } v\ l = \\
\quad \texttt{match } l \texttt{ with} \\
\qquad \mid [\,] \qquad \to v \\
\qquad \mid x :: l \to \texttt{if } x \in_1 v \\
\qquad\qquad\qquad \texttt{then dfs } v\ l \\
\qquad\qquad\qquad \texttt{else dfs } (x :: v)\ (\texttt{succs } x\ \texttt{@}\ \ l)
\end{array}
$$

Depth First Search

```
let rec x ∈₁ v =
  match v with
    | []     → false
    | y :: w → y = x or x ∈₁ w

let rec dfs v l =
  match l with
    | []     → v
    | x :: l → if x ∈₁ v
               then dfs v l
               else dfs (x :: v) (succs x @ l)
```

- For $=^?_X : \forall x\, y : X, \{b \mid x = y \iff b = \mathtt{true}\}$
- $\mathtt{succs} : X \to \mathtt{list}\ X$ (directed graph structure)

Depth First Search

let rec $x \in_1 v =$
  match v with
    $|$ [] $\rightarrow$ false
    $|$ $y :: w \rightarrow y = x$ or $x \in_1 w$

let rec dfs v l $=$
  match l with
    $|$ [] $\rightarrow v$
    $|$ $x :: l \rightarrow$ if $x \in_1 v$
        then dfs v l
        else dfs $(x :: v)$ (succs x @ l)

- For $=_X^? : \forall x\, y : X, \{b \mid x = y \iff b = \mathtt{true}\}$
- succs : $X \rightarrow$ list X (directed graph structure)
- Specification is not obvious

Depth First Search

```
let rec x ∈₁ v =
  match v with
    | []     → false
    | y :: w → y = x or x ∈₁ w

let rec dfs v l =
  match l with
    | []    → v
    | x :: l → if x ∈₁ v
               then dfs v l
               else dfs (x :: v) (succs x @ l)
```

- For $=^?_X : \forall x\, y : X, \{b \mid x = y \iff b = \texttt{true}\}$
- $\texttt{succs} : X \to \texttt{list}\ X$ (directed graph structure)
- Specification is not obvious
  - When/why does it terminate?

Depth First Search

```
let rec x ∈₁ v =
  match v with
    | []     → false
    | y :: w → y = x or x ∈₁ w

let rec dfs v l =
  match l with
    | []     → v
    | x :: l → if x ∈₁ v
               then dfs v l
               else dfs (x :: v) (succs x @ l)
```

- For $=^?_X : \forall x\, y : X, \{b \mid x = y \iff b = \mathtt{true}\}$
- $\mathtt{succs} : X \to \mathtt{list}\ X$ (directed graph structure)
- Specification is not obvious
  - When/why does it terminate?
  - What is the output?

Introduction to Induction-Recursion

$$
\begin{array}{l}
\texttt{Inductive}\ \mathbb{D}_{\texttt{dfs}} : \texttt{list}\ X \to \texttt{list}\ X \to \texttt{Prop} := \\
\quad \mid \mathbb{D}^0_{\texttt{dfs}} : \forall v, \qquad\qquad\qquad\qquad \mathbb{D}_{\texttt{dfs}}\ v\ [\,] \\
\quad \mid \mathbb{D}^1_{\texttt{dfs}} : \forall v\, x\, l,\ x \in_1 v \to \mathbb{D}_{\texttt{dfs}}\ v\ l \\
\qquad\qquad\qquad\qquad\qquad\qquad \to \mathbb{D}_{\texttt{dfs}}\ v\ (x :: l) \\
\quad \mid \mathbb{D}^2_{\texttt{dfs}} : \forall v\, x\, l,\ x \notin_1 v \to \mathbb{D}_{\texttt{dfs}}\ (x :: v)\ (\texttt{succs}\ x +\!\!+\, l) \\
\qquad\qquad\qquad\qquad\qquad\qquad \to \mathbb{D}_{\texttt{dfs}}\ v\ (x :: l) \\
\texttt{with Fixpoint dfs}\ v\ l\ (D : \mathbb{D}_{\texttt{dfs}}\ v\ l) : \texttt{list}\ X := \\
\quad \texttt{match}\ D\ \texttt{with} \\
\quad \mid \mathbb{D}^0_{\texttt{dfs}}\ v \qquad\qquad \Rightarrow v \\
\quad \mid \mathbb{D}^1_{\texttt{dfs}}\ v\ x\ l\ \_\ D \Rightarrow \texttt{dfs}\ v\ l\ D \\
\quad \mid \mathbb{D}^2_{\texttt{dfs}}\ v\ x\ l\ \_\ D \Rightarrow \texttt{dfs}\ (x :: v)\ (\texttt{succs}\ x +\!\!+\, l)\ D \\
\quad \texttt{end}
\end{array}
$$

Introduction to Induction-Recursion

$$
\begin{array}{l}
\texttt{Inductive}\ \mathbb{D}_{\texttt{dfs}} : \texttt{list}\ X \to \texttt{list}\ X \to \texttt{Prop} := \\
\quad \mid \mathbb{D}^0_{\texttt{dfs}} : \forall v, \qquad\qquad\qquad\quad\ \mathbb{D}_{\texttt{dfs}}\ v\ [] \\
\quad \mid \mathbb{D}^1_{\texttt{dfs}} : \forall v\, x\, l,\ x \in_1 v \to \mathbb{D}_{\texttt{dfs}}\ v\ l \\
\qquad\qquad\qquad\qquad\qquad\quad\ \to \mathbb{D}_{\texttt{dfs}}\ v\ (x :: l) \\
\quad \mid \mathbb{D}^2_{\texttt{dfs}} : \forall v\, x\, l,\ x \notin_1 v \to \mathbb{D}_{\texttt{dfs}}\ (x :: v)\ (\texttt{succs}\ x +\!\!+\ l) \\
\qquad\qquad\qquad\qquad\qquad\quad\ \to \mathbb{D}_{\texttt{dfs}}\ v\ (x :: l)
\end{array}
$$

$$
\begin{array}{l}
\texttt{with Fixpoint dfs}\ v\ l\ (D : \mathbb{D}_{\texttt{dfs}}\ v\ l) : \texttt{list}\ X := \\
\quad \texttt{match}\ D\ \texttt{with} \\
\quad \mid \mathbb{D}^0_{\texttt{dfs}}\ v \qquad\qquad\ \Rightarrow v \\
\quad \mid \mathbb{D}^1_{\texttt{dfs}}\ v\ x\ l\ \_\ D \Rightarrow \texttt{dfs}\ v\ l\ D \\
\quad \mid \mathbb{D}^2_{\texttt{dfs}}\ v\ x\ l\ \_\ D \Rightarrow \texttt{dfs}\ (x :: v)\ (\texttt{succs}\ x +\!\!+\ l)\ D \\
\quad \texttt{end}
\end{array}
$$

- Degenerate because `dfs` is not nested
- Coq does not have IR but we can simulate it

From the algo. to its graph

- From the dfs algorithm only

```
let rec dfs v l =
  match l with
    | []     → v
    | x :: l → if x ∈₁ v
                then dfs v l
                else dfs (x :: v) (succs x @ l)
```

From the algo. to its graph

- From the `dfs` algorithm only

$$\begin{array}{l}\texttt{let rec dfs}\ v\ l =\\ \quad\texttt{match}\ l\ \texttt{with}\\ \qquad \mid []\qquad \to v\\ \qquad \mid x :: l \to \texttt{if}\ x \in_{\texttt{l}} v\\ \qquad\qquad\qquad\ \texttt{then dfs}\ v\ l\\ \qquad\qquad\qquad\ \texttt{else dfs}\ (x :: v)\ (\texttt{succs}\ x\ \texttt{@}\ l)\end{array}$$

- Graph $\mathbb{G}_{\texttt{dfs}} : \texttt{list}\ X \to \texttt{list}\ X \to \texttt{list}\ X \to \texttt{Prop}$

$$\frac{}{\mathbb{G}_{\texttt{dfs}}\ v\ []\ v} \qquad \frac{x \in_{\texttt{l}} v \quad \mathbb{G}_{\texttt{dfs}}\ v\ l\ m}{\mathbb{G}_{\texttt{dfs}}\ v\ (x :: l)\ m}$$

$$\frac{x \notin_{\texttt{l}} v \quad \mathbb{G}_{\texttt{dfs}}\ (x :: v)\ (\texttt{succs}\ x +\!\!+\ l)\ m}{\mathbb{G}_{\texttt{dfs}}\ v\ (x :: l)\ m}$$

From the algo. to its graph

- From the dfs algorithm only

```
let rec dfs v l =
  match l with
    | []     → v
    | x :: l → if x ∈l v
                  then dfs v l
                  else dfs (x :: v) (succs x @ l)
```

- Graph $\mathbb{G}_{\mathtt{dfs}} : \mathtt{list}\ X \to \mathtt{list}\ X \to \mathtt{list}\ X \to \mathtt{Prop}$

$$\frac{}{\mathbb{G}_{\mathtt{dfs}}\ v\ []\ v} \qquad \frac{x \in_{\mathtt{l}} v \quad \mathbb{G}_{\mathtt{dfs}}\ v\ l\ m}{\mathbb{G}_{\mathtt{dfs}}\ v\ (x :: l)\ m}$$

$$\frac{x \notin_{\mathtt{l}} v \quad \mathbb{G}_{\mathtt{dfs}}\ (x :: v)\ (\mathtt{succs}\ x +\!\!+ l)\ m}{\mathbb{G}_{\mathtt{dfs}}\ v\ (x :: l)\ m}$$

- Show $\mathbb{D}_{\mathtt{dfs}}\ v\ l = (\exists m, \mathbb{G}_{\mathtt{dfs}}\ v\ l\ m) = \mathtt{Acc}\ <_{\mathrm{sc}}\ (v, l)$

$$\frac{x \in_{\mathtt{l}} v}{(v, l) <_{\mathrm{sc}} (v, x :: l)} \qquad \frac{x \notin_{\mathtt{l}} v}{(x :: v, \mathtt{succs}\ x +\!\!+ l) <_{\mathrm{sc}} (v, x :: l)}$$

From IR to correctness

- $\texttt{dfs\_full} : \forall v\, l, \mathbb{D}_{\texttt{dfs}}\ v\ l \to \{m \mid \mathbb{G}_{\texttt{dfs}}\ v\ l\ m\}$

From IR to correctness

- $\texttt{dfs\_full} : \forall v\, l, \mathbb{D}_{\texttt{dfs}}\ v\ l \to \{m \mid \mathbb{G}_{\texttt{dfs}}\ v\ l\ m\}$
- and $\texttt{dfs}\ v\ l\ D := \pi_1(\texttt{dfs\_full}\ v\ l\ D)$
- and the IR-scheme (constructors, fixpoint eqs...)

From IR to correctness

- $\texttt{dfs\_full} : \forall v\, l, \mathbb{D}_{\texttt{dfs}}\ v\ l \to \{m \mid \mathbb{G}_{\texttt{dfs}}\ v\ l\ m\}$
- and $\texttt{dfs}\ v\ l\ D := \pi_1(\texttt{dfs\_full}\ v\ l\ D)$
- and the IR-scheme (constructors, fixpoint eqs...)
- $\mathbb{D}_{\texttt{dfs}}$ has a dependent recursion principle
  - can be infered from the IR scheme
  - applies to $\mathbb{D}_{\texttt{dfs}}$-irrelevant predicates

From IR to correctness

- $\texttt{dfs\_full} : \forall v\, l, \mathbb{D}_{\texttt{dfs}}\ v\ l \to \{m \mid \mathbb{G}_{\texttt{dfs}}\ v\ l\ m\}$
- and $\texttt{dfs}\ v\ l\ D := \pi_1(\texttt{dfs\_full}\ v\ l\ D)$
- and the IR-scheme (constructors, fixpoint eqs...)
- $\mathbb{D}_{\texttt{dfs}}$ has a dependent recursion principle
  - can be infered from the IR scheme
  - applies to $\mathbb{D}_{\texttt{dfs}}$-irrelevant predicates
- $\texttt{dfs} : \forall v\, l, \mathbb{D}_{\texttt{dfs}}\ v\ l \to \texttt{list}\ X$ is domain irrelevant
  - $\texttt{dfs}\ v\ l\ D_1 = \texttt{dfs}\ v\ l\ D_2$
  - because $\mathbb{G}_{\texttt{dfs}}\ v\ l\ (\texttt{dfs}\ v\ l\ D)$ holds (π_2)
  - and $\mathbb{G}_{\texttt{dfs}}$ is functional

From IR to correctness

- $\texttt{dfs\_full} : \forall v\, l, \mathbb{D}_{\texttt{dfs}}\ v\ l \to \{m \mid \mathbb{G}_{\texttt{dfs}}\ v\ l\ m\}$
- and $\texttt{dfs}\ v\ l\ D := \pi_1(\texttt{dfs\_full}\ v\ l\ D)$
- and the IR-scheme (constructors, fixpoint eqs...)
- $\mathbb{D}_{\texttt{dfs}}$ has a dependent recursion principle
  - can be infered from the IR scheme
  - applies to $\mathbb{D}_{\texttt{dfs}}$-irrelevant predicates
- $\texttt{dfs} : \forall v\, l, \mathbb{D}_{\texttt{dfs}}\ v\ l \to \texttt{list}\ X$ is domain irrelevant
  - $\texttt{dfs}\ v\ l\ D_1 = \texttt{dfs}\ v\ l\ D_2$
  - because $\mathbb{G}_{\texttt{dfs}}\ v\ l\ (\texttt{dfs}\ v\ l\ D)$ holds (π_2)
  - and $\mathbb{G}_{\texttt{dfs}}$ is functional
- partial correctness by induction on $\mathbb{D}_{\texttt{dfs}}\ v\ l$
  - when `dfs` terminates
  - it computes a minimal invariant for `succs`

From IR to correctness

- $\mathtt{dfs\_full} : \forall v\, l, \mathbb{D}_{\mathtt{dfs}}\ v\ l \to \{m \mid \mathbb{G}_{\mathtt{dfs}}\ v\ l\ m\}$
- and $\mathtt{dfs}\ v\ l\ D := \pi_1(\mathtt{dfs\_full}\ v\ l\ D)$
- and the IR-scheme (constructors, fixpoint eqs...)
- $\mathbb{D}_{\mathtt{dfs}}$ has a dependent recursion principle
  - can be infered from the IR scheme
  - applies to $\mathbb{D}_{\mathtt{dfs}}$-irrelevant predicates
- $\mathtt{dfs} : \forall v\, l, \mathbb{D}_{\mathtt{dfs}}\ v\ l \to \mathtt{list}\ X$ is domain irrelevant
  - $\mathtt{dfs}\ v\ l\ D_1 = \mathtt{dfs}\ v\ l\ D_2$
  - because $\mathbb{G}_{\mathtt{dfs}}\ v\ l\ (\mathtt{dfs}\ v\ l\ D)$ holds (π_2)
  - and $\mathbb{G}_{\mathtt{dfs}}$ is functional
- partial correctness by induction on $\mathbb{D}_{\mathtt{dfs}}\ v\ l$
  - when `dfs` terminates
  - it computes a minimal invariant for `succs`
- we characterize termination (harder)
  - when there is an invariant
  - then `dfs` terminates