Extraction of recursive algorithms in Coq using the Braga method

Dominique Larchey-Wendling* & J.-F. Monin†

@GH/DmxLarchey/The-Braga-Method

Université de Lorraine, CNRS, LORIA* (Nancy)
CNRS & VERIMAG,† Université Grenoble Alpes

LC 2020/21, Poznań, July 24, 2021

Standard Recursion in Coq

- Structural recursion, rec. calls on sub-terms

$$\mathtt{fact}\ 0 = \mathtt{S}\ 0 \qquad \mathtt{fact}\,(\mathtt{S}\ n) = \mathtt{S}\ n \times \mathtt{fact}\ n$$

Standard Recursion in Coq

- Structural recursion, rec. calls on sub-terms

$$\texttt{fact}\ \texttt{0} = \texttt{S}\ \texttt{0} \qquad \texttt{fact}\,(\texttt{S}\ n) = \texttt{S}\ n \times \texttt{fact}\ n$$

- Well-founded recursion for $R : X \to X \to \texttt{Prop}$

$$\begin{array}{l}\texttt{Inductive Acc}\ R\ x : \texttt{Prop} :=\\ \quad \mid \texttt{Acc\_intro} : (\forall y,\ R\ y\ x \to \texttt{Acc}\ R\ y) \to \texttt{Acc}\ R\ x.\end{array}$$

$$\begin{array}{l}\texttt{Fixpoint}\ f\ x\ (T_x : \texttt{Acc}\ R\ x)\ \{\texttt{struct}\ T_x\} :=\\ \qquad \ldots\quad f\ y\ T_y \quad \ldots\end{array}$$

- Must define R before f, prove $\texttt{Acc}\ R\ x$ and ensure $\boxed{T_y <_{\mathrm{struct}} T_x}$

Standard Recursion in Coq

- Structural recursion, rec. calls on sub-terms

$$\texttt{fact}\ 0 = \texttt{S}\ 0 \qquad \texttt{fact}\,(\texttt{S}\ n) = \texttt{S}\ n \times \texttt{fact}\ n$$

- Well-founded recursion for $R : X \to X \to \texttt{Prop}$

$$\begin{array}{l}\texttt{Inductive Acc}\ R\ x : \texttt{Prop} := \\ \quad \mid \texttt{Acc\_intro} : (\forall y,\ R\ y\ x \to \texttt{Acc}\ R\ y) \to \texttt{Acc}\ R\ x.\end{array}$$

$$\begin{array}{l}\texttt{Fixpoint}\ f\ x\ (T_x : \texttt{Acc}\ R\ x)\ \{\texttt{struct}\ T_x\} := \\ \qquad \dots \quad f\ y\ T_y \quad \dots\end{array}$$

- Must define R before f, prove $\texttt{Acc}\ R\ x$ and ensure $\boxed{T_y <_{\text{struct}} T_x}$
- Particular case, decreasing measure (using `lt_wf`):
  - $R\ x\ y$ is $m\ x < m\ y$ for some $m : X \to \texttt{nat}$
- Too strong constraints for general recursion?

Complicated recursive schemes

- No obvious structural recursion, nor termination:

$$\texttt{min}\ f\ x = \texttt{if}\ f\ x = 0\ \texttt{then}\ x\ \texttt{else}\ \texttt{min}\ f\ (1 + x)$$

Complicated recursive schemes

- No obvious structural recursion, nor termination:

$$\texttt{min}\ f\ x = \texttt{if}\ f\ x = 0\ \texttt{then}\ x\ \texttt{else}\ \texttt{min}\ f\ (1+x)$$

$$\texttt{iter}_0\ f\ n = \texttt{if}\ n = 0\ \texttt{then}\ []\ \texttt{else}\ n :: \texttt{iter}_0\ f\ (f\ n)$$

Complicated recursive schemes

- No obvious structural recursion, nor termination:

$$\texttt{min}\ f\ x = \texttt{if}\ f\ x = 0\ \texttt{then}\ x\ \texttt{else}\ \texttt{min}\ f\ (1+x)$$

$$\texttt{iter}_0\ f\ n = \texttt{if}\ n = 0\ \texttt{then}\ []\ \texttt{else}\ n :: \texttt{iter}_0\ f\ (f\ n)$$

$$\texttt{th}\ f\ x\ y = \texttt{if}\ x = y\ \texttt{then}\ 0\ \texttt{else}\ 1 + \texttt{th}\ f\ (f\ x)\ (f^2\ y)$$

Complicated recursive schemes

- No obvious structural recursion, nor termination:

$$\texttt{min}\ f\ x = \texttt{if}\ f\ x = 0\ \texttt{then}\ x\ \texttt{else}\ \texttt{min}\ f\ (1+x)$$

$$\texttt{iter}_0\ f\ n = \texttt{if}\ n = 0\ \texttt{then}\ []\ \texttt{else}\ n :: \texttt{iter}_0\ f\ (f\ n)$$

$$\texttt{th}\ f\ x\ y = \texttt{if}\ x = y\ \texttt{then}\ 0\ \texttt{else}\ 1 + \texttt{th}\ f\ (f\ x)\ (f^2\ y)$$

- Complicated termination proof ($succs : \mathcal{V} \to \mathbb{L}\,\mathcal{V}$):

$$\begin{array}{lcll}
\texttt{dfs}\ v\ [] & = & v & \\
\texttt{dfs}\ v\ (x :: l) & = & \texttt{dfs}\ v\ l & \text{if } x \in v \\
\texttt{dfs}\ v\ (x :: l) & = & \texttt{dfs}\ (x :: v)\ (succs\ x \mathbin{+\!\!+} l) & \text{if } x \notin v
\end{array}$$

More complicated recursive schemes

- Nesting/mutual recursion:

  McCarthy $\quad f_{91}\ x = \texttt{if}\ x > 100\ \texttt{then}\ x - 10\ \texttt{else}\ f_{91}^2(x + 11)$

More complicated recursive schemes

- Nesting/mutual recursion:

McCarthy $f_{91}\ x = \texttt{if}\ x > 100\ \texttt{then}\ x - 10\ \texttt{else}\ f_{91}^2(x + 11)$

Knuth 1991 $k_{91}\ x = \texttt{if}\ x > a\ \texttt{then}\ x - b\ \texttt{else}\ k_{91}^c(x + d)$

where $f^n\ x = \texttt{iter}_p\ f\ n\ x$

$\texttt{iter}_p\ f\ n\ x = \texttt{if}\ n = 0\ \texttt{then}\ x\ \texttt{else}\ \texttt{iter}_p\ f\ (n - 1)\ (f\ x)$

More complicated recursive schemes

- Nesting/mutual recursion:

McCarthy $f_{91}\ x = \texttt{if}\ x > 100\ \texttt{then}\ x - 10\ \texttt{else}\ f_{91}^{2}(x+11)$

Knuth 1991 $k_{91}\ x = \texttt{if}\ x > a\ \texttt{then}\ x - b\ \texttt{else}\ k_{91}^{c}(x+d)$

where $f^{n}\ x = \texttt{iter}_p\ f\ n\ x$

$\texttt{iter}_p\ f\ n\ x = \texttt{if}\ n = 0\ \texttt{then}\ x\ \texttt{else}\ \texttt{iter}_p\ f\ (n-1)\ (f\ x)$

- Nesting&hard termination: $\texttt{unif}\ (m \cdot n)\ (m' \cdot n')$ is

$$\begin{cases} \emptyset & \text{if } \texttt{unif}\ m\ m' = \emptyset \\ \emptyset & \text{if } \texttt{unif}\ m\ m' = \lfloor\rho\rfloor \text{ and } \texttt{unif}\ (\rho\ n)\ (\rho\ n') = \emptyset \\ \lfloor\sigma \circ \rho\rfloor & \text{if } \texttt{unif}\ m\ m' = \lfloor\rho\rfloor \text{ and } \texttt{unif}\ (\rho\ n)\ (\rho\ n') = \lfloor\sigma\rfloor \end{cases}$$

Extraction in Coq

- `Extraction` = Coq command
  - auto. maps a Coq term to a program (OCaml)
  - captures the Computational Contents (CC)

Extraction in Coq

- `Extraction` = Coq command
  - auto. maps a Coq term to a program (OCaml)
  - captures the Computational Contents (CC)
- Consider a fully specified term t:

$$t : \forall x : X, \mathbb{D}\ x \to \{y : Y \mid \mathbb{G}\ x\ y\}$$

$\mathbb{D} : X \to \texttt{Prop}$	Domain	Pre-condition
$\mathbb{G} : X \to Y \to \texttt{Prop}$	Specification	Post-condition

Extraction in Coq

- `Extraction` = Coq command
  - auto. maps a Coq term to a program (OCaml)
  - captures the Computational Contents (CC)
- Consider a fully specified term t:

$$t : \forall x : X, \mathbb{D}\ x \to \{y : Y \mid \mathbb{G}\ x\ y\}$$

$\mathbb{D} : X \to \texttt{Prop}$	Domain	Pre-condition
$\mathbb{G} : X \to Y \to \texttt{Prop}$	Specification	Post-condition

- $\mathbb{D}\, x$ (domain) and $\mathbb{G}\, x\, y$ (spec)
  - are erased at extraction
  - $\text{EXTR}(t) : \text{EXTR}(X) \to \text{EXTR}(Y)$

Extraction in Coq

- `Extraction` = Coq command
  - auto. maps a Coq term to a program (OCaml)
  - captures the Computational Contents (CC)
- Consider a fully specified term t:

$$t : \forall x : X, \mathbb{D}\ x \to \{y : Y \mid \mathbb{G}\ x\ y\}$$

$\mathbb{D} : X \to$ `Prop`	Domain	Pre-condition
$\mathbb{G} : X \to Y \to$ `Prop`	Specification	Post-condition

- $\mathbb{D}\,x$ (domain) and $\mathbb{G}\,x\,y$ (spec)
  - are erased at extraction
  - $\mathrm{EXTR}(t) : \mathrm{EXTR}(X) \to \mathrm{EXTR}(Y)$
- What do $\mathbb{D}$ and $\mathbb{G}$ become?
  - it depends...
  - now: they are just erased
  - ideally (shortly ?): correctness of $\mathrm{EXTR}(t)$

Certification by Extraction

- How to certify by extraction ?
  - From a given OCaml algo. $\varphi : \alpha \to \beta$
  - Get $\varphi = \text{EXTR}(t_\varphi) : \text{EXTR}(X_\alpha) \to \text{EXTR}(X_\beta)$

$\mathbb{D}_\varphi : X_\alpha \to \texttt{Prop}$	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

Certification by Extraction

- How to certify by extraction ?
  - From a given OCaml algo. $\varphi : \alpha \to \beta$
  - Get $\varphi = \text{EXTR}(t_\varphi) : \text{EXTR}(X_\alpha) \to \text{EXTR}(X_\beta)$

$\mathbb{D}_\varphi : X_\alpha \to \texttt{Prop}$	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

- $\mathbb{D}_\varphi\, x$ (domain) and $\mathbb{G}_\varphi\, x\, y$ (spec)
  - erased at extraction
  - but contain the statement of correctness

Certification by Extraction

- How to certify by extraction ?
  - From a given OCaml algo. $\varphi : \alpha \to \beta$
  - Get $\varphi = \text{EXTR}(t_\varphi) : \text{EXTR}(X_\alpha) \to \text{EXTR}(X_\beta)$

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

- $\mathbb{D}_\varphi\, x$ (domain) and $\mathbb{G}_\varphi\, x\, y$ (spec)
  - erased at extraction
  - but contain the statement of correctness
- Problem: how to define such a t_φ in Coq ?
  - no `let rec`, only restricted `Fixpoints` (`struct`)
  - How to control the CC ?

Some influencial references

- Non-constructive recursion:
  - *Termination of Nested and Mutually Recursive Algorithms* (Giesl 97)
  - *Partial and Nested Recursive Function Definitions in Higher-Order Logic* (Krauss 09)
  - *Partiality and Recursion in Interactive Theorem Provers - An Overview* (Bove&Krauss&Sozeau 15)

Some influencial references

- Non-constructive recursion:
  - *Termination of Nested and Mutually Recursive Algorithms* (Giesl 97)
  - *Partial and Nested Recursive Function Definitions in Higher-Order Logic* (Krauss 09)
  - *Partiality and Recursion in Interactive Theorem Provers - An Overview* (Bove&Krauss&Sozeau 15)
- Constructive recursion:
  - *General recursion in TT* (Bove&Capretta 05)
  - the `Equations` package (2010 & 2019)
  - *The Braga method* (Types 2018 & WS 2021)

Some influencial references

- Non-constructive recursion:
  - *Termination of Nested and Mutually Recursive Algorithms* (Giesl 97)
  - *Partial and Nested Recursive Function Definitions in Higher-Order Logic* (Krauss 09)
  - *Partiality and Recursion in Interactive Theorem Provers - An Overview* (Bove&Krauss&Sozeau 15)
- Constructive recursion:
  - *General recursion in TT* (Bove&Capretta 05)
  - the `Equations` package (2010 & 2019)
  - *The Braga method* (Types 2018 & WS 2021)
- Extraction related:
  - Extraction in Coq (P. Letousey's thesis 2004)
  - MetaCoq and Œuf (CPP'18)

The Braga method, an overview

- Techniques in Coq with standard tools:
  - implement spec while controlling CC
  - separate defs. from correctness proofs
  - non-terminating algo.
  - nested&mutual non-terminating algo
  - but no co-recursion

The Braga method, an overview

- Techniques in Coq with standard tools:
  - implement spec while controlling CC
  - separate defs. from correctness proofs
  - non-terminating algo.
  - nested&mutual non-terminating algo
  - but no co-recursion
- We do not use Coq extensions:
  - `Program Fixpoint` for measure induction
  - `Equations` (great to define)
  - not so great to control CC
  - but compatible with Braga

The Braga method, an overview

- Techniques in Coq with standard tools:
  - implement spec while controlling CC
  - separate defs. from correctness proofs
  - non-terminating algo.
  - nested&mutual non-terminating algo
  - but no co-recursion
- We do not use Coq extensions:
  - `Program Fixpoint` for measure induction
  - `Equations` (great to define)
  - not so great to control CC
  - but compatible with Braga
- Illustrated on examples:
  - ∞-loop, DFS, Paulson's normalisation
  - control of CC and separation of LC from CC

The induction principle for `False`

- The empty proposition `False`:

```
Inductive False : Prop := .
```

The induction principle for False

- The empty proposition False:

  $\texttt{Inductive False} : \texttt{Prop} := .$

- $\texttt{False\_rect} : \forall X : \texttt{Type}, \texttt{False} \to X$

  $$\texttt{Definition False\_rect}\ X\ (f : \texttt{False}) : X := \texttt{match}\ f\ \texttt{return}\ X\ \texttt{with end}.$$

The induction principle for `False`

- The empty proposition `False`:

  `Inductive False : Prop := .`

- `False_rect` : $\forall X$: `Type`, `False` $\to X$

  `Definition False_rect` X $(f$: `False`$)$: X :=
  `match` f `return` X `with end`.

- extracts to

  `let false_rect _ = assert false`

- interpretation of partiality: exception/error

The induction principle for False

- The empty proposition False:

  $\texttt{Inductive False} : \texttt{Prop} := .$

- $\texttt{False\_rect} : \forall X : \texttt{Type}, \texttt{False} \to X$

  $$\texttt{Definition False\_rect}\ X\ (f : \texttt{False}) : X := \texttt{match}\ f\ \texttt{return}\ X\ \texttt{with end}.$$

- extracts to

  $$\texttt{let false\_rect \_ = assert false}$$

- interpretation of partiality: exception/error
- is there another proof of False_rect?

Eliminating `False` with an infinite loop

- Looping on `False` with dummy `unit` argument:
- $\texttt{loop} : \forall X : \texttt{Type}, \texttt{unit} \to \texttt{False} \to X$

$$\texttt{fix loop}\ \{X\}\ (\_ : \texttt{unit})\ (\underline{f} : \texttt{False}) : X :=$$
$$\texttt{loop tt}\ (\texttt{match}\ f\ \texttt{return False with end}).$$

Eliminating False with an infinite loop

- Looping on False with dummy unit argument:
- $\texttt{loop} : \forall X : \texttt{Type}, \texttt{unit} \to \texttt{False} \to X$

$$\texttt{fix}\ \ \texttt{loop}\ \{X\}\ (\_ : \texttt{unit})\ (\underline{f} : \texttt{False}) : X :=$$
$$\texttt{loop tt}\ (\texttt{match}\ f\ \texttt{return False with end}).$$

- notice: $\texttt{fix loop}\ \{X\}\ \_\ \underline{f} := \texttt{loop tt}\ f$ **fails**

Eliminating False with an infinite loop

- Looping on `False` with dummy `unit` argument:
- $\texttt{loop} : \forall X : \texttt{Type}, \texttt{unit} \to \texttt{False} \to X$

  $\texttt{fix loop} \; \{X\} \; (\_ : \texttt{unit}) \; (\underline{f} : \texttt{False}) : X :=$
  $\quad \texttt{loop tt} \; (\texttt{match} \; f \; \texttt{return False with end}).$

- notice: $\texttt{fix loop} \; \{X\} \; \_ \; \underline{f} := \texttt{loop tt} \; f$ **fails**
- Alt. elim.: $\texttt{False\_loop} : \forall X : \texttt{Type}, \texttt{False} \to X$

  $\texttt{Definition False\_loop} \; X := \texttt{@loop} \; X \; \texttt{tt}$

Eliminating False with an infinite loop

- Looping on `False` with dummy `unit` argument:
- $\texttt{loop} : \forall X : \texttt{Type}, \texttt{unit} \to \texttt{False} \to X$

  $$\texttt{fix loop } \{X\}\ (\_ : \texttt{unit})\ (\underline{f} : \texttt{False}) : X := \texttt{loop tt}\ (\texttt{match } f \texttt{ return False with end}).$$

- notice: $\texttt{fix loop } \{X\}\ \_\ \underline{f} := \texttt{loop tt } f$ **fails**
- Alt. elim.: $\texttt{False\_loop} : \forall X : \texttt{Type}, \texttt{False} \to X$

  $$\texttt{Definition False\_loop } X := \texttt{@loop } X \texttt{ tt}$$

- extracts to

```
let false_loop _ =
  let rec loop _ = loop ()
  in loop ()
```

First take home idea

- Matching on `False`:

$$\boxed{\texttt{match}\ f : \texttt{False}\ \texttt{return}\ X\ \texttt{with end}}$$

First take home idea

- Matching on `False`:

$$\boxed{\texttt{match } f : \texttt{False return } X \texttt{ with end}}$$

- is a term of **any type** X

First take home idea

- Matching on `False`:

$$\boxed{\texttt{match}\ f : \texttt{False}\ \texttt{return}\ X\ \texttt{with end}}$$

- is a term of **any type** X
- is **structurally smaller than any term** in X
  - when X is an inductive type

First take home idea

- Matching on `False`:

$$\boxed{\texttt{match}\ f : \texttt{False}\ \texttt{return}\ X\ \texttt{with end}}$$

- is a term of **any type** X
- is **structurally smaller than any term** in X
  - when X is an inductive type
- used extensively to rule out absurd cases
  - the `exfalso` tactic
  - the `discriminate` tactic
  - the `destruct` tactic on $H : \ldots \to \ldots \to \texttt{False}$
  - absurd cases for `inversion`

Depth First Search

$\texttt{let rec } x \in^?_{\mathcal{V}} v =$
$\quad \texttt{match } v \texttt{ with}$
$\qquad \mid [\,] \quad\;\; \to \texttt{false}$
$\qquad \mid y :: w \to y = x \texttt{ or } x \in^?_{\mathcal{V}} w$

$\texttt{let rec dfs } v\ l =$
$\quad \texttt{match } l \texttt{ with}$
$\qquad \mid [\,] \quad\; \to v$
$\qquad \mid x :: l \to \texttt{if } x \in^?_{\mathcal{V}} v$
$\qquad\qquad\qquad \texttt{then dfs } v\ l$
$\qquad\qquad\qquad \texttt{else dfs } (x :: v)\ (\texttt{succs } x \texttt{ @ } l)$

Depth First Search

$$\begin{array}{l}
\texttt{let rec } x \in^?_{\mathcal{V}} v = \\
\quad \texttt{match } v \texttt{ with} \\
\qquad \mid [\,] \quad \rightarrow \texttt{false} \\
\qquad \mid y :: w \rightarrow y = x \texttt{ or } x \in^?_{\mathcal{V}} w
\end{array}$$

$$\begin{array}{l}
\texttt{let rec dfs } v\ l = \\
\quad \texttt{match } l \texttt{ with} \\
\qquad \mid [\,] \quad \rightarrow v \\
\qquad \mid x :: l \rightarrow \texttt{if } x \in^?_{\mathcal{V}} v \\
\qquad\qquad\qquad \texttt{then dfs } v\ l \\
\qquad\qquad\qquad \texttt{else dfs } (x :: v)\ (\texttt{succs } x\ @\ l)
\end{array}$$

- For $=^?_{\mathcal{V}} : \forall x\, y : \mathcal{V}, \{b \mid x = y \iff b = \texttt{true}\}$
- $\texttt{succs} : \mathcal{V} \rightarrow \texttt{list } \mathcal{V}$ (directed graph structure)

Depth First Search

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

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

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

From the algo. to its computational graph

- From the dfs algorithm only

$$
\begin{array}{l}
\texttt{let rec dfs } v\ l = \\
\quad \texttt{match } l \texttt{ with} \\
\qquad \mid [] \quad \rightarrow v \\
\qquad \mid x :: l \rightarrow \texttt{if } x \in^?_{\mathcal{V}} v \\
\qquad\qquad\qquad \texttt{then dfs } v\ l \\
\qquad\qquad\qquad \texttt{else dfs } (x :: v)\ (\texttt{succs } x\ \texttt{@}\ l)
\end{array}
$$

From the algo. to its computational graph

- From the dfs algorithm only

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

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

$$
\frac{}{\mathbb{G}_{\texttt{dfs}}\ v\ []\ v}
\qquad
\frac{x \in^?_{\mathcal{V}} v \quad \mathbb{G}_{\texttt{dfs}}\ v\ l\ o}{\mathbb{G}_{\texttt{dfs}}\ v\ (x :: l)\ o}
$$

$$
\frac{x \notin^?_{\mathcal{V}} v \quad \mathbb{G}_{\texttt{dfs}}\ (x :: v)\ (\texttt{succs}\ x +\!\!+\ l)\ o}{\mathbb{G}_{\texttt{dfs}}\ v\ (x :: l)\ o}
$$

From the algo. to its computational graph

- From the dfs algorithm only

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

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

$$\frac{}{\mathbb{G}_{\texttt{dfs}}\ v\ [\,]\ v} \qquad \frac{x \in^?_{\mathcal{V}} v \quad \mathbb{G}_{\texttt{dfs}}\ v\ l\ o}{\mathbb{G}_{\texttt{dfs}}\ v\ (x :: l)\ o}$$

$$\frac{x \notin^?_{\mathcal{V}} v \quad \mathbb{G}_{\texttt{dfs}}\ (x :: v)\ (\texttt{succs } x +\!\!+\, l)\ o}{\mathbb{G}_{\texttt{dfs}}\ v\ (x :: l)\ o}$$

- functional: $\mathbb{G}_{\texttt{dfs}}\ v\ l\ o_1 \to \mathbb{G}_{\texttt{dfs}}\ v\ l\ o_2 \to o_1 = o_2$.

From the graph to the domain predicate

- Graph $\mathbb{G}_{\texttt{dfs}}$

$$\frac{}{\mathbb{G}_{\texttt{dfs}}\ v\ []\ v} \qquad \frac{x \in^?_{\mathcal{V}} v \quad \mathbb{G}_{\texttt{dfs}}\ v\ l\ o}{\mathbb{G}_{\texttt{dfs}}\ v\ (x :: l)\ o}$$

$$\frac{x \notin^?_{\mathcal{V}} v \quad \mathbb{G}_{\texttt{dfs}}\ (x :: v)\ (\texttt{succs}\ x +\!\!+ l)\ o}{\mathbb{G}_{\texttt{dfs}}\ v\ (x :: l)\ o}$$

From the graph to the domain predicate

- Graph $\mathbb{G}_{\texttt{dfs}}$

$$\frac{}{\mathbb{G}_{\texttt{dfs}}\ v\ []\ v} \qquad \frac{x \in^?_{\mathcal{V}} v \quad \mathbb{G}_{\texttt{dfs}}\ v\ l\ o}{\mathbb{G}_{\texttt{dfs}}\ v\ (x :: l)\ o}$$

$$\frac{x \notin^?_{\mathcal{V}} v \quad \mathbb{G}_{\texttt{dfs}}\ (x :: v)\ (\texttt{succs}\ x +\!\!+\ l)\ o}{\mathbb{G}_{\texttt{dfs}}\ v\ (x :: l)\ o}$$

- We erase the output parameter!
  - i.e. we project on the first two parameters

From the graph to the domain predicate

- Graph $\mathbb{G}_{\mathtt{dfs}}$

$$\frac{}{\mathbb{G}_{\mathtt{dfs}}\ v\ []\ v} \qquad \frac{x \in^?_{\mathcal{V}} v \quad \mathbb{G}_{\mathtt{dfs}}\ v\ l\ o}{\mathbb{G}_{\mathtt{dfs}}\ v\ (x :: l)\ o}$$

$$\frac{x \notin^?_{\mathcal{V}} v \quad \mathbb{G}_{\mathtt{dfs}}\ (x :: v)\ (\mathtt{succs}\ x +\!\!+\ l)\ o}{\mathbb{G}_{\mathtt{dfs}}\ v\ (x :: l)\ o}$$

- We erase the output parameter!
  - i.e. we project on the first two parameters
- Domain $\mathbb{D}_{\mathtt{dfs}} : \mathtt{list}\ \mathcal{V} \to \mathtt{list}\ \mathcal{V} \to \mathtt{Prop}$

$$\frac{}{\mathbb{D}_{\mathtt{dfs}}\ v\ []}\ \langle \mathbb{D}^1_{\mathtt{dfs}} \rangle \qquad \frac{x \in^?_{\mathcal{V}} v \quad \mathbb{D}_{\mathtt{dfs}}\ v\ l}{\mathbb{D}_{\mathtt{dfs}}\ v\ (x :: l)}\ \langle \mathbb{D}^2_{\mathtt{dfs}} \rangle$$

$$\frac{x \notin^?_{\mathcal{V}} v \quad \mathbb{D}_{\mathtt{dfs}}\ (x :: v)\ (\mathtt{succs}\, x +\!\!+\ l)}{\mathbb{D}_{\mathtt{dfs}}\ v\ (x :: l)}\ \langle \mathbb{D}^3_{\mathtt{dfs}} \rangle$$

From the graph to the domain predicate

- Graph $\mathbb{G}_{\texttt{dfs}}$

$$\frac{}{\mathbb{G}_{\texttt{dfs}}\ v\ []\ v} \qquad \frac{x \in^?_{\mathcal{V}} v \quad \mathbb{G}_{\texttt{dfs}}\ v\ l\ o}{\mathbb{G}_{\texttt{dfs}}\ v\ (x :: l)\ o}$$

$$\frac{x \notin^?_{\mathcal{V}} v \quad \mathbb{G}_{\texttt{dfs}}\ (x :: v)\ (\texttt{succs}\ x +\!\!+\ l)\ o}{\mathbb{G}_{\texttt{dfs}}\ v\ (x :: l)\ o}$$

- We erase the output parameter!
  - i.e. we project on the first two parameters
- Domain $\mathbb{D}_{\texttt{dfs}} : \texttt{list}\ \mathcal{V} \to \texttt{list}\ \mathcal{V} \to \texttt{Prop}$

$$\frac{}{\mathbb{D}_{\texttt{dfs}}\ v\ []}\ \langle \mathbb{D}^1_{\texttt{dfs}} \rangle \qquad \frac{x \in^?_{\mathcal{V}} v \quad \mathbb{D}_{\texttt{dfs}}\ v\ l}{\mathbb{D}_{\texttt{dfs}}\ v\ (x :: l)}\ \langle \mathbb{D}^2_{\texttt{dfs}} \rangle$$

$$\frac{x \notin^?_{\mathcal{V}} v \quad \mathbb{D}_{\texttt{dfs}}\ (x :: v)\ (\texttt{succs}\, x +\!\!+\ l)}{\mathbb{D}_{\texttt{dfs}}\ v\ (x :: l)}\ \langle \mathbb{D}^3_{\texttt{dfs}} \rangle$$

- We will show: $\mathbb{D}_{\texttt{dfs}}\ v\ l \iff \exists o,\ \mathbb{G}_{\texttt{dfs}}\ v\ l\ o$

DFS packed with conformity to $\mathbb{G}_{\texttt{dfs}}$

- $\texttt{dfs\_pwc} : \forall v\, l, \mathbb{D}_{\texttt{dfs}}\ v\ l \to \{o \mid \mathbb{G}_{\texttt{dfs}}\ v\ l\ o\}$
- By structural induction on the domain predicate D

```
Fixpoint dfs_pwc v l (D : 𝔻dfs v l) : {o | 𝔾dfs v l o}.
Proof. refine(
  match l with
    | []     ⇒ λ D, exist _ v 𝒪₁?
    | x :: l ⇒ λ D,
    match x ∈?𝒱 v as b return x ∈?𝒱 v = b → _ with
      | true  ⇒ λ E,
            let (o, Go) := dfs_pwc v l 𝒯₂?
            in  exist _ o 𝒪₂?
      | false ⇒ λ E,
            let (o, Go) := dfs_pwc (x :: v) (succs x ++ l) 𝒯₃?
            in  exist _ o 𝒪₃?
    end eq_refl
  end D).
  (* Proof obligations *)
Qed.
```

- But not by pattern matching on D!

Proof obligations: postconditions $\mathcal{O}^?_{1,2,3}$

- Postcondition e.g. $\mathcal{O}^?_2$

  $[\mathcal{O}^?_2] : \ldots, E : x \in^?_{\mathcal{V}} v = \texttt{true}, G_o : \mathbb{G}_{\texttt{dfs}}\ v\ l\ o \vdash \mathbb{G}_{\texttt{dfs}}\ v\ (x{::}l)\ o$

Proof obligations: postconditions $\mathcal{O}^?_{1,2,3}$

- Postcondition e.g. $\mathcal{O}^?_2$

  $[\mathcal{O}^?_2] : \ \ldots, E : x \in^?_{\mathcal{V}} v = \mathtt{true}, G_o : \mathbb{G}_{\mathtt{dfs}}\ v\ l\ o \vdash \mathbb{G}_{\mathtt{dfs}}\ v\ (x{::}l)\ o$

- is trivial to handle
- second constructor of the graph $\mathbb{G}_{\mathtt{dfs}}$:

$$\frac{x \in^?_{\mathcal{V}} v \qquad \mathbb{G}_{\mathtt{dfs}}\ v\ l\ o}{\mathbb{G}_{\mathtt{dfs}}\ v\ (x :: l)\ o}$$

Proof obligations: postconditions $\mathcal{O}^?_{1,2,3}$

- Postcondition e.g. $\mathcal{O}^?_2$

  $[\mathcal{O}^?_2] : \ \ldots, E : x \in^?_{\mathcal{V}} v = \texttt{true}, G_o : \mathbb{G}_{\texttt{dfs}}\ v\ l\ o \vdash \mathbb{G}_{\texttt{dfs}}\ v\ (x{::}l)\ o$

- is trivial to handle
- second constructor of the graph $\mathbb{G}_{\texttt{dfs}}$:

$$\frac{x \in^?_{\mathcal{V}} v \qquad \mathbb{G}_{\texttt{dfs}}\ v\ l\ o}{\mathbb{G}_{\texttt{dfs}}\ v\ (x :: l)\ o}$$

- same holds for $\mathcal{O}^?_1$ and $\mathcal{O}^?_3$

Proof obligations: postconditions $\mathcal{O}^?_{1,2,3}$

- Postcondition e.g. $\mathcal{O}^?_2$

  $[\mathcal{O}^?_2] : \ldots, E : x \in^?_{\mathcal{V}} v = \mathtt{true}, G_o : \mathbb{G}_{\mathtt{dfs}}\ v\ l\ o \vdash \mathbb{G}_{\mathtt{dfs}}\ v\ (x{::}l)\ o$

- is trivial to handle
- second constructor of the graph $\mathbb{G}_{\mathtt{dfs}}$:

$$\frac{x \in^?_{\mathcal{V}} v \qquad \mathbb{G}_{\mathtt{dfs}}\ v\ l\ o}{\mathbb{G}_{\mathtt{dfs}}\ v\ (x :: l)\ o}$$

- same holds for $\mathcal{O}^?_1$ and $\mathcal{O}^?_3$
- Termination certificates $\mathcal{T}^?_2$ and $\mathcal{T}^?_3$:
  - much more complicated to handle

Proof obligations: termination certificates

- Termination certificates $\mathcal{T}_2^?$ and $\mathcal{T}_3^?$

$$[\mathcal{T}_2^?] : \ \ldots, D : \mathbb{D}_{\texttt{dfs}}\ v\ (x :: l), E : x \in^?_{\mathcal{V}} v = \texttt{true} \vdash \mathbb{D}_{\texttt{dfs}}\ v\ l$$

Proof obligations: termination certificates

- Termination certificates $\mathcal{T}_2^?$ and $\mathcal{T}_3^?$

$$[\mathcal{T}_2^?] : \ \ldots, D : \mathbb{D}_{\texttt{dfs}}\ v\ (x :: l), E : x \in_{\mathcal{V}}^? v = \texttt{true} \vdash \mathbb{D}_{\texttt{dfs}}\ v\ l$$

- We need to provide a term of type:

$$\pi_{\mathbb{D}_{\texttt{dfs}}}\_2 : \forall v\, x\, l,\ \mathbb{D}_{\texttt{dfs}}\ v\ (x :: l) \to x \in_{\mathcal{V}}^? v = \texttt{true} \to \mathbb{D}_{\texttt{dfs}}\ v\ l$$

Proof obligations: termination certificates

- Termination certificates $\mathcal{T}_2^?$ and $\mathcal{T}_3^?$

  $[\mathcal{T}_2^?] : \ldots, D : \mathbb{D}_{\texttt{dfs}}\ v\ (x :: l), E : x \in_{\mathcal{V}}^? v = \texttt{true} \vdash \mathbb{D}_{\texttt{dfs}}\ v\ l$

- We need to provide a term of type:

  $\pi_{\mathbb{D}_{\texttt{dfs}}}\_2 : \forall v\, x\, l,\ \mathbb{D}_{\texttt{dfs}}\ v\ (x :: l) \to x \in_{\mathcal{V}}^? v = \texttt{true} \to \mathbb{D}_{\texttt{dfs}}\ v\ l$

- The projection $\pi_{\mathbb{D}_{\texttt{dfs}}}\_2$ inverts the constructor $\mathbb{D}^2_{\texttt{dfs}}$:

$$\frac{x \in_{\mathcal{V}}^? v \qquad \mathbb{D}_{\texttt{dfs}}\ v\ l}{\mathbb{D}_{\texttt{dfs}}\ v\ (x :: l)}\ \langle \mathbb{D}^2_{\texttt{dfs}} \rangle \qquad \frac{x \in_{\mathcal{V}}^? v \qquad \mathbb{D}_{\texttt{dfs}}\ v\ (x :: l)}{\mathbb{D}_{\texttt{dfs}}\ v\ l}\ \langle \pi_{\mathbb{D}_{\texttt{dfs}}}\_2 \rangle$$

- `Fixpoint` guard cond.: $\pi_{\mathbb{D}_{\texttt{dfs}}}\_2\ v\ x\ l\ D\ E <_{\text{struct}} D$
  - `inversion` tactic works but unreadable
  - Small inversions give a human checkable term

Small Inversions (J.F. Monin)

- Termination certificate $\mathcal{T}_2^?$ by dep. pattern matching
- Generic code which explicits structural decrease

```
Let shape (b : bool) v l :=
  match l with
    | []     ⇒ False
    | x :: l ⇒ x ∈?_𝒱 v = b
  end.

Let p_tl {b v l} : shape b v l → list 𝒱 :=
  match l with
    | []     ⇒ λ s, match s : False with end
    | _ :: l ⇒ λ _, l
  end.

Let π_𝔻dfs_2_gen {v l} (D_vl : 𝔻dfs v l) : ∀s : shape true v l, 𝔻dfs v (p_tl s) :=
  match D_vl in 𝔻dfs v′ l′ with
  return ∀s : shape true v′ l′, 𝔻dfs v′ (p_tl s)
  with
    | 𝔻¹dfs v          ⇒ λ s, match s : False with end
    | 𝔻²dfs v x l _ D  ⇒ λ _, D
    | 𝔻³dfs v x l H _  ⇒ λ s, match not_mem_true H s : False with end
  end.

Let π_𝔻dfs_2 v x l : 𝔻dfs v (x :: l) → x ∈?_𝒱 v = true → 𝔻dfs v l := π_𝔻dfs_2_gen.
```

The partial DFS algorithm

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

The partial DFS algorithm

- $\texttt{dfs\_pwc} : \forall v\, l, \mathbb{D}_{\texttt{dfs}}\ v\ l \to \{o \mid \mathbb{G}_{\texttt{dfs}}\ v\ l\ o\}$
- We define $\texttt{dfs}\ v\ l\ D := \pi_1(\texttt{dfs\_pwc}\ v\ l\ D)$

The partial DFS algorithm

- $\texttt{dfs\_pwc} : \forall v\, l, \mathbb{D}_{\texttt{dfs}}\ v\ l \to \{o \mid \mathbb{G}_{\texttt{dfs}}\ v\ l\ o\}$
- We define $\texttt{dfs}\ v\ l\ D := \pi_1(\texttt{dfs\_pwc}\ v\ l\ D)$
- get conformity via π_2 (low-level):

$$\boxed{\texttt{dfs\_spec} : \forall v\, l\, D,\ \mathbb{G}_{\texttt{dfs}}\ v\ l\ (\texttt{dfs}\ v\ l\ D)}$$

- Then fixpoint eqs and proof irrelevance

$$\begin{array}{l}
\texttt{dfs_pirr}\ \ : \texttt{dfs}\ v\ l\ D_1 = \texttt{dfs}\ v\ l\ D_2. \\
\texttt{dfs_fix_1} : \texttt{dfs}\ v\ []\ (\mathbb{D}^1_{\texttt{dfs}}\ v) = v. \\
\texttt{dfs_fix_2} : \texttt{dfs}\ v\ (x :: l)\ (\mathbb{D}^2_{\texttt{dfs}}\ v\ x\ l\ H\, D) = \texttt{dfs}\ v\ l\ D. \\
\texttt{dfs_fix_3} : \texttt{dfs}\ v\ (x :: l)\ (\mathbb{D}^3_{\texttt{dfs}}\ v\ x\ l\ H\, D) = \texttt{dfs}\ (x :: v)\ (\texttt{succs}\, x +\!\!+ l)\ D.
\end{array}$$

The partial DFS algorithm

- $\texttt{dfs\_pwc} : \forall v\, l, \mathbb{D}_{\texttt{dfs}}\ v\ l \to \{o \mid \mathbb{G}_{\texttt{dfs}}\ v\ l\ o\}$
- We define $\texttt{dfs}\ v\ l\ D := \pi_1(\texttt{dfs\_pwc}\ v\ l\ D)$
- get conformity via π_2 (low-level):

$$\boxed{\texttt{dfs\_spec} : \forall v\, l\, D,\ \mathbb{G}_{\texttt{dfs}}\ v\ l\ (\texttt{dfs}\ v\ l\ D)}$$

- Then fixpoint eqs and proof irrelevance

$$\begin{array}{l}
\texttt{dfs_pirr}\ \ : \texttt{dfs}\ v\ l\ D_1 = \texttt{dfs}\ v\ l\ D_2. \\
\texttt{dfs_fix_1} : \texttt{dfs}\ v\ []\ (\mathbb{D}^1_{\texttt{dfs}}\ v) = v. \\
\texttt{dfs_fix_2} : \texttt{dfs}\ v\ (x :: l)\ (\mathbb{D}^2_{\texttt{dfs}}\ v\ x\ l\ H\, D) = \texttt{dfs}\ v\ l\ D. \\
\texttt{dfs_fix_3} : \texttt{dfs}\ v\ (x :: l)\ (\mathbb{D}^3_{\texttt{dfs}}\ v\ x\ l\ H\, D) = \texttt{dfs}\ (x :: v)\ (\texttt{succs}\, x +\!\!+\, l)\ D.
\end{array}$$

- $\mathbb{D}_{\texttt{dfs}}$ has a dependent recursion principle

$$\begin{array}{l}
\texttt{Theorem}\ \mathbb{D}_{\texttt{dfs}}\texttt{_rect}\ (P : \forall v\, l,\ \mathbb{D}_{\texttt{dfs}}\, v\, l \to \texttt{Type}) : \\
\quad\quad (\forall v\, l\, D_1\, D_2,\ P\ v\ l\ D_1 \to P\ v\ l\ D_2) \\
\quad \to (\forall v,\ P\ _\ _\ (\mathbb{D}^1_{\texttt{dfs}}\ v)) \\
\quad \to (\forall v\, x\, l\, H D,\ P\ _\ _\ D \to P\ _\ _\ (\mathbb{D}^2_{\texttt{dfs}}\ v\ x\ l\ H\ D)) \\
\quad \to (\forall v\, x\, l\, H D,\ P\ _\ _\ D \to P\ _\ _\ (\mathbb{D}^3_{\texttt{dfs}}\ v\ x\ l\ H\ D)) \\
\quad \to (\forall v\, l\, D, P\ v\ l\ D).
\end{array}$$

Simulated Inductive-Recursive Scheme

- Thus we simulate the IR-scheme (Dybjer 2000)

$$
\begin{array}{l}
\texttt{Inductive}\ \mathbb{D}_{\texttt{dfs}} : \texttt{list}\ \mathcal{V} \to \texttt{list}\ \mathcal{V} \to \boxed{\texttt{Prop}} := \\
\quad \mid \mathbb{D}^1_{\texttt{dfs}} : \forall v, \qquad\qquad\qquad\quad \mathbb{D}_{\texttt{dfs}}\ v\ [] \\
\quad \mid \mathbb{D}^2_{\texttt{dfs}} : \forall v\, x\, l,\ x \in^?_{\mathcal{V}} v \to \mathbb{D}_{\texttt{dfs}}\ v\ l \\
\qquad\qquad\qquad\qquad\qquad\quad \to \mathbb{D}_{\texttt{dfs}}\ v\ (x :: l) \\
\quad \mid \mathbb{D}^3_{\texttt{dfs}} : \forall v\, x\, l,\ x \notin^?_{\mathcal{V}} v \to \mathbb{D}_{\texttt{dfs}}\ (x :: v)\ (\texttt{succs}\ x +\!\!+\, l) \\
\qquad\qquad\qquad\qquad\qquad\quad \to \mathbb{D}_{\texttt{dfs}}\ v\ (x :: l)
\end{array}
$$

$$
\begin{array}{l}
\texttt{with Fixpoint dfs}\ v\ l\ (D : \mathbb{D}_{\texttt{dfs}}\ v\ l) : \texttt{list}\ \mathcal{V} := \\
\quad \texttt{match}\ D\ \texttt{with} \\
\quad\quad \mid \mathbb{D}^1_{\texttt{dfs}}\ v \qquad\quad \Rightarrow v \\
\quad\quad \mid \mathbb{D}^2_{\texttt{dfs}}\ v\ x\ l\ \_\ D \Rightarrow \texttt{dfs}\ v\ l\ D \\
\quad\quad \mid \mathbb{D}^3_{\texttt{dfs}}\ v\ x\ l\ \_\ D \Rightarrow \texttt{dfs}\ (x :: v)\ (\texttt{succs}\ x +\!\!+\, l)\ D \\
\quad \texttt{end}
\end{array}
$$

Simulated Inductive-Recursive Scheme

- Thus we simulate the IR-scheme (Dybjer 2000)

$$
\begin{array}{l}
\texttt{Inductive}\ \mathbb{D}_{\texttt{dfs}} : \texttt{list}\ \mathcal{V} \to \texttt{list}\ \mathcal{V} \to \texttt{Prop} := \\
\quad \mid \mathbb{D}^1_{\texttt{dfs}} : \forall v, \qquad \mathbb{D}_{\texttt{dfs}}\ v\ [] \\
\quad \mid \mathbb{D}^2_{\texttt{dfs}} : \forall v\, x\, l,\ x \in^?_{\mathcal{V}} v \to \mathbb{D}_{\texttt{dfs}}\ v\ l \\
\qquad\qquad \to \mathbb{D}_{\texttt{dfs}}\ v\ (x :: l) \\
\quad \mid \mathbb{D}^3_{\texttt{dfs}} : \forall v\, x\, l,\ x \notin^?_{\mathcal{V}} v \to \mathbb{D}_{\texttt{dfs}}\ (x :: v)\ (\texttt{succs}\ x +\!\!+\ l) \\
\qquad\qquad \to \mathbb{D}_{\texttt{dfs}}\ v\ (x :: l) \\
\texttt{with Fixpoint dfs}\ v\ l\ (D : \mathbb{D}_{\texttt{dfs}}\ v\ l) : \texttt{list}\ \mathcal{V} := \\
\quad \texttt{match}\ D\ \texttt{with} \\
\quad \mid \mathbb{D}^1_{\texttt{dfs}}\ v \qquad \Rightarrow v \\
\quad \mid \mathbb{D}^2_{\texttt{dfs}}\ v\ x\ l\ \_\ D \Rightarrow \texttt{dfs}\ v\ l\ D \\
\quad \mid \mathbb{D}^3_{\texttt{dfs}}\ v\ x\ l\ \_\ D \Rightarrow \texttt{dfs}\ (x :: v)\ (\texttt{succs}\ x +\!\!+\ l)\ D \\
\quad \texttt{end}
\end{array}
$$

- Degenerate here because no nesting

High-level partial correctness for DFS

- Partial correctness by induction on $\mathbb{D}_{\mathtt{dfs}}\ v\ l$
  - when `dfs` terminates
  - it computes a minimal invariant for `succs`

High-level partial correctness for DFS

- Partial correctness by induction on $\mathbb{D}_{\texttt{dfs}}\ v\ l$
  - when `dfs` terminates
  - it computes a minimal invariant for `succs`

$$\texttt{Definition dfs\_invariant}_t\ (v\ l\ i : \texttt{list}\ \mathcal{V}) :=$$
$$v \subseteq i \wedge l \subseteq i \wedge (\forall x,\ x \in^?_{\mathcal{V}} i \to x \in^?_{\mathcal{V}} v \vee \texttt{succs}\ x \subseteq i).$$

$$\texttt{Theorem dfs\_invariant}\ v\ l\ (D : \mathbb{D}_{\texttt{dfs}}\ v\ l) :$$
$$\wedge \begin{cases} \texttt{dfs\_invariant}_t\ v\ l\ (\texttt{dfs}\ v\ l\ D) \\ \forall i,\ \texttt{dfs\_invariant}_t\ v\ l\ i \to \texttt{dfs}\ v\ l\ D \subseteq i. \end{cases}$$

High-level partial correctness for DFS

- Partial correctness by induction on $\mathbb{D}_{\texttt{dfs}}\, v\, l$
  - when `dfs` terminates
  - it computes a minimal invariant for `succs`

$$\texttt{Definition dfs\_invariant}_t\ (v\ l\ i : \texttt{list}\,\mathcal{V}) :=$$
$$v \subseteq i \wedge l \subseteq i \wedge (\forall x, x \in^?_{\mathcal{V}} i \to x \in^?_{\mathcal{V}} v \vee \texttt{succs}\, x \subseteq i).$$

$$\texttt{Theorem dfs\_invariant}\ v\ l\ (D : \mathbb{D}_{\texttt{dfs}}\, v\, l) :$$
$$\wedge \begin{cases} \texttt{dfs\_invariant}_t\ v\ l\ (\texttt{dfs}\ v\ l\ D) \\ \forall i, \texttt{dfs\_invariant}_t\ v\ l\ i \to \texttt{dfs}\ v\ l\ D \subseteq i. \end{cases}$$

- We can characterize termination (harder)
  - when there is an invariant
  - then `dfs` terminates

High-level partial correctness for DFS

- Partial correctness by induction on $\mathbb{D}_{\texttt{dfs}}\, v\, l$
  - when `dfs` terminates
  - it computes a minimal invariant for `succs`

$$\texttt{Definition dfs\_invariant}_t\ (v\ l\ i : \texttt{list}\ \mathcal{V}) :=$$
$$v \subseteq i \wedge l \subseteq i \wedge (\forall x,\ x \in^?_{\mathcal{V}} i \to x \in^?_{\mathcal{V}} v \vee \texttt{succs}\ x \subseteq i).$$

$$\texttt{Theorem dfs\_invariant}\ v\ l\ (D : \mathbb{D}_{\texttt{dfs}}\ v\ l) :$$
$$\wedge \begin{cases} \texttt{dfs\_invariant}_t\ v\ l\ (\texttt{dfs}\ v\ l\ D) \\ \forall i, \texttt{dfs\_invariant}_t\ v\ l\ i \to \texttt{dfs}\ v\ l\ D \subseteq i. \end{cases}$$

- We can characterize termination (harder)
  - when there is an invariant
  - then `dfs` terminates

$$\texttt{Theorem}\ \mathbb{D}_{\texttt{dfs}}\texttt{\_domain}\ v\ l :$$
$$\mathbb{D}_{\texttt{dfs}}\ v\ l \iff \exists i, \texttt{dfs\_invariant}_t\ v\ l\ i.$$

Second take home ideas

- From the computational graph $\mathbb{G}_\varphi$
- We derive the inductive domain $\mathbb{D}_\varphi$

Second take home ideas

- From the computational graph $\mathbb{G}_\varphi$
- We derive the inductive domain $\mathbb{D}_\varphi$
  - by projecting on the input values
  - in every rule defining $\mathbb{G}_\varphi$

Second take home ideas

- From the computational graph $\mathbb{G}_\varphi$
- We derive the inductive domain $\mathbb{D}_\varphi$
  - by projecting on the input values
  - in every rule defining $\mathbb{G}_\varphi$
- No high-level knowledge of φ needed
  - Termination is not needed for partial correctness
  - Partial correctness could be use for termination

Second take home ideas

- From the computational graph $\mathbb{G}_\varphi$
- We derive the inductive domain $\mathbb{D}_\varphi$
  - by projecting on the input values
  - in every rule defining $\mathbb{G}_\varphi$
- No high-level knowledge of φ needed
  - Termination is not needed for partial correctness
  - Partial correctness could be use for termination
- Beware with nested algos. (see later)
  - Projecting the graph a bit more complicated

Larry Paulson's normalization (1985)

$$
\begin{array}{l}
\texttt{let rec nm } e = \texttt{match } e \texttt{ with} \\
\quad \mid \alpha \qquad\qquad\qquad\quad \to \alpha \\
\quad \mid \omega(\alpha, y, z) \qquad\quad \to \omega(\alpha, \texttt{nm } y, \texttt{nm } z) \\
\quad \mid \omega(\omega(a, b, c), y, z) \to \texttt{nm}(\omega(a, \texttt{nm}(\omega(b, y, z)), \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad \texttt{nm}(\omega(c, y, z))))
\end{array}
$$

- Expressions in Ω : $b, x, y ::= \alpha \mid \omega\ b\ x\ y$
  - α is atomic expression
  - $\omega\ b\ x\ y$ denotes "`if` b `then` x `else` y"

Larry Paulson's normalization (1985)

$$
\begin{array}{l}
\texttt{let rec nm } e = \texttt{match } e \texttt{ with} \\
\quad \mid \alpha \qquad\qquad\qquad\quad \to \alpha \\
\quad \mid \omega(\alpha, y, z) \qquad\quad\, \to \omega(\alpha, \texttt{nm } y, \texttt{nm } z) \\
\quad \mid \omega(\omega(a,b,c),y,z) \to \texttt{nm}(\omega(a, \texttt{nm}(\omega(b,y,z)), \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad \texttt{nm}(\omega(c,y,z))))
\end{array}
$$

- Expressions in Ω : $b, x, y ::= \alpha \mid \omega\ b\ x\ y$
  - α is atomic expression
  - $\omega\ b\ x\ y$ denotes "if b then x else y"
- Interest of this algorithm:
  - recurring example (Giesl 97, B&C 05...)
  - has nested recursion but still compact
  - idealized but meaningful

Inductive capture of $\mathbb{D}_{\mathtt{nm}} : \Omega \to \mathtt{Prop}$

- Using the computational graph $\mathbb{G}_{\mathtt{nm}} : \Omega \to \Omega \to \mathtt{Prop}$

$$\frac{}{\mathbb{G}_{\mathtt{nm}}\ \alpha\ \alpha} \qquad \frac{\mathbb{G}_{\mathtt{nm}}\ y\ n_y \qquad \mathbb{G}_{\mathtt{nm}}\ z\ n_z}{\mathbb{G}_{\mathtt{nm}}\ (\omega\ \alpha\ y\ z)\ (\omega\ \alpha\ n_y\ n_z)}$$

$$\frac{\mathbb{G}_{\mathtt{nm}}\ (\omega\ b\ y\ z)\ n_b \qquad \mathbb{G}_{\mathtt{nm}}\ (\omega\ c\ y\ z)\ n_c \qquad \mathbb{G}_{\mathtt{nm}}\ (\omega\ a\ n_b\ n_c)\ n_a}{\mathbb{G}_{\mathtt{nm}}\ (\omega\ (\omega\ a\ b\ c)\ y\ z)\ n_a}$$

Inductive capture of $\mathbb{D}_{\texttt{nm}} : \Omega \to \texttt{Prop}$

- Using the computational graph $\mathbb{G}_{\texttt{nm}} : \Omega \to \Omega \to \texttt{Prop}$

$$\frac{}{\mathbb{G}_{\texttt{nm}}\ \alpha\ \alpha} \qquad \frac{\mathbb{G}_{\texttt{nm}}\ y\ n_y \qquad \mathbb{G}_{\texttt{nm}}\ z\ n_z}{\mathbb{G}_{\texttt{nm}}\ (\omega\ \alpha\ y\ z)\ (\omega\ \alpha\ n_y\ n_z)}$$

$$\frac{\mathbb{G}_{\texttt{nm}}\ (\omega\ b\ y\ z)\ n_b \qquad \mathbb{G}_{\texttt{nm}}\ (\omega\ c\ y\ z)\ n_c \qquad \mathbb{G}_{\texttt{nm}}\ (\omega\ a\ n_b\ n_c)\ n_a}{\mathbb{G}_{\texttt{nm}}\ (\omega\ (\omega\ a\ b\ c)\ y\ z)\ n_a}$$

- Define $\mathbb{D}_{\texttt{nm}} \simeq \lambda\, e, \exists n, \mathbb{G}_{\texttt{nm}}\, e\, n$ inductively by:

$$\frac{}{\mathbb{D}_{\texttt{nm}}\ \alpha} \qquad \frac{\mathbb{D}_{\texttt{nm}}\ y \qquad \mathbb{D}_{\texttt{nm}}\ z}{\mathbb{D}_{\texttt{nm}}\ (\omega\ \alpha\ y\ z)}$$

$$\frac{\begin{array}{c}\mathbb{D}_{\texttt{nm}}\ (\omega\ b\ y\ z) \qquad \mathbb{D}_{\texttt{nm}}\ (\omega\ c\ y\ z) \\ \forall n_b\, n_c, \mathbb{G}_{\texttt{nm}}\ (\omega\ b\ y\ z)\ n_b \to \mathbb{G}_{\texttt{nm}}\ (\omega\ c\ y\ z)\ n_c \to \mathbb{D}_{\texttt{nm}}\ (\omega\ a\ n_b\ n_c)\end{array}}{\mathbb{D}_{\texttt{nm}}\ (\omega\ (\omega\ a\ b\ c)\ y\ z)}$$

Inductive capture of $\mathbb{D}_{\mathtt{nm}} : \Omega \to \mathtt{Prop}$

- Using the computational graph $\mathbb{G}_{\mathtt{nm}} : \Omega \to \Omega \to \mathtt{Prop}$

$$\frac{}{\mathbb{G}_{\mathtt{nm}}\ \alpha\ \alpha} \qquad \frac{\mathbb{G}_{\mathtt{nm}}\ y\ n_y \qquad \mathbb{G}_{\mathtt{nm}}\ z\ n_z}{\mathbb{G}_{\mathtt{nm}}\ (\omega\ \alpha\ y\ z)\ (\omega\ \alpha\ n_y\ n_z)}$$

$$\frac{\mathbb{G}_{\mathtt{nm}}\ (\omega\ b\ y\ z)\ n_b \qquad \mathbb{G}_{\mathtt{nm}}\ (\omega\ c\ y\ z)\ n_c \qquad \mathbb{G}_{\mathtt{nm}}\ (\omega\ a\ n_b\ n_c)\ n_a}{\mathbb{G}_{\mathtt{nm}}\ (\omega\ (\omega\ a\ b\ c)\ y\ z)\ n_a}$$

- Define $\mathbb{D}_{\mathtt{nm}} \simeq \lambda\, e, \exists n, \mathbb{G}_{\mathtt{nm}}\, e\, n$ inductively by:

$$\frac{}{\mathbb{D}_{\mathtt{nm}}\ \alpha} \qquad \frac{\mathbb{D}_{\mathtt{nm}}\ y \qquad \mathbb{D}_{\mathtt{nm}}\ z}{\mathbb{D}_{\mathtt{nm}}\ (\omega\ \alpha\ y\ z)}$$

$$\frac{\begin{array}{c}\mathbb{D}_{\mathtt{nm}}\ (\omega\ b\ y\ z) \qquad \mathbb{D}_{\mathtt{nm}}\ (\omega\ c\ y\ z) \\ \forall n_b\, n_c, \mathbb{G}_{\mathtt{nm}}\ (\omega\ b\ y\ z)\ n_b \to \mathbb{G}_{\mathtt{nm}}\ (\omega\ c\ y\ z)\ n_c \to \mathbb{D}_{\mathtt{nm}}\ (\omega\ a\ n_b\ n_c)\end{array}}{\mathbb{D}_{\mathtt{nm}}\ (\omega\ (\omega\ a\ b\ c)\ y\ z)}$$

- The rules for $\mathbb{D}_{\mathtt{nm}}$ use $\mathbb{G}_{\mathtt{nm}}$ **for nested calls**

Def. $\texttt{nm\_pwc} : \forall e,\ \mathbb{D}_{\texttt{nm}}\ e \to \{n \mid \mathbb{G}_{\texttt{nm}}\ e\ n\}$

$$
\begin{array}{l}
\texttt{Fixpoint nm\_pwc}\ e\ (\underline{D} : \mathbb{D}_{\texttt{nm}}\ e) : \{n \mid \mathbb{G}_{\texttt{nm}}\ e\ n\}. \\
\texttt{refine(} \\
\quad \texttt{match}\ e\ \texttt{as}\ e'\ \texttt{return}\ \mathbb{D}_{\texttt{nm}}\ e' \to \{n \mid \mathbb{G}_{\texttt{nm}}\ e'\ n\}\ \texttt{with} \\
\quad\quad \mid \alpha \quad \Rightarrow \lambda D,\ \texttt{exist}\ \_\ \alpha\ \mathcal{O}_0^? \\
\quad\quad \mid \omega\ \alpha\ y\ z \quad \Rightarrow \lambda D,\ \texttt{let}\ (n_y, C_y) := \texttt{nm\_pwc}\ y\ \mathcal{T}_y^?\ \texttt{in} \\
\quad\quad\quad\quad \texttt{let}\ (n_z, C_z) := \texttt{nm\_pwc}\ z\ \mathcal{T}_z^? \\
\quad\quad\quad\quad \texttt{in}\ \ \texttt{exist}\ \_\ (\omega\ \alpha\ n_y\ n_z)\ \mathcal{O}_1^? \\
\quad\quad \mid \omega\ (\omega\ a\ b\ c)\ y\ z \quad \Rightarrow \lambda D,\ \texttt{let}\ (n_b, C_b) := \texttt{nm\_pwc}\ (\omega\ b\ y\ z)\ \mathcal{T}_b^?\ \texttt{in} \\
\quad\quad\quad\quad \texttt{let}\ (n_c, C_c) := \texttt{nm\_pwc}\ (\omega\ c\ y\ z)\ \mathcal{T}_c^?\ \texttt{in} \\
\quad\quad\quad\quad \texttt{let}\ (n_a, C_a) := \texttt{nm\_pwc}\ (\omega\ a\ n_b\ n_c)\ \mathcal{T}_a^? \\
\quad\quad\quad\quad \texttt{in}\ \ \texttt{exist}\ \_\ n_a\ \mathcal{O}_2^? \\
\quad \texttt{end}\ D\texttt{);}\ \texttt{simpl in}\ *. \\
\texttt{Proof.}\ \text{of certificates } \mathcal{T}_y^?, \mathcal{T}_z^?, \mathcal{T}_b^?, \mathcal{T}_c^?, \mathcal{T}_a^? \text{ and post-conditions } \mathcal{O}_0^?, \mathcal{O}_1^?, \mathcal{O}_2^?\ \ \texttt{Qed.}
\end{array}
$$

Def. nm_pwc : $\forall e,\ \mathbb{D}_{\mathtt{nm}}\ e \to \{n \mid \mathbb{G}_{\mathtt{nm}}\ e\ n\}$

$$
\begin{array}{l}
\texttt{Fixpoint nm\_pwc}\ e\ (\underline{D} : \mathbb{D}_{\mathtt{nm}}\ e) : \{n \mid \mathbb{G}_{\mathtt{nm}}\ e\ n\}.\\
\texttt{refine(}\\
\quad \texttt{match}\ e\ \texttt{as}\ e'\ \texttt{return}\ \mathbb{D}_{\mathtt{nm}}\ e' \to \{n \mid \mathbb{G}_{\mathtt{nm}}\ e'\ n\}\ \texttt{with}\\
\quad\ \mid \alpha \qquad\qquad\qquad \Rightarrow \lambda D, \qquad \texttt{exist}\ \_\ \alpha\ \mathcal{O}_0^?\\
\quad\ \mid \omega\ \alpha\ y\ z \qquad\quad\ \Rightarrow \lambda D,\ \texttt{let}\ (n_y, C_y) := \texttt{nm\_pwc}\ y\ \mathcal{T}_y^?\ \texttt{in}\\
\qquad\qquad\qquad\qquad\qquad\qquad\ \texttt{let}\ (n_z, C_z) := \texttt{nm\_pwc}\ z\ \mathcal{T}_z^?\\
\qquad\qquad\qquad\qquad\qquad\qquad\ \texttt{in}\ \ \texttt{exist}\ \_\ (\omega\ \alpha\ n_y\ n_z)\ \mathcal{O}_1^?\\
\quad\ \mid \omega\ (\omega\ a\ b\ c)\ y\ z \Rightarrow \lambda D,\ \texttt{let}\ (n_b, C_b) := \texttt{nm\_pwc}\ (\omega\ b\ y\ z)\ \mathcal{T}_b^?\ \texttt{in}\\
\qquad\qquad\qquad\qquad\qquad\qquad\ \texttt{let}\ (n_c, C_c) := \texttt{nm\_pwc}\ (\omega\ c\ y\ z)\ \mathcal{T}_c^?\ \texttt{in}\\
\qquad\qquad\qquad\qquad\qquad\qquad\ \texttt{let}\ (n_a, C_a) := \texttt{nm\_pwc}\ (\omega\ a\ n_b\ n_c)\ \mathcal{T}_a^?\\
\qquad\qquad\qquad\qquad\qquad\qquad\ \texttt{in}\ \ \texttt{exist}\ \_\ n_a\ \mathcal{O}_2^?\\
\quad \texttt{end}\ D\texttt{);}\ \texttt{simpl in}\ *.\\
\texttt{Proof.}\ \text{ of certificates } \mathcal{T}_y^?, \mathcal{T}_z^?, \mathcal{T}_b^?, \mathcal{T}_c^?, \mathcal{T}_a^? \text{ and post-conditions } \mathcal{O}_0^?, \mathcal{O}_1^?, \mathcal{O}_2^?\ \ \texttt{Qed.}
\end{array}
$$

- use of dependent pattern matching

Def. nm_pwc : $\forall e,\ \mathbb{D}_{\texttt{nm}}\ e \to \{n \mid \mathbb{G}_{\texttt{nm}}\ e\ n\}$

```
Fixpoint  nm_pwc e (D : 𝔻nm e) : {n | 𝔾nm e n}.
refine(
 match e as e' return 𝔻nm e' → {n | 𝔾nm e' n} with
   | α               ⇒ λ D,     exist _ α 𝒪₀?
   | ω α y z         ⇒ λ D, let (n_y, C_y) := nm_pwc y 𝒯_y? in
                             let (n_z, C_z) := nm_pwc z 𝒯_z?
                             in  exist _ (ω α n_y n_z) 𝒪₁?
   | ω (ω a b c) y z ⇒ λ D, let (n_b, C_b) := nm_pwc (ω b y z) 𝒯_b? in
                             let (n_c, C_c) := nm_pwc (ω c y z) 𝒯_c? in
                             let (n_a, C_a) := nm_pwc (ω a n_b n_c) 𝒯_a?
                             in  exist _ n_a 𝒪₂?
 end D); simpl in ∗.
```

Proof. of certificates $\mathcal{T}_y^?, \mathcal{T}_z^?, \mathcal{T}_b^?, \mathcal{T}_c^?, \mathcal{T}_a^?$ and post-conditions $\mathcal{O}_0^?, \mathcal{O}_1^?, \mathcal{O}_2^?$ Qed.

- use of dependent pattern matching
- LC (i.e. proof obligations) separated from CC

Def. nm_pwc : $\forall e,\ \mathbb{D}_{\mathtt{nm}}\ e \to \{n \mid \mathbb{G}_{\mathtt{nm}}\ e\ n\}$

```
Fixpoint  nm_pwc e (D : 𝔻nm e) : {n | 𝔾nm e n}.
refine(
 match e as e' return 𝔻nm e' → {n | 𝔾nm e' n} with
   | α              ⇒ λ D,    exist _ α 𝒪?0
   | ω α y z        ⇒ λ D, let (ny, Cy) := nm_pwc y 𝒯?y in
                           let (nz, Cz) := nm_pwc z 𝒯?z
                           in  exist _ (ω α ny nz) 𝒪?1
   | ω (ω a b c) y z ⇒ λ D, let (nb, Cb) := nm_pwc (ω b y z) 𝒯?b in
                           let (nc, Cc) := nm_pwc (ω c y z) 𝒯?c in
                           let (na, Ca) := nm_pwc (ω a nb nc) 𝒯?a
                           in  exist _ na 𝒪?2
 end D); simpl in ∗.
Proof.  of certificates 𝒯?y, 𝒯?z, 𝒯?b, 𝒯?c, 𝒯?a and post-conditions 𝒪?0, 𝒪?1, 𝒪?2  Qed.
```

- use of dependent pattern matching
- LC (i.e. proof obligations) separated from CC
- LC divided: termination certificates, post-conditions

Proof obligations (Logical Contents)

- Post-conditions by the constructors of $\mathbb{G}_{\mathtt{nm}}$

$$\begin{array}{lll}
\mathcal{O}_0^? & /\!\!/ & \ldots \vdash \mathbb{G}_{\mathtt{nm}}\ \alpha\ \alpha \\
\mathcal{O}_1^? & /\!\!/ & \ldots, C_y : \mathbb{G}_{\mathtt{nm}}\ y\ n_y, C_z : \mathbb{G}_{\mathtt{nm}}\ z\ n_z \vdash \mathbb{G}_{\mathtt{nm}}\ (\omega\ \alpha\ y\ z)\ (\omega\ \alpha\ n_y\ n_z) \\
\mathcal{O}_2^? & /\!\!/ & \ldots, C_b : \mathbb{G}_{\mathtt{nm}}\ (\omega\ b\ y\ z)\ n_b, C_c : \mathbb{G}_{\mathtt{nm}}\ (\omega\ c\ y\ z)\ n_c, \ldots \\
 & & \quad \ldots C_a : \mathbb{G}_{\mathtt{nm}}\ (\omega\ a\ n_b\ n_c)\ n_a \vdash \mathbb{G}_{\mathtt{nm}}\ (\omega\ (\omega\ a\ b\ c)\ y\ z)\ n_a
\end{array}$$

Proof obligations (Logical Contents)

- Post-conditions by the constructors of $\mathbb{G}_{\mathtt{nm}}$

$$\begin{array}{lll}
\mathcal{O}_0^? & /\!/ & \ldots \vdash \mathbb{G}_{\mathtt{nm}}\ \alpha\ \alpha \\
\mathcal{O}_1^? & /\!/ & \ldots, C_y : \mathbb{G}_{\mathtt{nm}}\ y\ n_y, C_z : \mathbb{G}_{\mathtt{nm}}\ z\ n_z \vdash \mathbb{G}_{\mathtt{nm}}\ (\omega\ \alpha\ y\ z)\ (\omega\ \alpha\ n_y\ n_z) \\
\mathcal{O}_2^? & /\!/ & \ldots, C_b : \mathbb{G}_{\mathtt{nm}}\ (\omega\ b\ y\ z)\ n_b, C_c : \mathbb{G}_{\mathtt{nm}}\ (\omega\ c\ y\ z)\ n_c, \ldots \\
 & & \quad \ldots C_a : \mathbb{G}_{\mathtt{nm}}\ (\omega\ a\ n_b\ n_c)\ n_a \vdash \mathbb{G}_{\mathtt{nm}}\ (\omega\ (\omega\ a\ b\ c)\ y\ z)\ n_a
\end{array}$$

- Termination certificates

$$\begin{array}{lll}
\mathcal{T}_y^? & /\!/ & \ldots, D : \mathbb{D}_{\mathtt{nm}}\ (\omega\ \alpha\ y\ z) \vdash \mathbb{D}_{\mathtt{nm}}\ y \\
\mathcal{T}_b^? & /\!/ & \ldots, D : \mathbb{D}_{\mathtt{nm}}\ (\omega\ (\omega\ a\ b\ c)\ y\ z) \vdash \mathbb{D}_{\mathtt{nm}}\ (\omega\ b\ y\ z) \\
\mathcal{T}_a^? & /\!/ & \ldots, D : \mathbb{D}_{\mathtt{nm}}\ (\omega\ (\omega\ a\ b\ c)\ y\ z), H_b : \mathbb{G}_{\mathtt{nm}}\ (\omega\ b\ y\ z)\ n_b, \ldots \\
 & & \quad \ldots H_c : \mathbb{G}_{\mathtt{nm}}\ (\omega\ c\ y\ z)\ n_c \vdash \mathbb{D}_{\mathtt{nm}}\ (\omega\ a\ n_b\ n_c)
\end{array}$$

- beware of structural decrease in term. certificates
  - by the `inversion` tactic
  - or “small inversion” (human readable)

Simulated IR scheme

- $\texttt{nm}\ e\ D := \pi_1(\texttt{nm\_pwc}\ e\ D)$ and $\pi_2 : \mathbb{G}_{\texttt{nm}}\ e\ (\texttt{nm}\ e\ D)$

Simulated IR scheme

- $\mathtt{nm}\ e\ D := \pi_1(\mathtt{nm\_pwc}\ e\ D)$ and $\pi_2 : \mathbb{G}_{\mathtt{nm}}\ e\ (\mathtt{nm}\ e\ D)$

$$
\begin{array}{lll}
\mathtt{Inductive}\ \ \mathbb{D}_{\mathtt{nm}} : \Omega \to \boxed{\mathtt{Prop}} := & & \\
\quad \mid \mathbb{D}^1_{\mathtt{nm}} & : & \mathbb{D}_{\mathtt{nm}}\ \alpha \\
\quad \mid \mathbb{D}^2_{\mathtt{nm}}\ y\ z & : & \mathbb{D}_{\mathtt{nm}}\ y \to \mathbb{D}_{\mathtt{nm}}\ z \to \mathbb{D}_{\mathtt{nm}}(\omega\ \alpha\ y\ z) \\
\quad \mid \mathbb{D}^3_{\mathtt{nm}}\ a\ b\ c\ y\ z\ D_b\ D_c & : & \mathbb{D}_{\mathtt{nm}}(\omega\ a\ (\mathtt{nm}\ (\omega\ b\ y\ z)\ D_b)\ (\mathtt{nm}\ (\omega\ c\ y\ z)\ D_c)) \\
 & & \to \mathbb{D}_{\mathtt{nm}}(\omega\ (\omega\ a\ b\ c)\ y\ z)
\end{array}
$$

$$
\begin{array}{ll}
\mathtt{with\ Fixpoint}\ \ \mathtt{nm}\ e\ (D_e : \mathbb{D}_{\mathtt{nm}}\ e) : \Omega := & \\
\ \mathtt{match}\ D_e\ \mathtt{with} & \\
\quad \mid \mathbb{D}^1_{\mathtt{nm}} & \Rightarrow \alpha \\
\quad \mid \mathbb{D}^2_{\mathtt{nm}}\ y\ z\ D_y\ D_z & \Rightarrow \omega\ \alpha\ (\mathtt{nm}\ y\ D_y)\ (\mathtt{nm}\ z\ D_z) \\
\quad \mid \mathbb{D}^3_{\mathtt{nm}}\ a\ b\ c\ y\ z\ D_b\ D_c\ D_a & \Rightarrow \mathtt{nm}\ \big(\omega\ a\ (\mathtt{nm}\ (\omega\ b\ y\ z)\ D_b) \\
 & \qquad\qquad (\mathtt{nm}\ (\omega\ c\ y\ z)\ D_c)\big)\ D_a \\
\ \mathtt{end.} &
\end{array}
$$

Simulated IR scheme

- $\mathtt{nm}\ e\ D := \pi_1(\mathtt{nm\_pwc}\ e\ D)$ and $\pi_2 : \mathbb{G}_{\mathtt{nm}}\ e\ (\mathtt{nm}\ e\ D)$

```
Inductive 𝔻nm : Ω → Prop :=
  | 𝔻¹nm                    : 𝔻nm α
  | 𝔻²nm y z                : 𝔻nm y → 𝔻nm z → 𝔻nm(ω α y z)
  | 𝔻³nm a b c y z Db Dc    : 𝔻nm(ω a (nm (ω b y z) Db) (nm (ω c y z) Dc))
                            → 𝔻nm(ω (ω a b c) y z)
with Fixpoint nm e (De : 𝔻nm e) : Ω :=
 match De with
  | 𝔻¹nm                    ⇒ α
  | 𝔻²nm y z Dy Dz          ⇒ ω α (nm y Dy) (nm z Dz)
  | 𝔻³nm a b c y z Db Dc Da ⇒ nm (ω a (nm (ω b y z) Db)
                                       (nm (ω c y z) Dc)) Da
 end.
```

- The domain $\mathbb{D}_{\mathtt{nm}} : \Omega \to \mathtt{Prop}$ is **non-informative**

Simulated IR scheme

- $\mathtt{nm}\ e\ D := \pi_1(\mathtt{nm\_pwc}\ e\ D)$ and $\pi_2 : \mathbb{G}_{\mathtt{nm}}\ e\ (\mathtt{nm}\ e\ D)$

$$
\begin{array}{ll}
\multicolumn{2}{l}{\mathtt{Inductive}\ \ \mathbb{D}_{\mathtt{nm}} : \Omega \to \boxed{\mathtt{Prop}} :=} \\
\quad \mid \mathbb{D}^1_{\mathtt{nm}} & : \mathbb{D}_{\mathtt{nm}}\ \alpha \\
\quad \mid \mathbb{D}^2_{\mathtt{nm}}\ y\ z & : \mathbb{D}_{\mathtt{nm}}\ y \to \mathbb{D}_{\mathtt{nm}}\ z \to \mathbb{D}_{\mathtt{nm}}(\omega\ \alpha\ y\ z) \\
\quad \mid \mathbb{D}^3_{\mathtt{nm}}\ a\ b\ c\ y\ z\ D_b\ D_c & : \mathbb{D}_{\mathtt{nm}}(\omega\ a\ (\mathtt{nm}\ (\omega\ b\ y\ z)\ D_b)\ (\mathtt{nm}\ (\omega\ c\ y\ z)\ D_c)) \\
 & \to \mathbb{D}_{\mathtt{nm}}(\omega\ (\omega\ a\ b\ c)\ y\ z) \\
\multicolumn{2}{l}{\mathtt{with\ Fixpoint}\ \ \mathtt{nm}\ e\ (D_e : \mathbb{D}_{\mathtt{nm}}\ e) : \Omega :=} \\
\multicolumn{2}{l}{\ \mathtt{match}\ D_e\ \mathtt{with}} \\
\quad \mid \mathbb{D}^1_{\mathtt{nm}} & \Rightarrow \alpha \\
\quad \mid \mathbb{D}^2_{\mathtt{nm}}\ y\ z\ D_y\ D_z & \Rightarrow \omega\ \alpha\ (\mathtt{nm}\ y\ D_y)\ (\mathtt{nm}\ z\ D_z) \\
\quad \mid \mathbb{D}^3_{\mathtt{nm}}\ a\ b\ c\ y\ z\ D_b\ D_c\ D_a & \Rightarrow \mathtt{nm}\ (\omega\ a\ (\mathtt{nm}\ (\omega\ b\ y\ z)\ D_b) \\
 & \qquad\qquad (\mathtt{nm}\ (\omega\ c\ y\ z)\ D_c))\ D_a \\
\multicolumn{2}{l}{\ \mathtt{end.}}
\end{array}
$$

- The domain $\mathbb{D}_{\mathtt{nm}} : \Omega \to \mathtt{Prop}$ is **non-informative**
- $\mathtt{nm} : \forall e, \mathbb{D}_{\mathtt{nm}}\ e \to \Omega$ is **proof-irrelevant**, i.e.
  $\mathtt{nm}\ x\ D_1 = \mathtt{nm}\ x\ D_2$

Simulated IR scheme

- $\texttt{nm}\ e\ D := \pi_1(\texttt{nm\_pwc}\ e\ D)$ and $\pi_2 : \mathbb{G}_{\texttt{nm}}\ e\ (\texttt{nm}\ e\ D)$

```
Inductive 𝔻nm : Ω → Prop :=
  | 𝔻nm¹                        : 𝔻nm α
  | 𝔻nm² y z                    : 𝔻nm y → 𝔻nm z → 𝔻nm(ω α y z)
  | 𝔻nm³ a b c y z Db Dc        : 𝔻nm(ω a (nm (ω b y z) Db) (nm (ω c y z) Dc))
                                → 𝔻nm(ω (ω a b c) y z)
with Fixpoint nm e (De : 𝔻nm e) : Ω :=
 match De with
  | 𝔻nm¹                        ⇒ α
  | 𝔻nm² y z Dy Dz              ⇒ ω α (nm y Dy) (nm z Dz)
  | 𝔻nm³ a b c y z Db Dc Da     ⇒ nm (ω a (nm (ω b y z) Db)
                                          (nm (ω c y z) Dc)) Da
 end.
```

- The domain $\mathbb{D}_{\texttt{nm}} : \Omega \to \texttt{Prop}$ is **non-informative**
- $\texttt{nm} : \forall e, \mathbb{D}_{\texttt{nm}}\ e \to \Omega$ is **proof-irrelevant**, i.e. $\texttt{nm}\ x\ D_1 = \texttt{nm}\ x\ D_2$
- Constructors, dep. elim. scheme and fixpoint equations *retrieved*

Extraction unaltered by $\mathbb{D}_{\mathtt{nm}}$ in $\mathtt{Prop}$

- In $\mathtt{nm}\ e\ (D : \mathbb{D}_{\mathtt{nm}}\ e)$ extract. erases $D : \mathbb{D}_{\mathtt{nm}}\ e : \mathtt{Prop}$

Extraction unaltered by $\mathbb{D}_{\texttt{nm}}$ in $\texttt{Prop}$

- In $\texttt{nm}\ e\ (D : \mathbb{D}_{\texttt{nm}}\ e)$ extract. erases $D : \mathbb{D}_{\texttt{nm}}\ e : \texttt{Prop}$
- Hence $\texttt{Extraction nm}$ gives the intended term:

$$
\begin{array}{l}
\texttt{let rec nm}\ e = \texttt{match}\ e\ \texttt{with} \\
\quad \mid \alpha \qquad\qquad\quad \to \alpha \\
\quad \mid \omega(x,y,z) \quad \to \texttt{match}\ x\ \texttt{with} \\
\qquad \mid \alpha \qquad\qquad \to \omega(\alpha, \texttt{nm}\ y, \texttt{nm}\ z) \\
\qquad \mid \omega(a,b,c) \to \texttt{nm}(\omega(a, \texttt{nm}(\omega(b,y,z)), \texttt{nm}(\omega(c,y,z))))
\end{array}
$$

Extraction unaltered by $\mathbb{D}_{\texttt{nm}}$ in Prop

- In $\texttt{nm}\ e\ (D : \mathbb{D}_{\texttt{nm}}\ e)$ extract. erases $D : \mathbb{D}_{\texttt{nm}}\ e : \texttt{Prop}$
- Hence `Extraction nm` gives the intended term:

$$
\begin{array}{l}
\texttt{let rec nm}\ e = \texttt{match}\ e\ \texttt{with} \\
\quad \mid \alpha \qquad\qquad\quad \to \alpha \\
\quad \mid \omega(x,y,z) \quad\ \to \texttt{match}\ x\ \texttt{with} \\
\qquad \mid \alpha \qquad\qquad \to \omega(\alpha, \texttt{nm}\ y, \texttt{nm}\ z) \\
\qquad \mid \omega(a,b,c) \to \texttt{nm}(\omega(a, \texttt{nm}(\omega(b,y,z)), \texttt{nm}(\omega(c,y,z))))
\end{array}
$$

- The proof term $D : \mathbb{D}_{\texttt{nm}}\ e$
  - has **no impact** on extracted algorithm

Extraction unaltered by $\mathbb{D}_{\texttt{nm}}$ in Prop

- In $\texttt{nm}\ e\ (D : \mathbb{D}_{\texttt{nm}}\ e)$ extract. erases $D : \mathbb{D}_{\texttt{nm}}\ e : \texttt{Prop}$
- Hence `Extraction nm` gives the intended term:

$$
\begin{array}{l}
\texttt{let rec nm}\ e = \texttt{match}\ e\ \texttt{with} \\
\quad \mid \alpha \qquad\qquad\qquad \to \alpha \\
\quad \mid \omega(x,y,z) \qquad \to \texttt{match}\ x\ \texttt{with} \\
\qquad \mid \alpha \qquad\qquad\quad \to \omega(\alpha, \texttt{nm}\ y, \texttt{nm}\ z) \\
\qquad \mid \omega(a,b,c) \to \texttt{nm}(\omega(a, \texttt{nm}(\omega(b,y,z)), \texttt{nm}(\omega(c,y,z))))
\end{array}
$$

- The proof term $D : \mathbb{D}_{\texttt{nm}}\ e$
  - has **no impact** on extracted algorithm
  - great complexity does not matter

Extraction unaltered by $\mathbb{D}_{\texttt{nm}}$ in Prop

- In $\texttt{nm}\ e\ (D : \mathbb{D}_{\texttt{nm}}\ e)$ extract. erases $D : \mathbb{D}_{\texttt{nm}}\ e : \texttt{Prop}$
- Hence `Extraction nm` gives the intended term:

$$
\begin{array}{l}
\texttt{let rec nm}\ e = \texttt{match}\ e\ \texttt{with} \\
\quad \mid \alpha \qquad\qquad\quad \to \alpha \\
\quad \mid \omega(x, y, z) \quad \to \texttt{match}\ x\ \texttt{with} \\
\qquad \mid \alpha \qquad\qquad \to \omega(\alpha, \texttt{nm}\ y, \texttt{nm}\ z) \\
\qquad \mid \omega(a, b, c) \to \texttt{nm}(\omega(a, \texttt{nm}(\omega(b, y, z)), \texttt{nm}(\omega(c, y, z))))
\end{array}
$$

- The proof term $D : \mathbb{D}_{\texttt{nm}}\ e$
  - has **no impact** on extracted algorithm
  - great complexity does not matter
  - use high-level tool (lex. prod, WQOs)

Termination postponed after definition

- Proving termination of $\mathtt{nm}$ at e is a term $D : \mathbb{D}_{\mathtt{nm}}\ e$
  - a "meaningful" characterization of $\mathbb{D}_{\mathtt{nm}}\ e$
  - for partial fun.: $P : \Omega \to \mathtt{Prop}$ and $P \subseteq \mathbb{D}_{\mathtt{nm}}$
  - for total functions: a proof of $\forall e, \mathbb{D}_{\mathtt{nm}}\ e$

Termination postponed after definition

- Proving termination of $\mathtt{nm}$ at e is a term $D : \mathbb{D}_{\mathtt{nm}}\ e$
  - a "meaningful" characterization of $\mathbb{D}_{\mathtt{nm}}\ e$
  - for partial fun.: $P : \Omega \to \mathtt{Prop}$ and $P \subseteq \mathbb{D}_{\mathtt{nm}}$
  - for total functions: a proof of $\forall e, \mathbb{D}_{\mathtt{nm}}\ e$
- The proof of $P \subseteq \mathbb{D}_{\mathtt{nm}}$ can be provided:
  - after $\mathbb{D}_{\mathtt{nm}} : \Omega \to \mathtt{Prop}$ and $\mathtt{nm} : \forall e,\ \mathbb{D}_{\mathtt{nm}}\ e \to \Omega$ are def'd
  - w/o consequences on extracted code
  - including by adding axioms (if necessary)

Termination postponed after definition

- Proving termination of $\mathtt{nm}$ at e is a term $D : \mathbb{D}_{\mathtt{nm}}\ e$
  - a "meaningful" characterization of $\mathbb{D}_{\mathtt{nm}}\ e$
  - for partial fun.: $P : \Omega \to \mathtt{Prop}$ and $P \subseteq \mathbb{D}_{\mathtt{nm}}$
  - for total functions: a proof of $\forall e, \mathbb{D}_{\mathtt{nm}}\ e$
- The proof of $P \subseteq \mathbb{D}_{\mathtt{nm}}$ can be provided:
  - after $\mathbb{D}_{\mathtt{nm}} : \Omega \to \mathtt{Prop}$ and $\mathtt{nm} : \forall e,\ \mathbb{D}_{\mathtt{nm}}\ e \to \Omega$ are def'd
  - w/o consequences on extracted code
  - including by adding axioms (if necessary)
- Tools from IR:
  - constructors
  - fixpoint equations

Partial correction postponed after def.

- Partial correction = higher-level charac. of `nm` e D

Partial correction postponed after def.

- Partial correction = higher-level charac. of $\texttt{nm}\ e\ D$
  - another spec/post-condition
  - by induction on $\mathbb{G}_{\texttt{nm}}\ e\ (\texttt{nm}\ e\ D)$
  - or using dependent elimination on (e, D) (IR)

Partial correction postponed after def.

- Partial correction = higher-level charac. of $\mathtt{nm}\ e\ D$
  - another spec/post-condition
  - by induction on $\mathbb{G}_{\mathtt{nm}}\ e\ (\mathtt{nm}\ e\ D)$
  - or using dependent elimination on (e, D) (IR)
- Partial correction: for meaningful $\mathbb{S}$
  - $\forall e\ (D : \mathbb{D}_{\mathtt{nm}}\ e), \mathbb{S}\ e\ (\mathtt{nm}\ e\ D)$

Partial correction postponed after def.

- Partial correction = higher-level charac. of $\mathtt{nm}\ e\ D$
  - another spec/post-condition
  - by induction on $\mathbb{G}_{\mathtt{nm}}\ e\ (\mathtt{nm}\ e\ D)$
  - or using dependent elimination on (e, D) (IR)
- Partial correction: for meaningful $\mathbb{S}$
  - $\forall e\ (D : \mathbb{D}_{\mathtt{nm}}\ e), \mathbb{S}\ e\ (\mathtt{nm}\ e\ D)$
- Tools from IR:
  - dependent elimination
  - fixpoint equations

Partial correction of nm on $\mathbb{D}_{\texttt{nm}}$

- dep. elim. $\mathbb{D}_{\texttt{nm}}$_rect for partial correction (IR)

Partial correction of $\mathtt{nm}$ on $\mathbb{D}_{\mathtt{nm}}$

- dep. elim. $\mathbb{D}_{\mathtt{nm}}\mathtt{\_rect}$ for partial correction (IR)
- $\mathtt{nm\_normal} : \forall\, e\, (D : \mathbb{D}_{\mathtt{nm}}\ e),\ \mathtt{normal}\,(\mathtt{nm}\ e\ D)$
  - the shape $\omega\ (\omega\ \_\ \_\ \_)\ \_\ \_$ is forbidden

Partial correction of $\mathtt{nm}$ on $\mathbb{D}_{\mathtt{nm}}$

- dep. elim. $\mathbb{D}_{\mathtt{nm}}$_rect for partial correction (IR)
- $\mathtt{nm\_normal} : \forall\, e\, (D : \mathbb{D}_{\mathtt{nm}}\ e),\ \mathtt{normal}\,(\mathtt{nm}\ e\ D)$
  - the shape $\omega\ (\omega\ \_\ \_\ \_)\ \_\ \_$ is forbidden
- $\mathtt{nm\_equiv} : \forall\, e\, (D : \mathbb{D}_{\mathtt{nm}}\ e),\ e \simeq_{\Omega} \mathtt{nm}\ e\ D$
  - the normal form is computationaly equiv.

Partial correction of $\mathtt{nm}$ on $\mathbb{D}_{\mathtt{nm}}$

- dep. elim. $\mathbb{D}_{\mathtt{nm}}$_rect for partial correction (IR)
- $\mathtt{nm\_normal} : \forall\, e\, (D : \mathbb{D}_{\mathtt{nm}}\ e),\ \mathtt{normal}\,(\mathtt{nm}\ e\ D)$
  - the shape $\omega\ (\omega\ \_\ \_\ \_)\ \_\ \_$ is forbidden
- $\mathtt{nm\_equiv} : \forall\, e\, (D : \mathbb{D}_{\mathtt{nm}}\ e),\ e \simeq_{\Omega} \mathtt{nm}\ e\ D$
  - the normal form is computationaly equiv.
- $\mathtt{nm\_dec} : \forall\, e\, (D : \mathbb{D}_{\mathtt{nm}}\ e),\ |\mathtt{nm}\ e\ D| \leqslant |e|$
  - some "size" $|\cdot| : \Omega \to \mathtt{nat}$ is preserved (Giesl 97)

$$|\alpha| = 1 \qquad |\omega\ x\ y\ z| = |x| \cdot (1 + |y| + |z|)$$

Totality of $\mathbb{D}_{\mathtt{nm}}$ / Termination of $\mathtt{nm}$

$$\mathbb{D}_{\mathtt{nm}}\mathtt{\_total} : \forall e,\ \mathbb{D}_{\mathtt{nm}}\ e$$

- By induction on the size $|e|$

Totality of $\mathbb{D}_{\texttt{nm}}$ / Termination of $\texttt{nm}$

$$\mathbb{D}_{\texttt{nm}}\texttt{\_total} : \forall e,\ \mathbb{D}_{\texttt{nm}}\ e$$

- By induction on the size $|e|$
  - we use $\texttt{nm\_dec} : \forall\, e\ (D : \mathbb{D}_{\texttt{nm}}\ e),\ |\texttt{nm}\ e\ D| \leqslant |e|$

Totality of $\mathbb{D}_{\mathtt{nm}}$ / Termination of $\mathtt{nm}$

$$\mathbb{D}_{\mathtt{nm}}\_\mathtt{total} : \forall e, \mathbb{D}_{\mathtt{nm}}\ e$$

- By induction on the size $|e|$
  - we use $\mathtt{nm\_dec} : \forall\, e\ (D : \mathbb{D}_{\mathtt{nm}}\ e), |\mathtt{nm}\ e\ D| \leqslant |e|$
  - and $|\omega\ x\ y\ z| \leqslant |\omega\ x'\ y'\ z'|$ (monotonic)
    - i.e. when $|x| \leqslant |x'|, |y| \leqslant |y'|, |z| \leqslant |z'|$
  - and $|\omega\ u\ y\ z| < |\omega\ v\ y\ z|$ when $|u| < |v|$
  - and $|y| < |\omega\ x\ y\ z|$ and $|z| < |\omega\ x\ y\ z|$
  - & $|\omega\ a\ (\omega\ b\ y\ z)\ (\omega\ c\ y\ z)| < |\omega\ (\omega\ a\ b\ c)\ y\ z|$
- Partial correction / termination indep. of definition

$$\boxed{\mathtt{paulson\_nm} : \forall e : \Omega, \{n_e : \Omega \mid e \simeq_{\Omega} n_e \wedge \mathtt{normal}\ e\}}$$

Third take home ideas

- Domain $\mathbb{D}$ for nested schemes
  - use $\mathbb{G}$ to characterize (nested) output values
  - and define $\mathbb{D}$ after&using $\mathbb{G}$

Third take home ideas

- Domain $\mathbb{D}$ for nested schemes
  - use $\mathbb{G}$ to characterize (nested) output values
  - and define $\mathbb{D}$ after&using $\mathbb{G}$
- Correctness of nested schemes
  - can be studied before termination
  - can be used to prove termination

Conclusion

- The Braga method separates tasks
  - definition of the function in Coq
  - prove its partial correctness (IR or graph ind.)
  - prove (partial) termination (from correctness)

Conclusion

- The Braga method separates tasks
  - definition of the function in Coq
  - prove its partial correctness (IR or graph ind.)
  - prove (partial) termination (from correctness)
- The algorithm is enough
  - to define the function
  - no need to know why it terminates
  - no need to know what it computes

Conclusion

- The Braga method separates tasks
  - definition of the function in Coq
  - prove its partial correctness (IR or graph ind.)
  - prove (partial) termination (from correctness)
- The algorithm is enough
  - to define the function
  - no need to know why it terminates
  - no need to know what it computes
- Extraction
  - erases the Logical Contents (LC)
  - keeps the Computational Contents (CC)
  - give access to partial algorithms
  - incl. nested and mutual

Conclusion

- The Braga method separates tasks
  - definition of the function in Coq
  - prove its partial correctness (IR or graph ind.)
  - prove (partial) termination (from correctness)
- The algorithm is enough
  - to define the function
  - no need to know why it terminates
  - no need to know what it computes
- Extraction
  - erases the Logical Contents (LC)
  - keeps the Computational Contents (CC)
  - give access to partial algorithms
  - incl. nested and mutual
- Perspectives
  - better integrate with existing tools
  - more examples, e.g. Knuth k_{91}, μ-rec. algos.
  - partial functions as guarded total functions