

Program-ming with Computational Induction

or: STP 2003-07-10 revisited

James McKinna
based on joint work with
Dan Synek, Eelis vd Weegen

Foundations/Intelligent Systems, Radboud Universiteit Nijmegen
Thanks to: NWO/BRICKS project 'ARPA', NWO cluster 'DIAMANT'

STP talk, St.Andrews 2009-06-12

Outline

- ▶ The problem
- ▶ An imperative algorithm for reachability
- ▶ A tail-recursive, functional representation
- ▶ Problems with representation in type theory
- ▶ PROGRAM: separating programs from proof obligations
- ▶ Computational induction: a canonical choice of specification?
- ▶ Further work and conclusions

Full paper at: <http://www.cs.ru.nl/~james>

The problem

- ▶ to compute the subset of nodes of a finite graph reachable from an initial subset *start*
- ▶ and prove your implementation correct (Dijkstra, 1959)
- ▶ *i.e.* given a subset S and an edge relation R , compute R^*S , the *least* set X satisfying
 - ▶ $S \subseteq X$
 - ▶ $R * X \subseteq X$

The twist in the tail

- ▶ to write a terminating function in a fragment of CIC (type theory of Coq)
- ▶ ... and prove it correct in Coq
- ▶ starting from a non-structurally recursive algorithm (imperative code!)

The problem

- ▶ to compute the subset of nodes of a finite graph reachable from an initial subset *start*
- ▶ and prove your implementation correct (Dijkstra, 1959)
- ▶ *i.e.* given a subset S and an edge relation R , compute R^*S , the *least* set X satisfying
 - ▶ $S \subseteq X$
 - ▶ $R * X \subseteq X$

The twist in the tail

- ▶ to write a terminating function in a fragment of CIC (type theory of Coq)
- ▶ ... and prove it correct in Coq
- ▶ starting from a non-structurally recursive algorithm (imperative code!)

The problem

- ▶ to compute the subset of nodes of a finite graph reachable from an initial subset *start*
- ▶ and prove your implementation correct (Dijkstra, 1959)
- ▶ *i.e.* given a subset S and an edge relation R , compute R^*S , the *least* set X satisfying
 - ▶ $S \subseteq X$
 - ▶ $R * X \subseteq X$

The twist in the tail

- ▶ to write a terminating function in a fragment of CIC (type theory of Coq)
- ▶ ... and prove it correct in Coq
- ▶ starting from a non-structurally recursive algorithm (imperative code!)

Algorithm reachable: imperative pseudocode

```
inputs:  subset "start" of vertices.
pre-condition:
output:  subset "result" of vertices.
post-condition:
  for all v, (v in result)  $\leftrightarrow$ 
    (v reachable from start).
begin
locals:  subsets "visited" and "waiting".
  visited := {};
  waiting := start;
  while (waiting  $\neq$  {})
    {pick w in waiting;
     visited := visited  $\cup$  {w};
     waiting := (neighbours w)  $\cup$  waiting  $\setminus$  visited;
    };
  result := visited;
end
```

A tail-recursive, pseudo-functional representation

Factor imperative control into a *worker* function, expressing the result in terms of the local variables

```
Fixpoint reachable_worker (V W: SubsetV) :=  
  match W with  
  | {} => V  
  | {w} ∪ W => (* w ∉ W *)  
    reachable_worker ({w} ∪ V) (rstep w V W)  
end.
```

and a *wrapper*, which initialises them

```
Definition reachable  
  := reachable_worker {} start.
```

where the step function in the tail-recursive worker is

```
Definition rstep w V W  
  := (R*{w} ∪ W) \ ({w} ∪ V).
```

A tail-recursive, pseudo-functional representation

Factor imperative control into a *worker* function, expressing the result in terms of the local variables

```
Fixpoint reachable_worker (V W: SubsetV) :=  
  match W with  
  | {} => V  
  | {w} ∪ W => (* w ∉ W *)  
    reachable_worker ({w} ∪ V) (rstep w V W)  
end.
```

and a *wrapper*, which initialises them

```
Definition reachable  
  := reachable_worker {} start.
```

where the step function in the tail-recursive worker is

```
Definition rstep w V W  
  := (R*{w} ∪ W) \ ({w} ∪ V).
```

The problem with type theory/CIC/Coq

The definition *reachable_worker* $V W$ is **not**

- ▶ structurally recursive
- ▶ obviously terminating at all

So *reachable* S and *reachable_worker* $V W$ are **not** legitimate definitions in type theory.

Less seriously, we have work to do to represent *subsets*

- ▶ use *lists* (1970s/80s solution): invariants and correctness get polluted with additional ADT invariants
- ▶ use *type classes* (1990s/2000s): abstract over how sets/subsets *behave*; show that, e.g. respect these abstractions

The problem with type theory/CIC/Coq

The definition *reachable_worker* $V W$ is **not**

- ▶ structurally recursive
- ▶ obviously terminating at all

So *reachable* S and *reachable_worker* $V W$ are **not** legitimate definitions in type theory.

Less seriously, we have work to do to represent *subsets*

- ▶ use *lists* (1970s/80s solution): invariants and correctness get polluted with additional ADT invariants
- ▶ use *type classes* (1990s/2000s): abstract over how sets/subsets *behave*; show that, e.g. respect these abstractions

The problem with type theory/CIC/Coq

The definition *reachable_worker* $V W$ is **not**

- ▶ structurally recursive
- ▶ obviously terminating at all

So *reachable* S and *reachable_worker* $V W$ are **not** legitimate definitions in type theory.

Less seriously, we have work to do to represent *subsets*

- ▶ use *lists* (1970s/80s solution): invariants and correctness get polluted with additional ADT invariants
- ▶ use *type classes* (1990s/2000s): abstract over how sets/subsets *behave*; show that, e.g. respect these abstractions

Possible solutions

- ▶ (1970s/1980s) Use well-founded recursion directly to show that *reachable_worker V W* is definable
 - ▶ hard work to do by hand
 - ▶ then harder to prove that the function has any properties
- ▶ (late 1990s) Use Bove-Capretta 'domain predicates' to characterise inductively, when *reachable_worker V W* is definable
 - ▶ define *reachable_worker V W* by induction on **proofs** of this predicate
 - ▶ **and** show that this predicate always holds
 - ▶ **but** need to repeat this proof technique to establish any properties
- ▶ ...
- ▶ (2007–present) Use Matthieu Sozeau's Program machinery (PhD thesis, Paris XII/LRI, dec.2008)

Possible solutions

- ▶ (1970s/1980s) Use well-founded recursion directly to show that *reachable_worker* V W is definable
 - ▶ hard work to do by hand
 - ▶ then harder to prove that the function has any properties
- ▶ (late 1990s) Use Bove-Capretta 'domain predicates' to characterise inductively, when *reachable_worker* V W is definable
 - ▶ define *reachable_worker* V W by induction on **proofs** of this predicate
 - ▶ **and** show that this predicate always holds
 - ▶ **but** need to repeat this proof technique to establish any properties
- ▶ ...
- ▶ (2007–present) Use Matthieu Sozeau's Program machinery (PhD thesis, Paris XII/LRI, dec.2008)

Program: formalising the separation of concerns

Program is a suite of commands and tactics which permit:

- ▶ definition of non-structural recursive functions (via measures)
- ▶ separate termination from correctness
- ▶ permit definitions ‘up to’ proof obligations. . .
- ▶ . . . whose solution can be deferred (or even admitted as axioms)

The notation:

- ▶ is upwardly compatible with standard syntax
- ▶ supports dependently-typed programming via *predicate subtyping* (yes, NuPRL and PVS were here first!)
- ▶ extracts to ‘the program you first thought of’

Program: formalising the separation of concerns

Program is a suite of commands and tactics which permit:

- ▶ definition of non-structural recursive functions (via measures)
- ▶ separate termination from correctness
- ▶ permit definitions ‘up to’ proof obligations. . .
- ▶ . . . whose solution can be deferred (or even admitted as axioms)

The notation:

- ▶ is upwardly compatible with standard syntax
- ▶ supports dependently-typed programming via *predicate subtyping* (yes, NuPRL and PVS were here first!)
- ▶ extracts to ‘the program you first thought of’

The ‘specification’ problem; the ‘correctness’ problem

Given a putative function definition, how should we show it meets its specification?

- ▶ bake it in, via predicate subtyping, to its `Program Definition`
- ▶ ‘separation of concerns’
 - ▶ do the minimum amount of work to define a well-typed *terminating* function (e.g. from lists to lists)
 - ▶ prove that, when defined, this function has the desired properties
- ▶ do a little more work: you know a non-trivial predicate which the output **must** satisfy, if it is defined at all...
- ▶ ... previously (and erroneously!) called ‘McCarthy-Painter’ induction (JHM)

The ‘specification’ problem; the ‘correctness’ problem

Given a putative function definition, how should we show it meets its specification?

- ▶ bake it in, via predicate subtyping, to its `Program Definition`
- ▶ ‘separation of concerns’
 - ▶ do the minimum amount of work to define a well-typed *terminating* function (e.g. from lists to lists)
 - ▶ prove that, when defined, this function has the desired properties
- ▶ do a little more work: you know a non-trivial predicate which the output **must** satisfy, if it is defined at all...
- ▶ ... previously (and erroneously!) called ‘McCarthy-Painter’ induction (JHM)

Another, earlier, talk at STP, 2003-07-10

... six years ago ...

<http://www.cs.st-and.ac.uk/~james/RESEARCH/stp.pdf>

Another, earlier, talk at STP, 2003-07-10

... six years ago ...

<http://www.cs.st-and.ac.uk/~james/RESEARCH/stp.pdf>

Another, earlier, talk at STP, 2003-07-10

... six years ago ...

<http://www.cs.st-and.ac.uk/~james/RESEARCH/stp.pdf>

Computational Induction principle for `reachable`

Given V, W subsets of nodes, call a subset X of nodes

- ▶ (V, W) -*reachable*, which we write
- ▶ as a 3-place relation $\text{Reachable } V W X$,

where $\text{Reachable } V W X$ is the *least* relation satisfying:

- ▶ $\text{Reachable } X \emptyset X$,
- ▶ if $w \notin W$ and $\text{Reachable } (\{w\} \cup V) (\text{rstep } w V W) X$
then $\text{Reachable } V (\{w\} \cup W) X$, where
- ▶ $\text{rstep } w V W = (R * \{w\} \cup W) \setminus V$

This is McCarthy's *computational induction* principle for the
function $\text{reachable_worker } V W$

If $\text{reachable_worker } V W$ is well-defined, then
 $\text{Reachable } V W (\text{reachable_worker } V W)$.

Computational Induction principle for `reachable`

Given V, W subsets of nodes, call a subset X of nodes

- ▶ (V, W) -*reachable*, which we write
- ▶ as a 3-place relation $\text{Reachable } V W X$,

where $\text{Reachable } V W X$ is the *least* relation satisfying:

- ▶ $\text{Reachable } X \emptyset X$,
- ▶ if $w \notin W$ and $\text{Reachable } (\{w\} \cup V) (\text{rstep } w V W) X$
then $\text{Reachable } V (\{w\} \cup W) X$, where
- ▶ $\text{rstep } w V W = (R * \{w\} \cup W) \setminus V$

This is McCarthy's *computational induction* principle for the
function $\text{reachable_worker } V W$

If $\text{reachable_worker } V W$ is well-defined, then
 $\text{Reachable } V W (\text{reachable_worker } V W)$.

Correctness

Computational Induction for a function f is

- ▶ **rule induction** for the least fixedpoint semantics of f ...
- ▶ ... considered as a **relation** (Prolog!)

Moreover, it can be given a **compositional** definition in terms of f

Correctness for $reachable_worker\ V\ W$ thus reduces to

- ▶ **termination**: a Program Definition that it satisfies $Reachable\ V\ W\ (reachable_worker\ V\ W)$
 - ▶ well-founded recursion on a measure
 - ▶ the other proof obligations are **automatic**: that's how $Reachable\ V\ W\ X$ is defined!
- ▶ **partial correctness**: proofs by induction on $Reachable\ V\ W\ X$ that X is indeed what you want

Correctness

Proofs omitted!

Compositionality, Abstraction, Modularity

Compositionality:

- ▶ the function invocation $reachable_worker\ V\ \{w\} \cup W$ is defined in terms of $rstep\ w\ V\ W$
- ▶ but $rstep\ w\ V\ W$ will have an inductively-defined graph $Rstep\dots$
- ▶ ... and **any** function satisfying that graph relation can be used instead
- ▶ because $Reachable\ V\ W\ X$ can be defined in terms of $Rstep$
- ▶ indeed, in terms of $Pick$ relation as well (not shown)

Abstraction, modularity:

- ▶ other selection strategies are possible: depth-first, breadth-first, best-first (for SSSP)
- ▶ relatively weak axiomatisation needed for $Rstep$
- ▶ get a factorisation of the correctness proof

Further work

Within the *object logic* of the theorem prover

- ▶ Separate out abstract set-theoretic representation via *type classes*
- ▶ Factor out the relation characterising `rstep`
- ▶ Extend to SSSP problem

Within the *implementation* of the theorem prover

- ▶ Mechanise the passage from function to inductive relation, incl. compositionality
- ▶ Allow the user control of what inductions are used
- ▶ Analyse/modify the computational behaviour of the internally-defined term

Most of the object reasoning been done, but not the implementation level work (need a committed, and involved, CoQ developer)

Conclusions

Possible conclusions:

- ▶ users of HOL, PVS, perhaps not so surprised
- ▶ users of Coq now need not scratch their heads too hard over non-structural recursions
- ▶ Program is a cool suite of functionality (with a lot of formally-checked meta-theory behind it)
- ▶ no difference between McCarthy in LISP, Dijkstra in IMP, Clark in PROLOG about semantics as far as correctness is concerned?

Thanks to:

- ▶ Alan Smaill and Sanjay Poria (Edinburgh 1996)
- ▶ **none** to the LOPSTR PC of that year
- ▶ Conor McBride (Edinburgh 1995–Strathclyde, present day)
- ▶ you all for listening

Questions?

Correctness Proof: Setup

Say a subset of nodes X of the graph is:

- ▶ *sound*, iff $X \subseteq R^*S$
- ▶ *complete*, iff $R^*S \subseteq X$, for which it is sufficient that
 - ▶ $S \subseteq X$, and
 - ▶ $R^*X \subseteq X$.

Given V, W subsets of nodes, say that X is

- ▶ (V, W) -*complete*, iff
 - ▶ $V \cup W \subseteq X$, and
 - ▶ $R^*X \subseteq X$.

Say that (V, W) is

- ▶ *Dijkstra*, iff
 - ▶ $V \cap W \subseteq \emptyset$
 - ▶ $R^*V \subseteq V \cup W$

Correctness Proof: Setup

Say a subset of nodes X of the graph is:

- ▶ *sound*, iff $X \subseteq R^*S$
- ▶ *complete*, iff $R^*S \subseteq X$, for which it is sufficient that
 - ▶ $S \subseteq X$, and
 - ▶ $R^*X \subseteq X$.

Given V, W subsets of nodes, say that X is

- ▶ (V, W) -*complete*, iff
 - ▶ $V \cup W \subseteq X$, and
 - ▶ $R^*X \subseteq X$.

Say that (V, W) is

- ▶ *Dijkstra*, iff
 - ▶ $V \cap W \subseteq \emptyset$
 - ▶ $R^*V \subseteq V \cup W$

Correctness Proof: Setup

Say a subset of nodes X of the graph is:

- ▶ *sound*, iff $X \subseteq R^*S$
- ▶ *complete*, iff $R^*S \subseteq X$, for which it is sufficient that
 - ▶ $S \subseteq X$, and
 - ▶ $R^*X \subseteq X$.

Given V, W subsets of nodes, say that X is

- ▶ (V, W) -*complete*, iff
 - ▶ $V \cup W \subseteq X$, and
 - ▶ $R^*X \subseteq X$.

Say that (V, W) is

- ▶ *Dijkstra*, iff
 - ▶ $V \cap W \subseteq \emptyset$
 - ▶ $R^*V \subseteq V \cup W$

Correctness Proof: induction, induction, induction

By definition (almost)

- ▶ \emptyset is sound; S is sound
- ▶ if X is (\emptyset, S) -complete, then X is complete
- ▶ (\emptyset, S) is Dijkstra; if (X, \emptyset) is Dijkstra, then $R * X \subseteq X$

The remainder of the proof consists of two lemmas proved by induction on *Reachable* $V W X$:

- ▶ if *Reachable* $V W X$, and V, W are both sound, then X is sound
- ▶ if *Reachable* $V W X$, and (V, W) is Dijkstra, then X is (V, W) -complete

and a termination proof given by well-founded induction on the measure $||V|| = |G| - |V|$:

- ▶ for all V, W , if (V, W) is Dijkstra, then there exists an X satisfying *Reachable* $V W X$

Correctness Proof: induction, induction, induction

By definition (almost)

- ▶ \emptyset is sound; S is sound
- ▶ if X is (\emptyset, S) -complete, then X is complete
- ▶ (\emptyset, S) is Dijkstra; if (X, \emptyset) is Dijkstra, then $R * X \subseteq X$

The remainder of the proof consists of two lemmas proved by induction on $Reachable\ V\ W\ X$:

- ▶ if $Reachable\ V\ W\ X$, and V, W are both sound, then X is sound
- ▶ if $Reachable\ V\ W\ X$, and (V, W) is Dijkstra, then X is (V, W) -complete

and a termination proof given by well-founded induction on the measure $||V|| = |G| - |V|$:

- ▶ for all V, W , if (V, W) is Dijkstra, then there exists an X satisfying $Reachable\ V\ W\ X$

Correctness Proof: induction, induction, induction

By definition (almost)

- ▶ \emptyset is sound; S is sound
- ▶ if X is (\emptyset, S) -complete, then X is complete
- ▶ (\emptyset, S) is Dijkstra; if (X, \emptyset) is Dijkstra, then $R * X \subseteq X$

The remainder of the proof consists of two lemmas proved by induction on $Reachable\ V\ W\ X$:

- ▶ if $Reachable\ V\ W\ X$, and V, W are both sound, then X is sound
- ▶ if $Reachable\ V\ W\ X$, and (V, W) is Dijkstra, then X is (V, W) -complete

and a termination proof given by well-founded induction on the measure $||V|| = |G| - |V|$:

- ▶ for all V, W , if (V, W) is Dijkstra, then there exists an X satisfying $Reachable\ V\ W\ X$

Conditions on $rstep\ w\ V\ W$: abstracting the proof

The main ideas in the correctness of the iterative step, which computes the new waiting set $W' = rstep\ w\ V\ W$, and updates the visited set to $V' = \{w\} \cup V$, are:

- ▶ that $W' \subseteq R * \{w\} \cup W$; this is needed for *soundness*, so that only immediately reachable nodes, or those already inductively considered, will be examined;
- ▶ that $W \subseteq W'$; this is needed for *completeness*, and ensures that no previously considered node will be 'forgotten';
- ▶ that if $V \cap (\{w\} \cup W) \subseteq \emptyset$, then $V' \cap W' \subseteq \emptyset$; this maintains the first, and simpler, element of Dijkstra's invariant;
- ▶ finally, that the other half of the invariant is preserved:

$$R * V \subseteq V \cup (\{w\} \cup W) \implies R * \{w\} \subseteq V' \cup W'$$

Questions?