Increasingly Correct Scientific Computing

Cezar Ionescu

Scientific computing

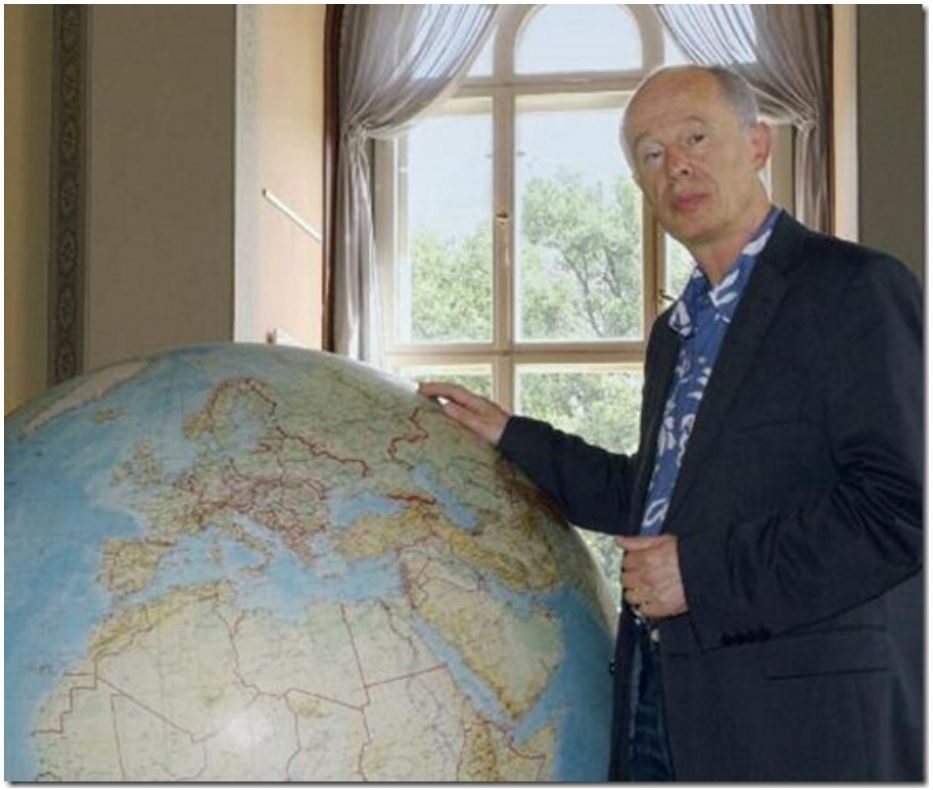
"the subfield of computer science concerned with constructing mathematical models and quantitative analysis techniques and using computers to analyze and solve scientific problems.

Wikipedia, retrieved on 2012-07-09

"is distinguished from most other parts of computer science in that it deals with quantities that are continuous, as opposed to discrete. It is concerned with functions and equations whose underlying variables–time, distance, velocity, temperature, density, pressure, stress, and the like–are continuous in nature."

M. Heath *Scientific Computing. An Introductory Survey*, 1997

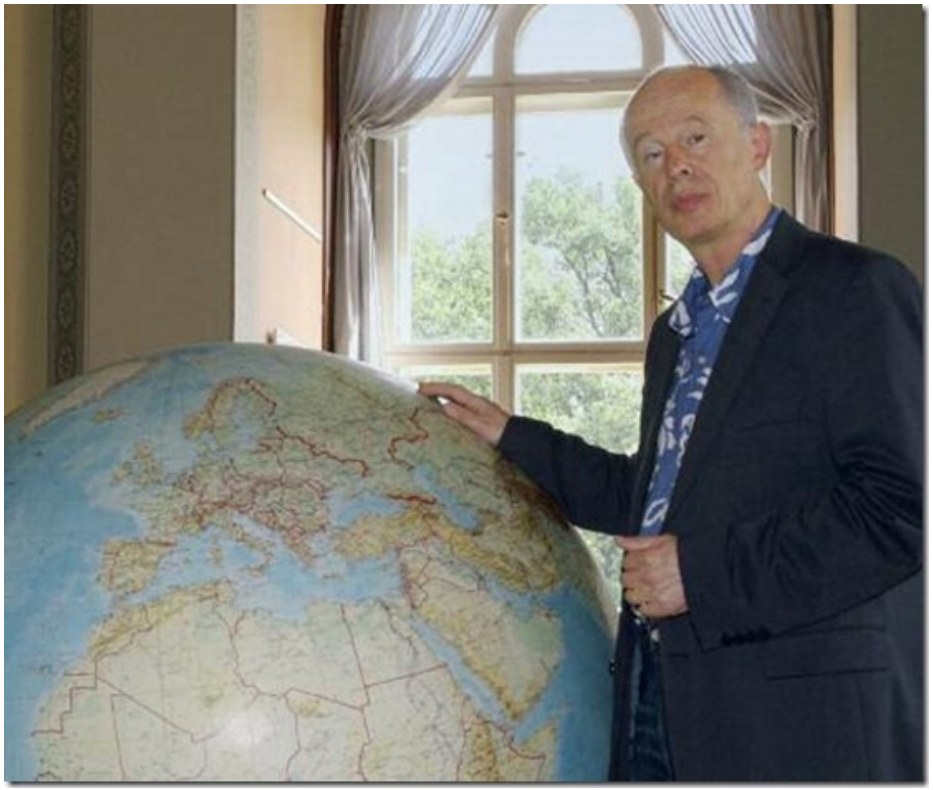
The Potsdam Institute for Climate Impact Research



"Die Rolle der Klimaforschung bleibt weiterhin, die Problemfakten auf den Tisch zu knallen und Optionen für geeignete Lösungswege zu identifizieren."

H.-J. Schellnhuber in *Frankfurter Allgemeine* from 2012-06-19

The Potsdam Institute for Climate Impact Research



"The role of the climate researcher continues to be to slam the hard facts on the table and to indicate the possible solutions to the problems".

H.-J. Schellnhuber in *Frankfurter Allgemeine* from 2012-06-19

The Potsdam Institute for Climate Impact Research

PIK addresses crucial scientific questions in the fields of global change, climate impacts and sustainable development.

Researchers from the natural and social sciences work together to generate interdisciplinary insights and to provide society with sound information for decision making.

The main methodologies are systems and scenarios analysis, modelling, computer simulation, and data integration.

PIK Mission, www.pik-potsdam.de, retrieved 2012-07-09

The Potsdam Institute for Climate Impact Research

PIK addresses crucial scientific questions in the fields of global change, climate impacts and sustainable development.

Researchers from the natural and social sciences work together to generate interdisciplinary insights and to provide society with sound information for decision making.

The main methodologies are systems and scenarios analysis, modelling, computer simulation, and data integration.

PIK Mission, www.pik-potsdam.de, retrieved 2012-07-09

The Potsdam Institute for Climate Impact Research

PIK addresses crucial scientific questions in the fields of global change, climate impacts and sustainable development.

Researchers from the natural and social sciences work together to generate interdisciplinary insights and to provide society with sound information for decision making.

The main methodologies are systems and scenarios analysis, modelling, computer simulation, and data integration.

PIK Mission, www.pik-potsdam.de, retrieved 2012-07-09

The Potsdam Institute for Climate Impact Research

PIK addresses crucial scientific questions in the fields of global change, climate impacts and sustainable development.

Researchers from the natural and social sciences work together to generate interdisciplinary insights and to provide society with sound information for decision making.

The main methodologies are systems and scenarios analysis, modelling, computer simulation, and data integration.

PIK Mission, www.pik-potsdam.de, retrieved 2012-07-09

Computer simulation

"Simulation is a third way of doing science. Like deduction, it starts with a set of explicit assumptions. But unlike deduction, it does not prove theorems.

Instead, a simulation generates data that can be analyzed inductively. Unlike typical induction, however, the simulated data comes from a rigorously specified set of rules rather than direct measurement of the real world.

R. Axelrod *Advancing the Art of Simulation in the Social Sciences*, 2003

Computer simulation

"Simulation is a third way of doing science. Like deduction, it starts with a set of explicit assumptions. But unlike deduction, it does not prove theorems.

Instead, a simulation generates data that can be analyzed inductively. Unlike typical induction, however, the simulated data comes from a rigorously specified set of rules rather than direct measurement of the real world.

R. Axelrod *Advancing the Art of Simulation in the Social Sciences*, 2003

Correctness of computer simulations

The correctness of a computer simulation therefore depends on

- having explicit assumptions
- having rigorous rules to generate data
- some relationship between the two

Sometimes, these conditions are not met...

Basic economics: models of exchange

The quintessential economic situation: exchange of goods.

1. Two agents, two goods, X units of the first good, Y units of the second.
2. Agent i has x_i unit of the first good, and y_i units of the second.
3. A distribution of goods to agents, such as $((x_1, y_1), (x_2, y_2))$ is called an *allocation*. Agents have preferences over allocations.
4. Agents are allowed to exchange their goods in order to find a better allocation $((x'_1, y'_1), (x'_2, y'_2))$. Only *feasible* allocations are acceptable: $x'_1 + x'_2 = X$, $y'_1 + y'_2 = Y$.

What is a good allocation?

(Weak) Pareto efficiency

A feasible allocation **x** is a **Pareto efficient** allocation if there is no feasible allocation $\mathbf{x}'$ such that all agents strictly prefer $\mathbf{x}'$ to **x**.

Varian, *Microeconomic Analysis*, p. 323

An allocation x is Pareto efficient, if there exists no feasible allocation that dominates it *strictly everywhere*.

Example: Cobb-Douglas economy

A typical example is the Cobb-Douglas economy, in which the agents preferences induced by the utility functions

$$u_1(x, y) = x^{a_1} y^{(1-a_1)}$$

$$u_2(x, y) = x^{a_2} y^{(1-a_2)}$$

where $a_1, a_2 \in (0,\ 1)$.

Introducing prices

If goods have prices p_x, p_y then an initial allocation gives each agent a *budget*:

$$B_i = p_x x_i + p_y y_i.$$

An agent has to solve:

maximize $u(x, y)$ such that

$$p_x x + p_y y = B_i$$

Whether the resulting allocation is feasible depends on the prices.

Walrasian equilibrium

An allocation-price pair

$$((x_1^*, y_1^*), (x_2^*, y_2^*)), (p_x, p_y)$$

is a **Walrasian equilibrium** if (1) the allocation is feasible, and (2) each agent is making an optimal choice from its budget set:

1. $x_1^* + x_2^* = X$, $y_1^* + y_2^* = Y$
2. If $u_i(x_i', y_i') > u_i(x_i^*, y_i^*)$, then $p_x x_i' + p_y y_i' > B_i$

Varian, *Microeconomic Analysis*, p. 325

First welfare theorem: Walrasian equilibria are Pareto efficient.

Mainstream economics

Refinements

- several agents
- production and consumption
- iterated exchanges
- introduce agents representing banks, governments, ...
- ...

Most of the models used for policy advice are based on extensions of this idea.

Mainstream economics

Refinements

- several agents
- production and consumption
- iterated exchanges
- introduce agents representing banks, governments, ...
- ...

Most of the models used for policy advice are based on extensions of this idea.
Problems:

Mainstream economics

Refinements

- several agents
- production and consumption
- iterated exchanges
- introduce agents representing banks, governments, ...
- ...

Most of the models used for policy advice are based on extensions of this idea.
Problems:

- Where do prices come from?

Mainstream economics

Refinements

- several agents
- production and consumption
- iterated exchanges
- introduce agents representing banks, governments, ...
- ...

Most of the models used for policy advice are based on extensions of this idea.
Problems:

- Where do prices come from?
- Which equilibria get selected?

Multi-agent models

Multi-agent models are models in which agents interact with each other directly.

There is no central mechanism that solves the optimization problem and gives all agents their due.

Multi-agent models attempt to make the process of equilibrium selection understandable.

The Gintis model

"We thus provide, for the first time, a general, decentralized disequilibrium adjustment mechanism that renders market equilibrium dynamically stable in a highly simplified production and exchange economy."

"Our results should be considered empirical rather than theoretical: we have created a class of economies and investigated their properties for a range of parameters."

Herbert Gintis *The Emergence of a Price System from Decentralized Bilateral Exchange*, 2006

The Gintis model, ctd.

At PIK, the interest was fueld by the Lagom project:

> "The model has provided the conceptual basis for two major studies commissioned by the German ministry for the Environment, the first assessing the economic implications of German climate policy, the second designing sustainable answers to the financial crisis."
>
> From the homepage of the *Lagom* project,

In 2009, Mandel and Botta proved results for a simplified model with stronger assumptions. Many features of the Gintis model resisted mathematical analysis, and reproduction of the results failed.

The Gintis model, ctd.

Independently, Pelle Evensen and Mait Märdin investigated the model and published results in *An Extensible and Scalable Agent-Based Simulation of Barter Economics* M.Sc. Thesis, Chalmers 2009.
Both groups discovered a serious bug in the implementation:

$$\frac{\sum_j p_{ij} x_{ij}}{\sum_j p_{ij} o_j}$$

was implemented as

$$\frac{\sum_j p_{ij} x_{ij}}{\sum_j p_{ij} x_{ij}}$$

This led to less variance in the computation of prices, and consequently to fast convergence.

The Gintis model, ctd.

Main problem: the "explicit hypothesis" were ambiguous, and the relationship to the code unclear.

> "The discrepancies between the description and the original implementation of the barter economy confirm the importance of replication."
>
> Evensen and Märdin, 2009

> "In practice, however, model re-implementation on the basis of narrative descriptions is nearly impossible. For consistent, independent model re-implementation, one needs unambiguous mathematical specifications."
>
> Botta et. al. *A functional framework for agent-based models of exchange*, 2011

Specifications in scientific computing

We need *specifications* that

- ensure that "explicit hypothesis" and the "rigorously specified set of rules" are not contradicting each other
- allow checking correctness of implementations, model re-implementation, replication of results, etc.

We found little advice on specifications in scientific computing (e.g. *Writing Scientific Software – A Guide to Good Style* (Oliveira and Stewart, 2006) doesn't address specifications).

In many cases, the mathematical descriptions of their problems and algorithms are insufficient as specifications.

Example: GEM-E3

GEM-E3 is an applied general equilibrium model that covers the interactions between the Economy, the Energy system and the Environment. It is well suited to evaluate climate and energy policies, as well as fiscal issues.

The GEM-E3 model has been used for several Directorates General of the European Commission, as well as for national authorities. The GEM-E3 modelling groups are also partner in several research projects, and analyses based on GEM-E3 have been published widely.

GEM-E3 website, retrieved 2012-07-09

GEM-E3 household specification

"The general specification [. . .] can be written as follows:

$$\max U(q(t)) = \int_{t=0}^{\infty} e^{-\delta t} u(q(t)) dt$$

where . . . "

GEM-E3 reference manual p. 13

GEM-E3 household specification

"The general specification [...] can be written as follows:

$$\max U(q(t)) = \int_{t=0}^{\infty} e^{-\delta t} u(q(t)) dt$$

where ..."

GEM-E3 reference manual p. 13

But in the code ...

GEM-E3 household implementation

- Continuous time has been replaced by discrete time.
- The infinite horizon has been replaced by a finite horizon.
- Therefore, the integral to be maximzed has been replaced by a finite sum.
- The maximization has been replaced with the necessary (but not sufficient) first-order conditions.
- ...

Many of these steps are explained in the GEM-E3 manual, but not in a way which would allow re-implementation of the model.

Constructive mathematics

The gap between mathematics and programming is too large and we need to bridge it.

Constructive mathematics

The gap between mathematics and programming is too large and we need to bridge it.

> "Now, it is the contention of the intuitionists (or constructivists, I shall use these terms synonymously) that the basic mathematical notions, above all the notion of function, ought to be interpreted in such a way that the cleavage between mathematics, classical mathematics, that is, and programming that we are witnessing at present disappears."
>
> P. Martin-Löf, *Constructive Mathematics and Computer Programming*, 1984

Constructive mathematics and type theory

"[Type theory] provides a precise notation not only, like other programming languages, for the programs themselves but also for the tasks that the programs are supposed to perform.

Thus the correctness of a program written in the theory of types is proved formally at the same time as it is being synthesized."

P. Martin-Löf, *Constructive Mathematics and Computer Programming*, 1984

Good news

We tested the expressive power of type theory by formalizing different equilibria in Agda and Idris, together with the relationships betwen them.

We could write specifications for certain kinds of economic agents in Ginits-like models.

We had several sessions with Lagom modelers, and they found the specifications understandable.

Walrasian equilibrium in (old) Idris

```
params (omega  : Vect (Vect Float nG) nA,
        prices : Vect Float nG,
        prefs  : Fin nA -> TotalPreorder (Vect Float nG)) {

  Feasible      : Vect (Vect Float nG) nA -> Set;
  Feasible xss  = SumCols xss = SumCols omega;

  Optimal       : Vect (Vect Float nG) nA -> Set;
  Optimal xss = (i : Fin nA, xss' : Vect (Vect Float nG) nA) ->
                gt (prefs i) (index i xss')
                             (index i xss)                    ->
                gt floatOrder (prices .* (index i xss'))
                              (prices  .* (index i xss));

  WalrasEq      : Vect (Vect Float nG) nA -> Set;
  WalrasEq xss = (Feasible xss, Optimal xss);
```

Walrasian equilibrium, revisited

Even with an established text and elementary concepts, there are still surprises.

An allocation-price pair

$$((x_1^*, y_1^*), (x_2^*, y_2^*)), (p_x, p_y)$$

is a **Walrasian equilibrium** if (1) the allocation is feasible, and (2) each agent is making an optimal choice from its budget set:

1. $x_1^* + x_2^* = X$, $y_1^* + y_2^* = Y$
2. If $u_i(x_i', y_i') > u_i(x_i^*, y_i^*)$, then $p_x x_i' + p_y y_i' > B_i$

Varian, *Microeconomic Analysis*, p. 325

Question: Is $p_x x_i^* + p_y y_i^* = B_i$ necessarily true?

Bad news

Therefore, it appears that we can express the "explicit hypothesis" and the "rules" that drive our simulations...

Bad news

Therefore, it appears that we can express the "explicit hypothesis" and the "rules" that drive our simulations...

but not the relationship between them.

- Economic theory is mostly non-constructive (K. Vellupilai, 2002): the divide between mathematical specification and implementations is still there.
- Most modelers are not numerical analysts: they want to use external routines.
- No usable library of numerical methods for constructive reals.
- (Some) modelers are willing to write formal specifications, but less willing to write formal proofs, let alone *constructive* formal proofs.

Good news

Having specifications is better than having no specifications.

Having specifications which can be partially machine-checked is better than having specifications which cannot be machine-checked at all.

Having classical proofs of correctness is better than having no proofs of correctness.

Using type theory for specifications can also guide the efforts of the constructive mathematics community.

And so on: just because we cannot now have fully verified models should not prevent us from taking advantage of what we have!

Some Idris datatypes

The datatype of bounded numbers in Idris:

```
data Fin : Nat → Set where
     fO  : Fin (S k)
     fS  : Fin k → Fin (S k)
```

Finite-sized lists:

```
data Vect : Set → Nat → Set where
     Nil  : Vect a O
     (::) : a → Vect a n → Vect a (S n)
```

Some *Fin* functions

Bounding a natural number:

$$\begin{aligned} &\mathit{toFin} && : (n : \mathit{Nat}) \to \mathit{Fin}\ (S\ n) \\ &\mathit{toFin}\ O && = \mathit{fO} \\ &\mathit{toFin}\ (S\ n) && = \mathit{fS}\ (\mathit{toFin}\ n) \end{aligned}$$

Canonical embedding:

$$\begin{aligned} &\mathit{next} && : (t : \mathit{Fin}\ n) \to \mathit{Fin}\ (S\ n) \\ &\mathit{next}\ \mathit{fO} && = \mathit{fO} \\ &\mathit{next}\ (\mathit{fS}\ t) && = \mathit{fS}\ (\mathit{next}\ t) \end{aligned}$$

Maximizing utility over a finite set

We want

$$max\ :\ (Fin\ (S\ n)\ \rightarrow\ Float)\ \rightarrow\ (Fin\ (S\ n), Float)$$

such that

$$\begin{aligned} maxSpec\ :\ & (u\ :\ Fin\ (S\ n)\ \rightarrow\ Float)\ \rightarrow \\ & (i\ :\ Fin\ (S\ n)) \qquad\qquad\ \rightarrow \\ & (u\ (fst\ (max\ u)) = snd\ (max\ u), \\ & \ u\ i \leqslant snd\ (max\ u)) \end{aligned}$$

Maximizing utility over a finite set

$$
\begin{aligned}
&max\ :(Fin\ (S\ n)\ \rightarrow\ Float)\ \rightarrow\ (Fin\ (S\ n), Float)\\
&max\ \{n = O\}\ u\quad\ = (fO, u\ fO)\\
&max\ \{n = S\ m\}\ u = max'\ u\ (fO, u\ fO)\ fO
\end{aligned}
$$

$$
\begin{aligned}
&max'\ \{n\}\ u\ (best, bestU)\ c' =\\
&\quad \textbf{let}\ c\ =\ fS\ c'\ \textbf{in}\quad \text{-- c is the candidate}\\
&\quad \textbf{let}\ uc =\ u\ c\ \textbf{in}\\
&\qquad \textbf{case}\ c == toFin\ n\ \textbf{of}\quad \text{-- c is the last candidate}\\
&\qquad\quad True\ \Rightarrow \textbf{if}\ uc \leqslant bestU\ \textbf{then}\ (best, bestU)\\
&\qquad\qquad\qquad\qquad \textbf{else}\ (c, uc)\\
&\qquad\quad False \Rightarrow \textbf{if}\ uc \leqslant bestU\\
&\qquad\qquad\qquad\qquad \textbf{then}\ max'\ u\ (best, bestU)\ c\\
&\qquad\qquad\qquad\qquad \textbf{else}\ \ max'\ u\ (c, uc)\ c
\end{aligned}
$$

Maximizing utility over a finite set

$$
\begin{aligned}
&max\ :(Fin\ (S\ n)\ \rightarrow\ Float)\ \rightarrow\ (Fin\ (S\ n), Float)\\
&max\ \{n = O\}\ u\quad\ = (fO, u\ fO)\\
&max\ \{n = S\ m\}\ u = max'\ u\ (fO, u\ fO)\ fO
\end{aligned}
$$

$$
\begin{aligned}
&max'\ \{n\}\ u\ (best, bestU)\ c' =\\
&\quad \textbf{let}\ c\ =\ fS\ c'\ \textbf{in}\quad \text{-- c is the candidate}\\
&\quad \textbf{let}\ uc =\ u\ c\ \textbf{in}\\
&\qquad \textbf{case}\ c == toFin\ n\ \textbf{of}\quad \text{-- c is the last candidate}\\
&\qquad\quad True\ \Rightarrow \textbf{if}\ uc \leqslant bestU\ \textbf{then}\ (best, bestU)\\
&\qquad\qquad\qquad\qquad \textbf{else}\ (c, uc)\\
&\qquad\quad False \Rightarrow \textbf{if}\ uc \leqslant bestU\\
&\qquad\qquad\qquad\qquad \textbf{then}\ max'\ u\ (best, bestU)\ c\quad \text{-- !}\\
&\qquad\qquad\qquad\qquad \textbf{else}\ \ max'\ u\ (c, uc)\ c\quad \text{-- !}
\end{aligned}
$$

Maximizing utility over a finite set

$$
\begin{array}{lll}
max' & : & (Fin\ (S\ n) \to Float) \to \quad \text{-- utility} \\
 & & (Fin\ (S\ n), Float) \to \quad \text{-- best-so-far} \\
 & & Fin\ n \to \quad \text{-- count / candidate} \\
 & & (Fin\ (S\ n), Float) \quad \text{-- optimum}
\end{array}
$$

$$
\begin{array}{l}
max'\ \{n\}\ u\ (best, bestU)\ c' = \\
\quad \textbf{let}\ c = fS\ c'\ \textbf{in} \quad \text{-- c is the candidate} \\
\quad \textbf{let}\ uc = u\ c\ \textbf{in} \\
\quad\quad \textbf{case}\ c == toFin\ n\ \textbf{of} \quad \text{-- c is the last candidate} \\
\quad\quad\quad True \Rightarrow \textbf{if}\ uc \leqslant bestU\ \textbf{then}\ (best, bestU) \\
\quad\quad\quad\quad\quad\quad \textbf{else}\ (c, uc) \\
\quad\quad\quad False \Rightarrow \textbf{if}\ uc \leqslant bestU \\
\quad\quad\quad\quad\quad\quad \textbf{then}\ max'\ u\ (best, bestU)\ c \quad \text{-- !} \\
\quad\quad\quad\quad\quad\quad \textbf{else}\ \ max'\ u\ (c, uc)\ c \quad \text{-- !}
\end{array}
$$

Maximizing utility over a finite set

```
forceEmbed  : Fin (S n) → Fin n
forceEmbed i = ?

max′ {n} u (best, bestU) c′ =
  let c  = fS c′ in    -- c is the candidate
  let uc = u c in
    case c == toFin n of    -- c is the last candidate
      True  ⇒ if uc ⩽ bestU then (best, bestU)
                            else (c, uc)
      False ⇒ if uc ⩽ bestU
                            then max′ u (best, bestU) (forceEmbed c)
                            else max′ u (c, uc) (forceEmbed c)
```

Maximizing utility over a finite set

```
forceEmbed : Fin (S n) → Fin n
forceEmbed i = believe_me i

max' {n} u (best, bestU) c' =
  let c = fS c' in    -- c is the candidate
  let uc = u c in
    case c == toFin n of    -- c is the last candidate
      True ⇒ if uc ⩽ bestU then (best, bestU)
                           else (c, uc)
      False ⇒ if uc ⩽ bestU
                           then max' u (best, bestU) (forceEmbed c)
                           else max' u (c, uc) (forceEmbed c)
```

Programming style

How do we specify that the outputs a program $X \to Y$ have to be in the relation R with the inputs?

Nordström et. al.:

$$f : (x : X) \to (y : Y ** R\,(x, y))$$

Thompson:

$$(f : X \to Y ** (x : X) \to R\,(x, f\ x))$$

Generic programming

Less code, fewer errors: generic programming.

Dependently-typed programming languages are good at generic programming.

Example: dynamic programming for sequential decision problems.

ReMIND-R

ReMIND-R is a global multi-regional model incorporating the economy, the climate system and a detailed representation of the energy sector.

It solves for an inter-temporal Pareto optimum in economic and energy investments in the model regions, fully accounting for interregional trade in goods, energy carriers and emissions allowances.

ReMIND-R allows for the analysis of technology options and policy proposals for climate mitigation.

ReMIND-R stands for 'Refined Model of Investments and Technological Development - Regionalized' and it is programmed in GAMS.

ReMIND-R home page, retrieved 2012-07-09

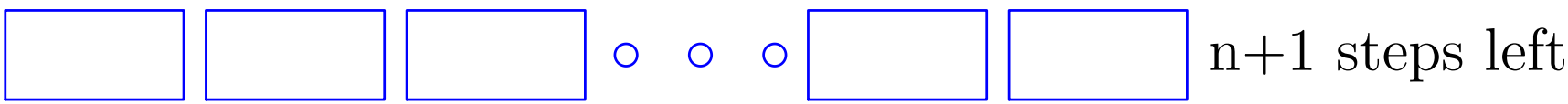
ReMIND-R, ctd.

The intertemporal social welfare function:

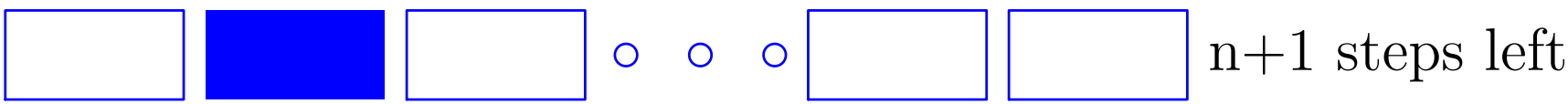
$$U = \sum_{r} \left(W(r) \sum_{t=t_0}^{t_{end}} \left(\Delta t \cdot e^{-\zeta(r)(t-t_0)} \tilde{U}(t,r) \right) \right)$$

from *ReMIND-R – the Equations*

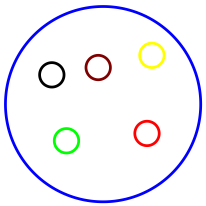
Sequential decision problems

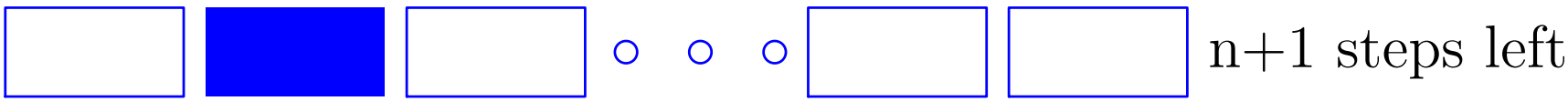


You are here. . .

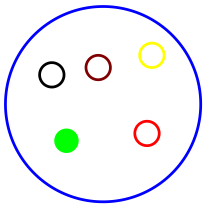


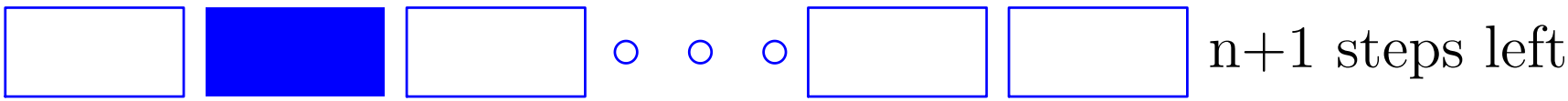
These are your options. . .



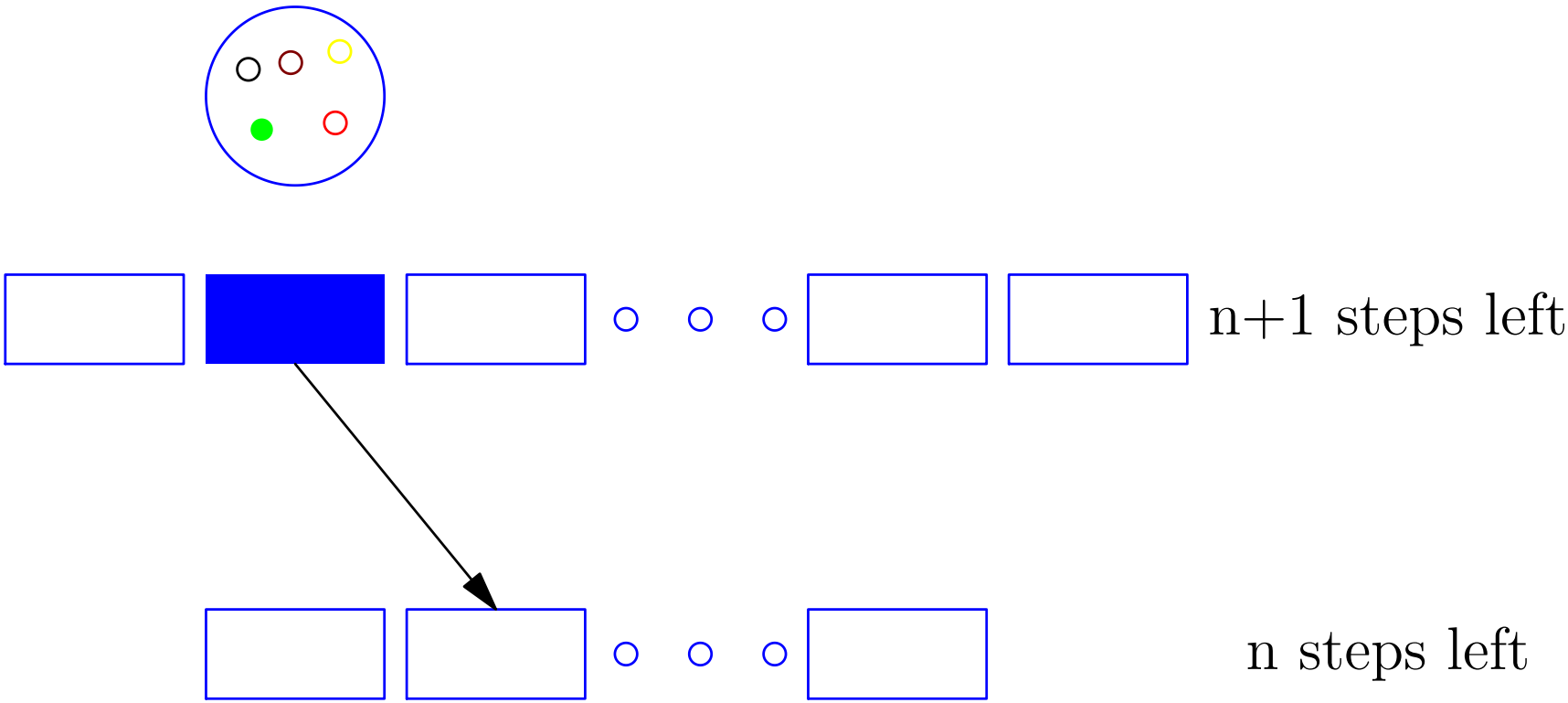


Pick one!

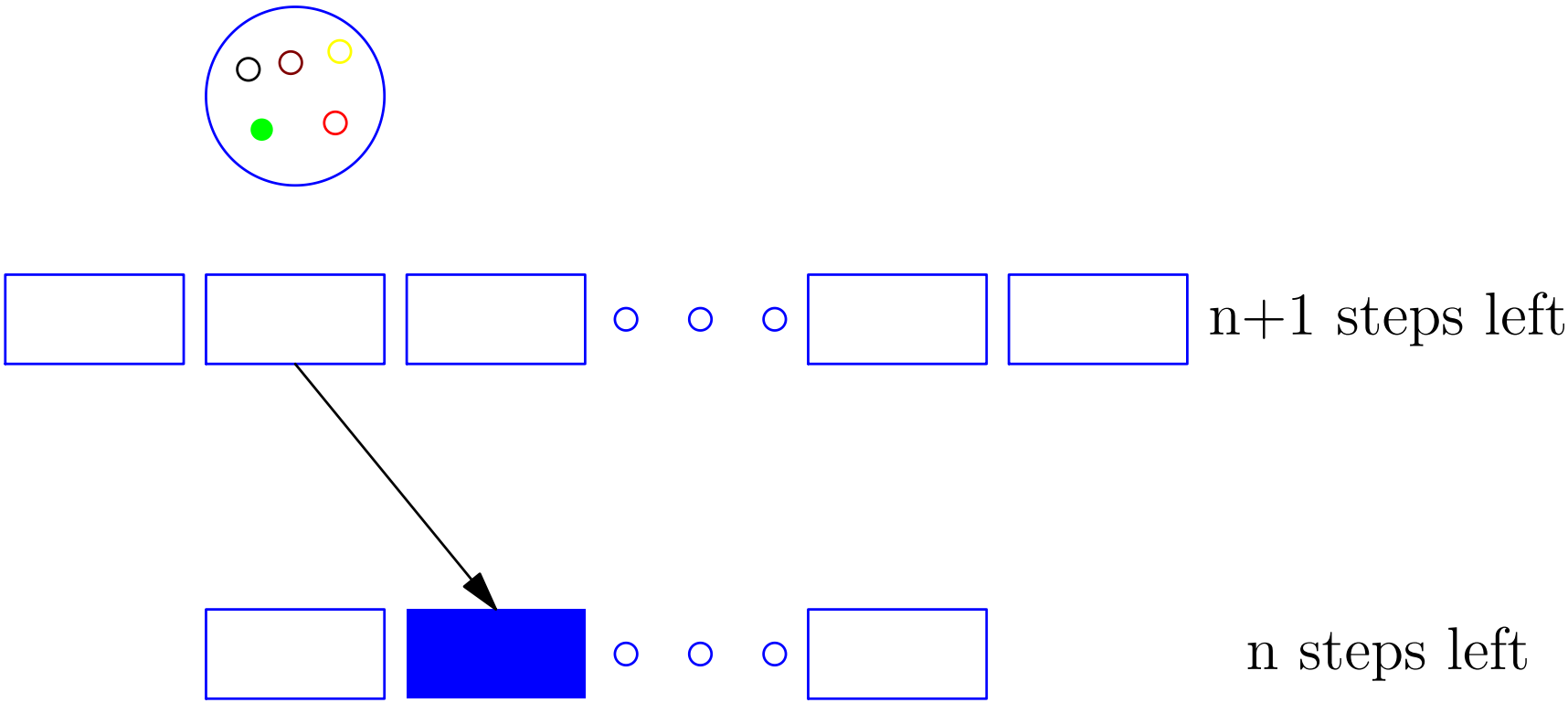




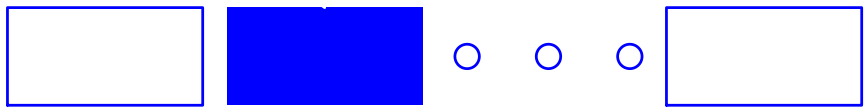
Advance one step. . .



. . . collect. . .

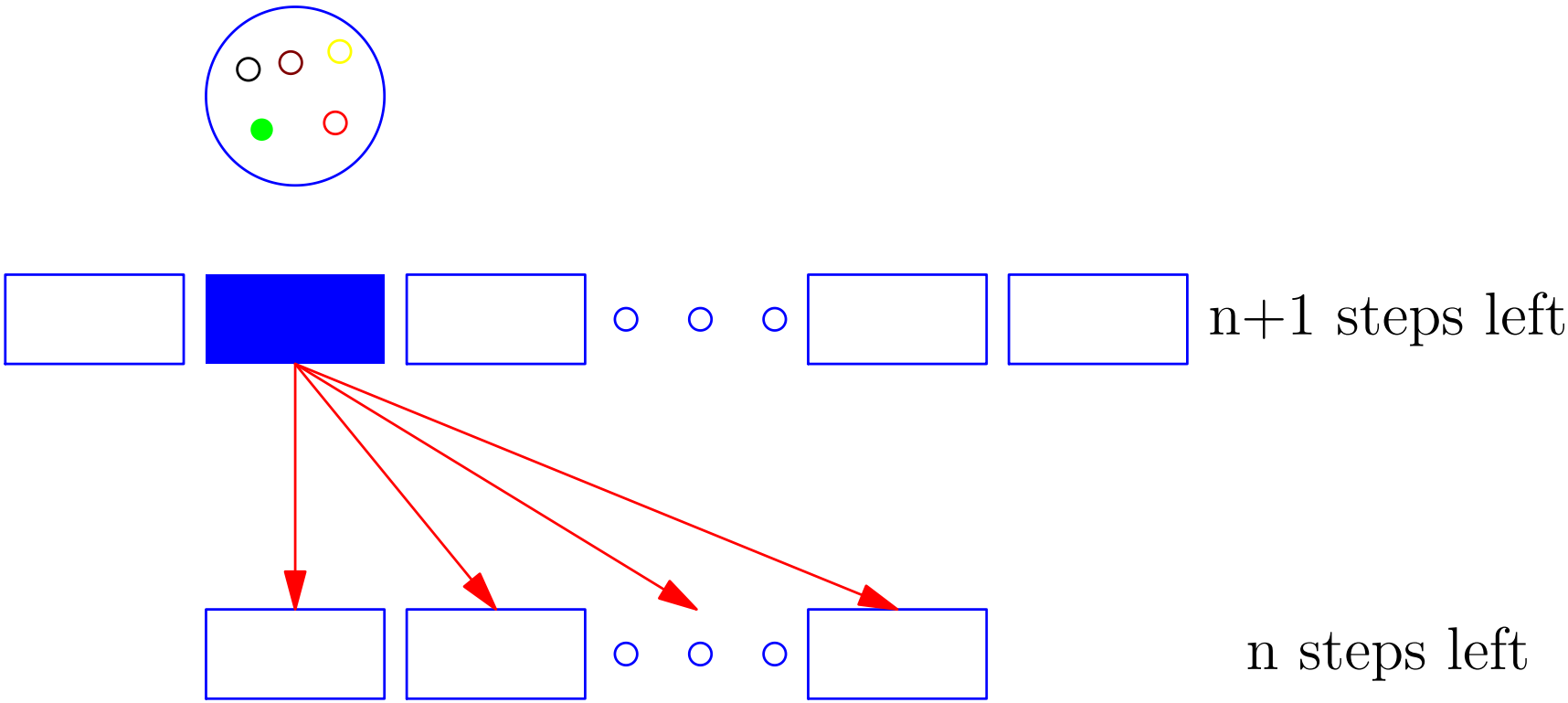


. . . and go!



n steps left

General sequential decision problems



Formalizing the deterministic case

$$NrSteps = S\ NrSteps'$$

$$State \quad : \quad Fin\ NrSteps \ \rightarrow \ Set$$

$$Ctrl \quad : \quad State\ (fS\ t) \ \rightarrow \ Set$$

$$step \quad : \quad (s \ : \ State\ (fS\ t)) \ \rightarrow \ (c \ : \ Ctrl\ s) \ \rightarrow \ State\ (next\ t)$$

$$reward \quad : \quad (s \ : \ State\ (fS\ t)) \ \rightarrow \ (c \ : \ Ctrl\ s) \ \rightarrow \ (s' \ : \ State\ (next\ t)) \ \rightarrow \ Float$$

Off-by-one error?

The intertemporal social welfare function in ReMIND-R:

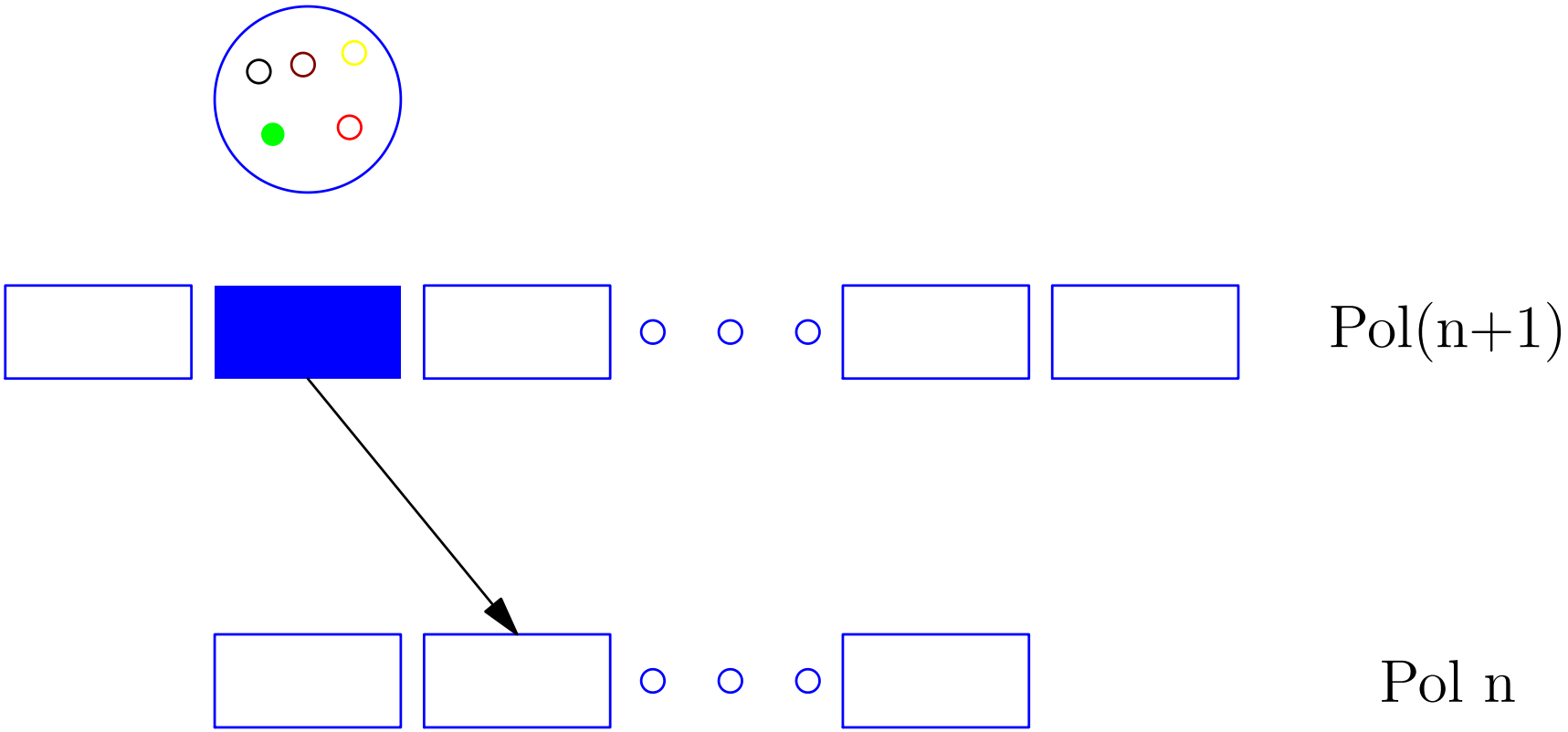
$$U = \sum_r \left(W(r) \sum_{t=t_0}^{t_{end}} \left(\Delta t \cdot e^{-\zeta(r)(t-t_0)} \tilde{U}(t,r) \right) \right)$$

For $t_{end} = t_0$ we have:

$$U = \sum_r \left(W(r) \left(\Delta t \cdot \tilde{U}(t_0,r) \right) \right)$$

In order to compute $\tilde{U}(t_0, r)$ we need data for times t_0 and t_1.

Formalizing policies



Formalizing the deterministic case, ctd.

A policy is a function of type

$$policy\ :\ (t\ :\ Fin\ NrSteps')\ \rightarrow\ (s\ :\ State\ (fS\ t))\ \rightarrow\ Ctrl\ s$$

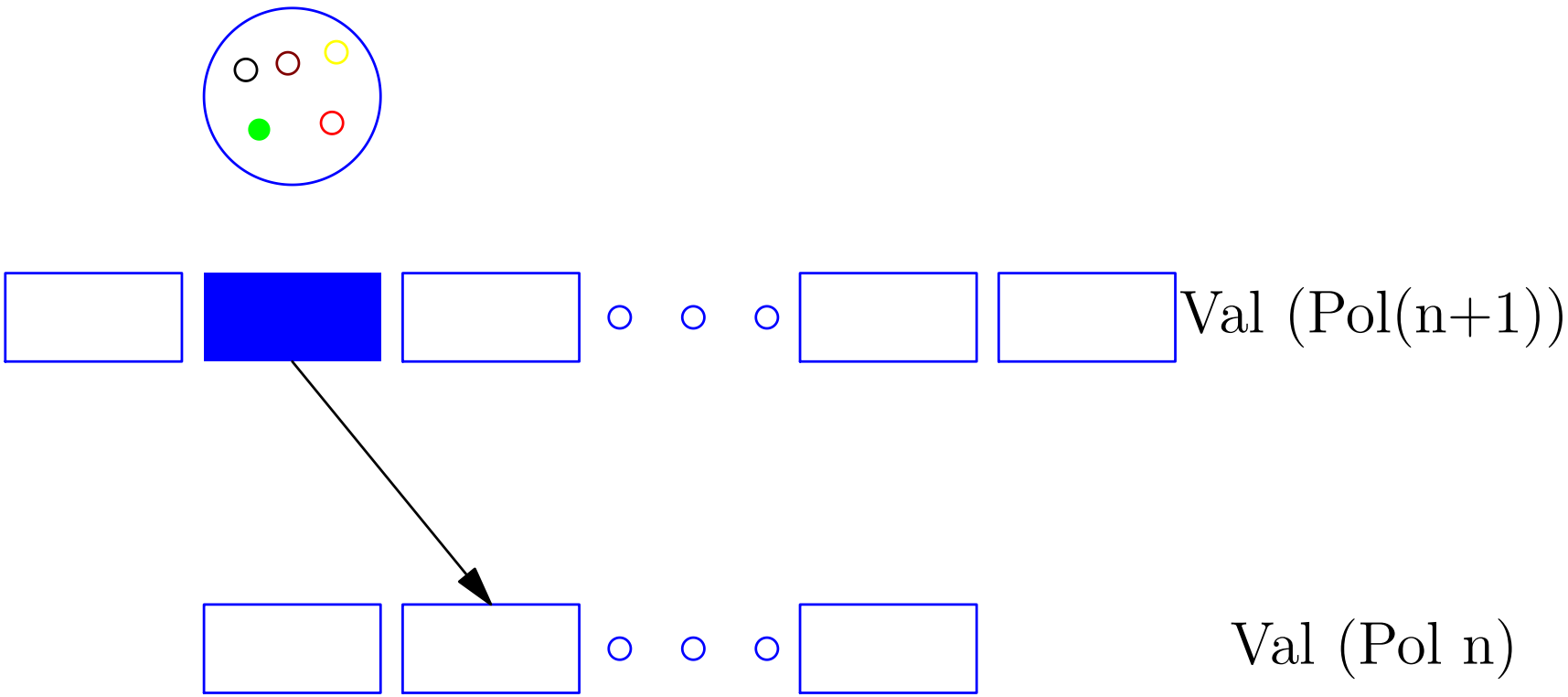
We can take sections along the number of steps to be done:

$$\begin{aligned} LocalPol\ &:\ (t\ :\ Fin\ NrSteps')\ \rightarrow\ Set \\ LocalPol\ t\ &=\ (s\ :\ State\ (fS\ t))\ \rightarrow\ Ctrl\ s \end{aligned}$$

We can construct a "vector" of local policies:

$$\begin{aligned} &\textbf{data}\ Pol\ :\ (t\ :\ Fin\ NrSteps)\ \rightarrow\ Set\ \textbf{where} \\ &\quad Nil\ :\ Pol\ fO \\ &\quad Cons\ :\ LocalPol\ t\ \rightarrow\ Pol\ (next\ t)\ \rightarrow\ Pol\ (fS\ t) \end{aligned}$$

The value of a policy



Formalizing the deterministic case, ctd.

The value of a policy for a given state is the accumulated reward we get from that state by applying the policy to the end.

$$
\begin{array}{l}
\mathit{Val} \quad : (s : \mathit{State}\ t) \to \mathit{Pol}\ t \to \mathit{Float} \\
\mathit{Val}\ \_\ \mathit{Nil} = 0 \\
\mathit{Val}\ \{t = \mathit{fS}\ t'\}\ s\ (\mathit{Cons}\ \mathit{lp}\ \mathit{pols}) = \\
\quad \mathit{reward}\ s\ c\ s' \oplus \mathit{Val}\ s'\ \mathit{pols} \\
\quad \textbf{where} \\
\quad c \ : \mathit{Ctrl}\ s \\
\quad c \ = \mathit{lp}\ s \\
\quad s' \ : \mathit{State}\ (\mathit{next}\ t') \\
\quad s' = \mathit{step}\ s\ c
\end{array}
$$

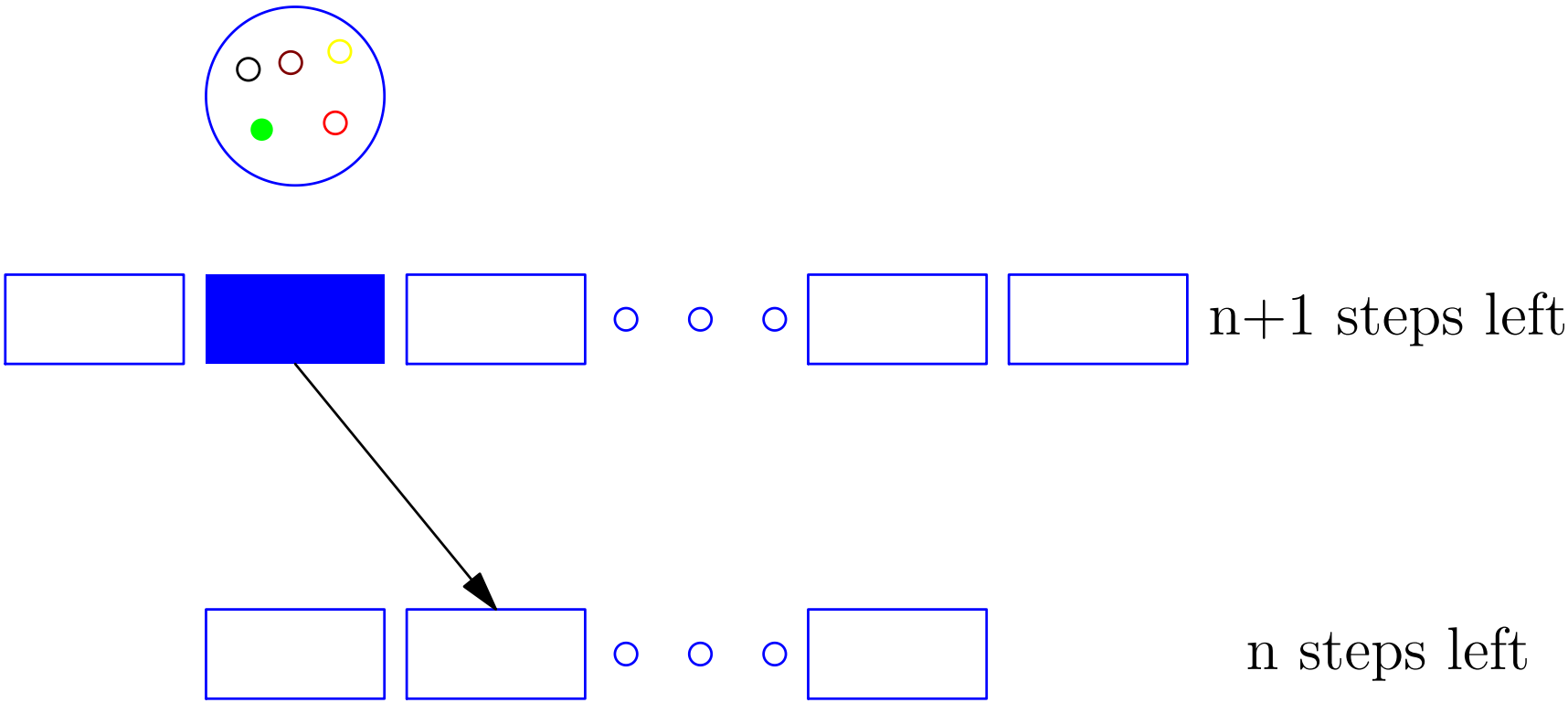
Formalizing the deterministic case, ctd.

A policy for t steps is optimal if it is better than all other alteratives for all possible matching states.

$$Opt \quad : \; Pol\ t \;\to\; Set$$

$$Opt\ \{t\}\ pol = (pol' \; : \; Pol\ t) \;\to\; (s \; : \; State\ t) \;\to\; Val\ s\ pol' \leqslant Val\ s\ pol$$

Dynamic programming



Formalizing the deterministic case, ctd.

Optimal extension of a policy:

$$
\begin{aligned}
\mathit{OptExt} \quad &: \{t : \mathit{Fin}\ \mathit{NrSteps}'\} \to \\
&\quad (\mathit{lp} : \mathit{LocalPol}\ t) \to \\
&\quad (\mathit{pol} : \mathit{Pol}\ (\mathit{next}\ t)) \to \\
&\quad \mathit{Set}
\end{aligned}
$$

$$
\begin{aligned}
\mathit{OptExt}\ \{t\}\ \mathit{lp}\ \mathit{pol} = &\ (\mathit{lp}' : \mathit{LocalPol}\ t) \to \\
&\ (s : \mathit{State}\ (\mathit{fS}\ t)) \to \\
&\ \mathit{Val}\ s\ (\mathit{Cons}\ \mathit{lp}'\ \mathit{pol}) \leqslant \mathit{Val}\ s\ (\mathit{Cons}\ \mathit{lp}\ \mathit{pol})
\end{aligned}
$$

Dynamic programming, deterministic case ctd.

Bellman: an optimal extension of an optimal policy is optimal.

$$\begin{array}{lll} \mathit{Bellman} : \{t : \mathit{Fin}\ \mathit{NrSteps}'\} & & \to \\ (\mathit{pol} : \mathit{Pol}\ (\mathit{next}\ t)) & \to\ \mathit{Opt}\ \mathit{pol} & \to \\ (\mathit{lp} : \mathit{LocalPol}\ t) & \to\ \mathit{OptExt}\ \mathit{lp}\ \mathit{pol} & \to \\ \mathit{Opt}\ (\mathit{Cons}\ \mathit{lp}\ \mathit{pol}) & & \end{array}$$

For any $\mathit{lp}' : \mathit{LocalPol}\ t$ and $\mathit{pol}' : \mathit{Pol}\ (\mathit{next}\ t)$, we have for any $s : \mathit{State}\ (\mathit{fS}\ t)$

$$\mathit{Val}\ s\ (\mathit{Cons}\ \mathit{lp}'\ \mathit{pols}') \leqslant \mathit{Val}\ s\ (\mathit{Cons}\ \mathit{lp}\ \mathit{pols})$$

Dynamic programming, deterministic case ctd.

Proof:
Let $c' = lp'\ s$, $s' = step\ s\ c'$. We have

$$
\begin{aligned}
& Val\ s\ (Cons\ lp'\ pols') \\
= \quad & \{\ \text{by definition}\ \} \\
& reward\ s\ c'\ s' \oplus Val\ s'\ pols' \\
\leqslant \quad & \{\ \text{monotonicity of } \oplus, \textit{pol} \text{ optimal}\ \} \\
& reward\ s\ c'\ s' \oplus Val\ s'\ pols \\
= \quad & \{\ \text{by definition}\ \} \\
& Val\ s\ (Cons\ lp'\ pols) \\
\leqslant \quad & \{\ \text{lp optimal extension}\ \} \\
& Val\ s\ (Cons\ lp\ pols)
\end{aligned}
$$

Dynamic programming, deterministic case ctd.

Idris implementation:

$$
\begin{array}{l}
\mathit{Bellman} : \{t : \mathit{Fin}\ \mathit{NrSteps}'\} \qquad\qquad\qquad\qquad\qquad\qquad \to \\
\qquad (\mathit{pol} : \mathit{Pol}\ (\mathit{next}\ t)) \to \mathit{Opt}\ \mathit{pol} \qquad\qquad\qquad\quad \to \\
\qquad (\mathit{lp} : \mathit{LocalPol}\ t) \qquad\quad \to \mathit{OptExt}\ \mathit{lp}\ \mathit{pol} \to \\
\qquad \mathit{Opt}\ (\mathit{Cons}\ \mathit{lp}\ \mathit{pol})
\end{array}
$$

$$
\begin{array}{l}
\mathit{Bellman}\ \mathit{pol}\ \mathit{pol\_opt}\ \mathit{lp}\ \mathit{lp\_opt}\ (\mathit{Cons}\ \mathit{lp}'\ \mathit{pol}')\ s = \\
\quad \textbf{let}\ c' = \mathit{lp}'\ s \qquad \textbf{in} \\
\quad \textbf{let}\ s' = \mathit{step}\ s\ c'\ \textbf{in} \\
\qquad \mathit{lteTrans} \\
\qquad\quad (\mathit{plusMonR}\ (\mathit{pol\_opt}\ \mathit{pol}'\ s')) \\
\qquad\quad (\mathit{lp\_opt}\ \mathit{lp}'\ s)
\end{array}
$$

Dynamic programming, deterministic case ctd.

We reduce the problem of finding optimal policies to that of finding optimal extensions.

$$
\begin{array}{l}
\mathit{extend} \;:\; \mathit{Pol}\;(\mathit{next}\;t) \;\to\; \mathit{LocalPol}\;t \\
\mathit{extend}\;\mathit{pol}\;s = \mathit{max}\;\mathit{ctrlVal} \\
\quad \textbf{where} \\
\quad \mathit{ctrlVal} \;:\; \mathit{Ctrl}\;s \;\to\; \mathit{Float} \\
\quad \mathit{ctrlVal}\;c = \textbf{let}\;s' = \mathit{step}\;s\;c\;\textbf{in} \\
\qquad\qquad\qquad \mathit{reward}\;s\;c\;s' \oplus \mathit{Val}\;s'\;\mathit{pol}
\end{array}
$$

Dynamic programming, the essential kit

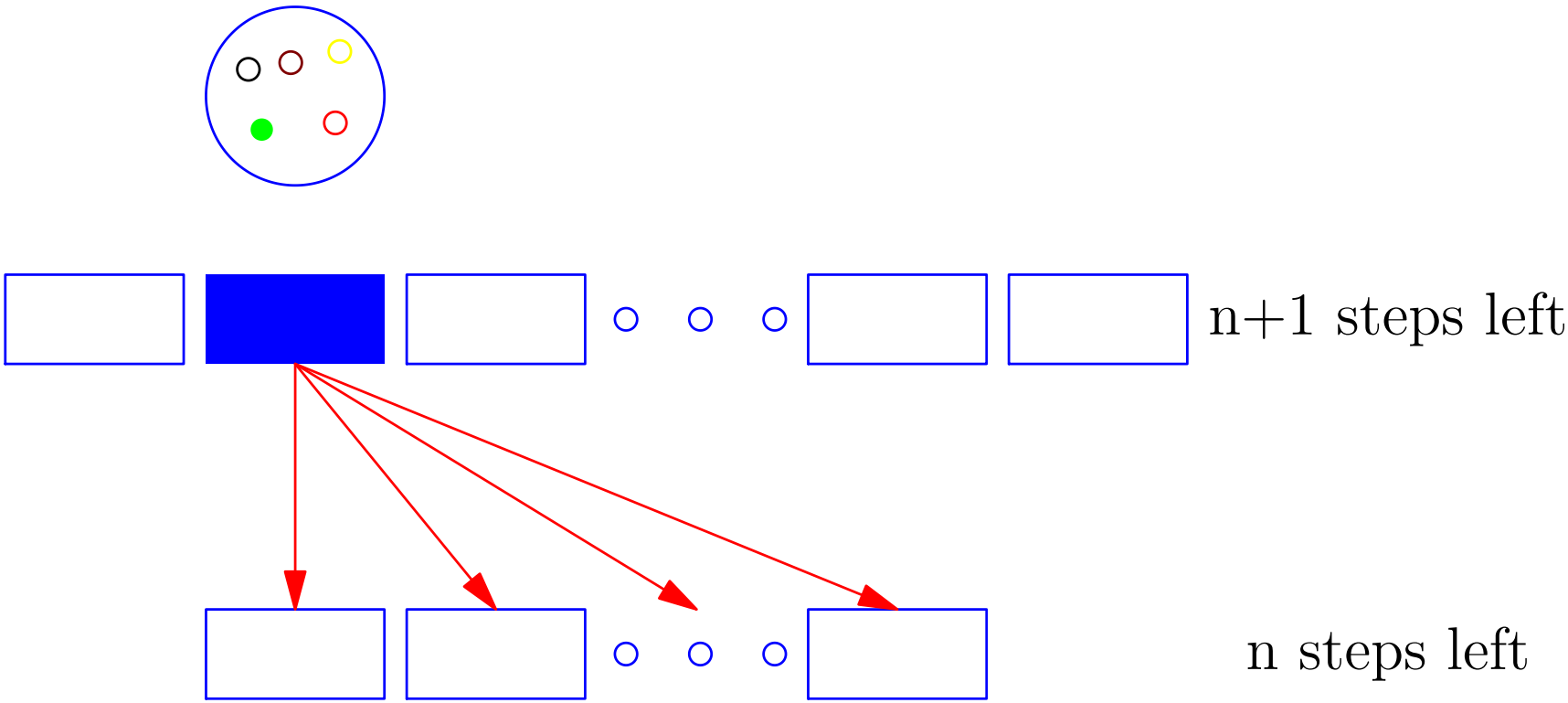
$$
\begin{aligned}
&\mathit{extend} \;:\; \mathit{Pol}\;(\mathit{next}\;t) \;\to\; \mathit{LocalPol}\;t\\
&\mathit{extend}\;\mathit{pol}\;s = \mathit{max}\;\mathit{ctrlVal}
\end{aligned}
$$

$$
\begin{aligned}
\mathit{MaxSpec} \;:\; &(u \;:\; X \;\to\; \mathit{Float}) \;\to\\
&(x \;:\; X) \qquad\qquad\quad \to\\
&u\;x \leqslant u\;(\mathit{max}\;\mathit{utility})
\end{aligned}
$$

$$
\begin{aligned}
\mathit{extend\_opt} \;:\; &(\mathit{pol} \;:\; \mathit{Pol}\;(\mathit{next}\;t)) \qquad\qquad\qquad\qquad \to\\
&(\mathit{lp}' \;:\; \mathit{LocalPol}\;t) \;\to\; (s \;:\; \mathit{State}\;(\mathit{fS}\;t)) \;\to\\
&\mathit{Val}\;s\;(\mathit{Cons}\;\mathit{lp}'\;\mathit{pol}) \leqslant \mathit{Val}\;s\;(\mathit{Cons}\;(\mathit{extend}\;\mathit{pol})\;\mathit{pol})
\end{aligned}
$$

$$
\mathit{extend\_opt}\;\mathit{pol}\;\mathit{lp}'\;s = \mathit{MaxSpec}\;\mathit{ctrlVal}\;(\mathit{lp}'\;s)
$$

The general case



The general case

What changes from the deterministic case?

$$NrSteps = S\ NrSteps'$$

$$State : Fin\ NrSteps \to Set$$

$$Ctrl : State\ (fS\ t) \to Set$$

$$step : (s : State\ (fS\ t)) \to (c : Ctrl\ s) \to State\ (next\ t)$$

$$reward : (s : State\ (fS\ t)) \to (c : Ctrl\ s) \to (s' : State\ (next\ t)) \to Float$$

The general case

What changes from the deterministic case?

$$NrSteps = S\ NrSteps'$$

$$State \quad : \quad Fin\ NrSteps \ \rightarrow \ Set$$

$$Ctrl \quad : \quad State\ (fS\ t) \ \rightarrow \ Set$$

$$step \quad : \quad (s \ : \ State\ (fS\ t)) \ \rightarrow \ (c \ : \ Ctrl\ s) \ \rightarrow \ M\ (State\ (next\ t))$$

$$reward \quad : \quad (s \ : \ State\ (fS\ t)) \ \rightarrow \ (c \ : \ Ctrl\ s) \ \rightarrow \ (s' \ : \ State\ (next\ t)) \ \rightarrow \ Float$$

The general case, ctd.

We consider M to be an endo-functor on Set.

$$M \quad : \; Set \;\rightarrow\; Set$$

$$Mmap \; : \; (A \;\rightarrow\; B) \;\rightarrow\; M\ A \;\rightarrow\; M\ B$$

$$step \; : \; (s \; : \; State\ (fS\ t)) \;\rightarrow\; (c \; : \; Ctrl\ s) \;\rightarrow\; M\ (State\ (next\ t))$$

The general case, ctd.

What changes from the deterministic case?

$$\begin{aligned} &\mathit{LocalPol} \quad : (t \ : \ \mathit{Fin}\ \mathit{NrSteps}') \ \to \ \mathit{Set} \\ &\mathit{LocalPol}\ t = (s \ : \ \mathit{State}\ (\mathit{fS}\ t)) \ \to \ \mathit{Ctrl}\ s \end{aligned}$$

$$\begin{aligned} &\textbf{data}\ \mathit{Pol} \ : (t \ : \ \mathit{Fin}\ \mathit{NrSteps}) \ \to \ \mathit{Set}\ \textbf{where} \\ &\quad \mathit{Nil} \quad : \mathit{Pol}\ \mathit{fO} \\ &\quad \mathit{Cons} \quad : \mathit{LocalPol}\ t \ \to \ \mathit{Pol}\ (\mathit{next}\ t) \ \to \ \mathit{Pol}\ (\mathit{fS}\ t) \end{aligned}$$

The general case, ctd.

No changes from the deterministic case.

$$\begin{aligned} &\mathit{LocalPol} \quad : (t \ : \ \mathit{Fin}\ \mathit{NrSteps}') \ \rightarrow \ \mathit{Set} \\ &\mathit{LocalPol}\ t = (s \ : \ \mathit{State}\ (\mathit{fS}\ t)) \ \rightarrow \ \mathit{Ctrl}\ s \end{aligned}$$

$$\begin{aligned} &\textbf{data}\ \mathit{Pol} \ : (t \ : \ \mathit{Fin}\ \mathit{NrSteps}) \ \rightarrow \ \mathit{Set}\ \textbf{where} \\ &\quad \mathit{Nil} \quad : \mathit{Pol}\ \mathit{fO} \\ &\quad \mathit{Cons} \quad : \mathit{LocalPol}\ t \ \rightarrow \ \mathit{Pol}\ (\mathit{next}\ t) \ \rightarrow \ \mathit{Pol}\ (\mathit{fS}\ t) \end{aligned}$$

The general case, ctd.

What changes from the deterministic case?

$$
\begin{aligned}
&\mathit{Val} \quad : (s : \mathit{State}\ t) \to \mathit{Pol}\ t \to \mathit{Float} \\
&\mathit{Val}\ \_\ \mathit{Nil} = 0 \\
&\mathit{Val}\ \{t = \mathit{fS}\ t'\}\ s\ (\mathit{Cons}\ \mathit{lp}\ \mathit{pols}) = \\
&\quad \mathit{reward}\ s\ c\ s' \oplus \mathit{Val}\ s'\ \mathit{pols} \\
&\quad \textbf{where} \\
&\quad c \ : \ \mathit{Ctrl}\ s \\
&\quad c \ = \ \mathit{lp}\ s \\
&\quad s' \ : \ \mathit{State}\ (\mathit{next}\ t') \\
&\quad s' = \ \mathit{step}\ s\ c
\end{aligned}
$$

The general case, ctd.

The return type of *step*...

$$
\begin{aligned}
&\mathit{Val} \quad : (s : \mathit{State}\ t) \to \mathit{Pol}\ t \to \mathit{Float} \\
&\mathit{Val}\ \_\ \mathit{Nil} = 0 \\
&\mathit{Val}\ \{t = \mathit{fS}\ t'\}\ s\ (\mathit{Cons}\ \mathit{lp}\ \mathit{pols}) = \\
&\quad \mathit{reward}\ s\ c\ s' \oplus \mathit{Val}\ s'\ \mathit{pols} \\
&\quad \textbf{where} \\
&\quad c \quad : \mathit{Ctrl}\ s \\
&\quad c \quad = \mathit{lp}\ s \\
&\quad \mathit{ms}' \ : \mathit{M}\ \mathit{State}\ (\mathit{next}\ t') \\
&\quad \mathit{ms}' = \mathit{step}\ s\ c
\end{aligned}
$$

The general case, ctd.

. . . requiring an *Mmap*. . .

$$
\begin{aligned}
&\mathit{Val} \quad : (s : \mathit{State}\ t) \to \mathit{Pol}\ t \to \mathit{Float}\\
&\mathit{Val}\ \_\ \mathit{Nil} = 0\\
&\mathit{Val}\ \{t = \mathit{fS}\ t'\}\ s\ (\mathit{Cons}\ \mathit{lp}\ \mathit{pols}) =\\
&\quad \mathit{Mmap}\ (\lambda s' \Rightarrow \mathit{reward}\ s\ c\ s' \oplus \mathit{Val}\ s'\ \mathit{pols})\ \mathit{ms}'\\
&\quad \textbf{where}\\
&\quad c \quad : \mathit{Ctrl}\ s\\
&\quad c \quad = \mathit{lp}\ s\\
&\quad \mathit{ms}' \ : \mathit{M}\ \mathit{State}\ (\mathit{next}\ t')\\
&\quad \mathit{ms}' = \mathit{step}\ s\ c
\end{aligned}
$$

The general case, ctd.

. . . requiring a $meas : M\ Float \to Float$.

$$
\begin{aligned}
&Val : (s : State\ t) \to Pol\ t \to Float \\
&Val\ \_\ Nil = 0 \\
&Val\ \{t = fS\ t'\}\ s\ (Cons\ lp\ pols) = \\
&\quad meas\ (Mmap\ (\lambda s' \Rightarrow reward\ s\ c\ s' \oplus Val\ s'\ pols)\ ms') \\
&\quad \textbf{where} \\
&\quad c : Ctrl\ s \\
&\quad c = lp\ s \\
&\quad ms' : M\ State\ (next\ t') \\
&\quad ms' = step\ s\ c
\end{aligned}
$$

The general case, ctd.

What changes from the deterministic case?

$$
\begin{aligned}
&\mathit{Opt} && : \mathit{Pol}\ t \to \mathit{Set}\\
&\mathit{Opt}\ \{t\}\ \mathit{pol} && = (\mathit{pol}' : \mathit{Pol}\ t) \to (s : \mathit{State}\ t) \to \\
& && \quad \mathit{Val}\ s\ \mathit{pol}' \leqslant \mathit{Val}\ s\ \mathit{pol}\\
&\mathit{OptExt} && : \{t : \mathit{Fin}\ \mathit{NrSteps}'\} \to \\
& && \quad (\mathit{lp} : \mathit{LocalPol}\ t) \to \\
& && \quad (\mathit{pol} : \mathit{Pol}\ (\mathit{next}\ t)) \to \\
& && \quad \mathit{Set}\\
&\mathit{OptExt}\ \{t\}\ \mathit{lp}\ \mathit{pol} && = (\mathit{lp}' : \mathit{LocalPol}\ t) \to \\
& && \quad (s : \mathit{State}\ (\mathit{fS}\ t)) \to \\
& && \quad \mathit{Val}\ s\ (\mathit{Cons}\ \mathit{lp}'\ \mathit{pol}) \leqslant \mathit{Val}\ s\ (\mathit{Cons}\ \mathit{lp}\ \mathit{pol})
\end{aligned}
$$

The general case, ctd.

No changes from the deterministic case.

$$
\begin{aligned}
&\mathit{Opt} && : \mathit{Pol}\ t \to \mathit{Set}\\
&\mathit{Opt}\ \{t\}\ \mathit{pol} && = (\mathit{pol}' : \mathit{Pol}\ t) \to (s : \mathit{State}\ t) \to \\
& && \quad \mathit{Val}\ s\ \mathit{pol}' \leqslant \mathit{Val}\ s\ \mathit{pol}\\
&\mathit{OptExt} && : \{t : \mathit{Fin}\ \mathit{NrSteps}'\} \to \\
& && \quad (\mathit{lp} : \mathit{LocalPol}\ t) \to \\
& && \quad (\mathit{pol} : \mathit{Pol}\ (\mathit{next}\ t)) \to \\
& && \quad \mathit{Set}\\
&\mathit{OptExt}\ \{t\}\ \mathit{lp}\ \mathit{pol} && = (\mathit{lp}' : \mathit{LocalPol}\ t) \to \\
& && \quad (s : \mathit{State}\ (\mathit{fS}\ t)) \to \\
& && \quad \mathit{Val}\ s\ (\mathit{Cons}\ \mathit{lp}'\ \mathit{pol}) \leqslant \mathit{Val}\ s\ (\mathit{Cons}\ \mathit{lp}\ \mathit{pol})
\end{aligned}
$$

Dynamic programming, the general case

What changes from the deterministic case?
Bellman: an optimal extension of an optimal policy is optimal.

$$\begin{array}{lll} \mathit{Bellman} : \{t : \mathit{Fin}\ \mathit{NrSteps}'\} & & \to \\ (\mathit{pol} : \mathit{Pol}\ (\mathit{next}\ t)) & \to\ \mathit{Opt}\ \mathit{pol} & \to \\ (\mathit{lp} : \mathit{LocalPol}\ t) & \to\ \mathit{OptExt}\ \mathit{lp}\ \mathit{pol} & \to \\ \mathit{Opt}\ (\mathit{Cons}\ \mathit{lp}\ \mathit{pol}) & & \end{array}$$

For any $\mathit{lp}' : \mathit{LocalPol}\ t$ and $\mathit{pol}' : \mathit{Pol}\ (\mathit{next}\ t)$, we have for any $s : \mathit{State}\ (\mathit{fS}\ t)$

$$\mathit{Val}\ s\ (\mathit{Cons}\ \mathit{lp}'\ \mathit{pols}') \leqslant \mathit{Val}\ s\ (\mathit{Cons}\ \mathit{lp}\ \mathit{pols})$$

Dynamic programming, the general case

No changes from the deterministic case.
Bellman: an optimal extension of an optimal policy is optimal.

$$\begin{array}{lll} \mathit{Bellman} : \{t : \mathit{Fin}\ \mathit{NrSteps}'\} & & \to \\ (\mathit{pol} : \mathit{Pol}\ (\mathit{next}\ t)) & \to\ \mathit{Opt}\ \mathit{pol} & \to \\ (\mathit{lp} : \mathit{LocalPol}\ t) & \to\ \mathit{OptExt}\ \mathit{lp}\ \mathit{pol} & \to \\ \mathit{Opt}\ (\mathit{Cons}\ \mathit{lp}\ \mathit{pol}) & & \end{array}$$

For any $\mathit{lp}' : \mathit{LocalPol}\ t$ and $\mathit{pol}' : \mathit{Pol}\ (\mathit{next}\ t)$, we have for any $s : \mathit{State}\ (\mathit{fS}\ t)$

$$\mathit{Val}\ s\ (\mathit{Cons}\ \mathit{lp}'\ \mathit{pols}') \leqslant \mathit{Val}\ s\ (\mathit{Cons}\ \mathit{lp}\ \mathit{pols})$$

Dynamic programming, the general case ctd.

What chages from the deterministic case?
Let $c' = lp'\ s$, $s' = step\ s\ c'$. We have

$$
\begin{aligned}
& Val\ s\ (Cons\ lp'\ pols') \\
= \quad & \{\ \text{by definition}\ \} \\
& reward\ s\ c'\ s' \oplus Val\ s'\ pols' \\
\leqslant \quad & \{\ \text{monotonicity of } \oplus, \textit{pol} \text{ optimal}\ \} \\
& reward\ s\ c'\ s' \oplus Val\ s'\ pols \\
= \quad & \{\ \text{by definition}\ \} \\
& Val\ s\ (Cons\ lp'\ pols) \\
\leqslant \quad & \{\ \text{lp optimal extension}\ \} \\
& Val\ s\ (Cons\ lp\ pols)
\end{aligned}
$$

Dynamic programming, the general case ctd.

What chages from the deterministic case?
Let $c' = lp'\ s$, $ms' = step\ s\ c'$. We have

$$
\begin{array}{ll}
 & Val\ s\ (Cons\ lp'\ pols') \\
= & \quad \{\ \text{by definition}\ \} \\
 & meas\ (Mmap\ (\lambda s' \Rightarrow reward\ s\ c'\ s' \oplus Val\ s'\ pols')\ ms') \\
\leqslant & \quad \{\ ???\ \} \\
 & meas\ (Mmap\ (\lambda s' \Rightarrow reward\ s\ c'\ s' \oplus Val\ s'\ pols)\ ms') \\
= & \quad \{\ \text{by definition}\ \} \\
 & Val\ s\ (Cons\ lp'\ pols) \\
\leqslant & \quad \{\ \text{lp optimal extension}\ \} \\
 & Val\ s\ (Cons\ lp\ pols)
\end{array}
$$

Monotonicity

A sufficient monotonicity requirement for *meas*:

$$\begin{aligned} \mathit{measMon} : {} & (f : X \to \mathit{Float}) \to (g : X \to \mathit{Float}) \to \\ & ((x : X) \to f\ x \leqslant g\ x) \to \\ & (mx : M\ X) \to \\ & \mathit{meas}\ (\mathit{Mmap}\ f\ mx) \leqslant \mathit{meas}\ (\mathit{Mmap}\ g\ mx) \end{aligned}$$

Dynamic programming, the general case ctd.

Let $c' = lp'\ s$, $ms' = step\ s\ c'$. We have

$$
\begin{array}{ll}
 & Val\ s\ (Cons\ lp'\ pols') \\
= & \quad \{\ \text{by definition}\ \} \\
 & meas\ (Mmap\ (\lambda s' \Rightarrow reward\ s\ c'\ s' \oplus Val\ s'\ pols')\ ms') \\
\leqslant & \quad \{\ \text{measMon, monotonicity of } \oplus, \textit{pol} \text{ optimal}\ \} \\
 & meas\ (Mmap\ (\lambda s' \Rightarrow reward\ s\ c'\ s' \oplus Val\ s'\ pols)\ ms') \\
= & \quad \{\ \text{by definition}\ \} \\
 & Val\ s\ (Cons\ lp'\ pols) \\
\leqslant & \quad \{\ \text{lp optimal extension}\ \} \\
 & Val\ s\ (Cons\ lp\ pols)
\end{array}
$$

Dynamic programming, general case ctd.

Idris implementation:

$$
\begin{array}{l}
\mathit{Bellman}\ \{\mathit{t}\}\ \mathit{pol}\ \mathit{pol\_opt}\ \mathit{lp}\ \mathit{lp\_opt}\ (\mathit{Cons}\ \mathit{lp}'\ \mathit{pol}')\ \mathit{s} = \\
\quad \mathit{lteTrans}\ \mathit{step1}\ \mathit{step2} \\
\quad \textbf{where} \\
\quad\quad \mathit{c}' \quad\quad = \mathit{lp}'\ \mathit{s} \\
\quad\quad \mathit{ms}' \quad\quad = \mathit{step}\ \mathit{s}\ \mathit{c} \\
\quad\quad \mathit{f} \quad\quad = \lambda \mathit{s}' \Rightarrow \mathit{reward}\ \mathit{s}\ \mathit{c}\ \mathit{s}' \oplus \mathit{Val}\ \mathit{s}'\ \mathit{pol}' \\
\quad\quad \mathit{g} \quad\quad = \lambda \mathit{s}' \Rightarrow \mathit{reward}\ \mathit{s}\ \mathit{c}\ \mathit{s}' \oplus \mathit{Val}\ \mathit{s}'\ \mathit{pol} \\
\\
\quad\quad \mathit{lemma1} \quad : (\mathit{s}' : \mathit{State}\ (\mathit{next}\ \mathit{t})) \rightarrow \mathit{f}\ \mathit{s}' \leqslant \mathit{g}\ \mathit{s}' \\
\quad\quad \mathit{lemma1}\ \mathit{s}' = \mathit{plusMonR}\ (\mathit{pol\_opt}\ \mathit{pol}'\ \mathit{s}') \\
\quad\quad \mathit{step1} \quad : \mathit{meas}\ (\mathit{Mmap}\ \mathit{f}\ \mathit{ms}') \leqslant \mathit{meas}\ (\mathit{Mmap}\ \mathit{g}\ \mathit{ms}') \\
\quad\quad \mathit{step1} \quad = \mathit{measMon}\ \mathit{f}\ \mathit{g}\ \mathit{lemma1}\ \mathit{ms}' \\
\quad\quad \mathit{step2} \quad : \mathit{meas}\ (\mathit{Mmap}\ \mathit{g}\ \mathit{ms}') \leqslant \mathit{Val}\ \mathit{s}\ (\mathit{Cons}\ \mathit{lp}\ \mathit{pol}) \\
\quad\quad \mathit{step2} \quad = \mathit{lp\_opt}\ \mathit{lp}'\ \mathit{s}
\end{array}
$$

Dynamic programming, the general case ctd.

What changes from the deterministic case?

$$
\begin{aligned}
&\mathit{extend} \;:\; \mathit{Pol}\;(\mathit{next}\;t) \;\rightarrow\; \mathit{LocalPol}\;t \\
&\mathit{extend}\;\mathit{pol}\;s = \mathit{max}\;\mathit{ctrlVal} \\
&\quad \textbf{where} \\
&\quad \mathit{ctrlVal} \quad : \mathit{Ctrl}\;s \;\rightarrow\; \mathit{Float} \\
&\quad \mathit{ctrlVal}\;c = \textbf{let}\;s' = \mathit{step}\;s\;c\;\textbf{in} \\
&\qquad\qquad\qquad \mathit{reward}\;s\;c\;s' \oplus \mathit{Val}\;s'\;\mathit{pol}
\end{aligned}
$$

Dynamic programming, the general case ctd.

$$
\begin{aligned}
&\mathit{extend} \;:\; \mathit{Pol}\;(\mathit{next}\;t) \;\rightarrow\; \mathit{LocalPol}\;t \\
&\mathit{extend}\;\mathit{pol}\;s = \mathit{max}\;\mathit{ctrlVal} \\
&\quad \textbf{where} \\
&\quad \mathit{ctrlVal} \quad : \; \mathit{Ctrl}\;s \;\rightarrow\; \mathit{Float} \\
&\quad \mathit{ctrlVal}\;c = \textbf{let}\;\mathit{ms}' = \mathit{step}\;s\;c\;\textbf{in} \\
&\qquad\qquad\qquad \mathit{meas}\;(\mathit{Mmap}\;(\lambda s' \Rightarrow \mathit{reward}\;s\;c\;s' \oplus \mathit{Val}\;s'\;\mathit{pol})\;\mathit{ms}')
\end{aligned}
$$

Dynamic programming, the essential kit

What chages from the deterministic case?

$$
\begin{aligned}
&\mathit{extend} \;:\; \mathit{Pol}\;(\mathit{next}\;t) \;\to\; \mathit{LocalPol}\;t\\
&\mathit{extend}\;\mathit{pol}\;s = \mathit{max}\;\mathit{ctrlVal}
\end{aligned}
$$

$$
\begin{aligned}
\mathit{MaxSpec} \;:\; &(u \;:\; X \;\to\; \mathit{Float}) \;\to\\
&(x \;:\; X) \qquad\qquad\quad \to\\
&u\;x \leqslant u\;(\mathit{max}\;\mathit{utility})
\end{aligned}
$$

$$
\begin{aligned}
\mathit{extend\_opt} \;:\; &(\mathit{pol} \;:\; \mathit{Pol}\;(\mathit{next}\;t)) \qquad\qquad\qquad\qquad \to\\
&(\mathit{lp}' \;:\; \mathit{LocalPol}\;t) \;\to\; (s \;:\; \mathit{State}\;(\mathit{fS}\;t)) \;\to\\
&\mathit{Val}\;s\;(\mathit{Cons}\;\mathit{lp}'\;\mathit{pol}) \leqslant \mathit{Val}\;s\;(\mathit{Cons}\;(\mathit{extend}\;\mathit{pol})\;\mathit{pol})
\end{aligned}
$$

$$
\mathit{extend\_opt}\;\mathit{pol}\;\mathit{lp}'\;s = \mathit{MaxSpec}\;\mathit{ctrlVal}\;(\mathit{lp}'\;s)
$$

Dynamic programming, the essential kit

No chages from the deterministic case.

$$
\begin{aligned}
&\mathit{extend} \;:\; \mathit{Pol}\;(\mathit{next}\;t) \;\to\; \mathit{LocalPol}\;t\\
&\mathit{extend}\;\mathit{pol}\;s = \mathit{max}\;\mathit{ctrlVal}
\end{aligned}
$$

$$
\begin{aligned}
\mathit{MaxSpec} \;:\; &(u \;:\; X \;\to\; \mathit{Float}) \;\to\\
&(x \;:\; X) \qquad\qquad\quad \to\\
&u\;x \leqslant u\;(\mathit{max}\;\mathit{utility})
\end{aligned}
$$

$$
\begin{aligned}
\mathit{extend\_opt} \;:\; &(\mathit{pol} \;:\; \mathit{Pol}\;(\mathit{next}\;t)) \qquad\qquad\qquad\qquad \to\\
&(\mathit{lp}' \;:\; \mathit{LocalPol}\;t) \;\to\; (s \;:\; \mathit{State}\;(\mathit{fS}\;t)) \;\to\\
&\mathit{Val}\;s\;(\mathit{Cons}\;\mathit{lp}'\;\mathit{pol}) \leqslant \mathit{Val}\;s\;(\mathit{Cons}\;(\mathit{extend}\;\mathit{pol})\;\mathit{pol})
\end{aligned}
$$

$$
\mathit{extend\_opt}\;\mathit{pol}\;\mathit{lp}'\;s = \mathit{MaxSpec}\;\mathit{ctrlVal}\;(\mathit{lp}'\;s)
$$

Optimization problems

We have used

$$\begin{aligned} \mathit{max} &: \{s : \mathit{State}\ (\mathit{fS}\ t)\} \to (\mathit{utility} : \mathit{Ctrl}\ s \to \mathit{Float}) \to \\ &\quad \mathit{Ctrl}\ s \end{aligned}$$

$$\begin{aligned} \mathit{MaxSpec} &: \{s : \mathit{State}\ (\mathit{fS}\ t)\} \to (\mathit{utility} : \mathit{Ctrl}\ s \to \mathit{Float}) \to \\ &\quad (c : \mathit{Ctrl}\ s) \to \\ &\quad \mathit{utility}\ c \leqslant \mathit{utility}\ (\mathit{max}\ \mathit{utility}) \end{aligned}$$

What if *Ctrl s* is infinite, e.g. an interval?

Optimization problems, ctd.

Current practice: use an external optimizer and assume it works.

Optimization problems, ctd.

Current practice: use an external optimizer and assume it works.

MaxSpec serves as a documentation of this assumption.

Optimization problems, ctd.

Current practice: use an external optimizer and assume it works.

MaxSpec serves as a documentation of this assumption.

Often, the type of *utility* is constrained to functions for which *MaxSpec* is less of a lie.

Optimization problems, ctd.

E.g.: for elementary functions defined on "convenient" intervals one can show that Newton-based methods converge. The result is an interval guaranteed to contain the solution.

Optimization problems, ctd.

E.g.: for elementary functions defined on "convenient" intervals one can show that Newton-based methods converge. The result is an interval guaranteed to contain the solution.

Even then, formalizing the proof in Idris is not trivial: standard proofs are classical. Thus all we can usually show is that the resulting interval *cannot fail to contain the solution*.

Optimization problems, ctd.

E.g.: for elementary functions defined on "convenient" intervals one can show that Newton-based methods converge. The result is an interval guaranteed to contain the solution.

Even then, formalizing the proof in Idris is not trivial: standard proofs are classical. Thus all we can usually show is that the resulting interval *cannot fail to contain the solution*.

At the moment, we use external libraries for interval analysis anyway...

Conclusions

We have our work cut out for us:

- Specify more commonly used external routines, e.g. for interpolation.

Conclusions

We have our work cut out for us:

- Specify more commonly used external routines, e.g. for interpolation.
- Extend the dynamic programming example to a full model such as Remind.

Conclusions

We have our work cut out for us:

- Specify more commonly used external routines, e.g. for interpolation.
- Extend the dynamic programming example to a full model such as Remind.
- Improve notation for dependent-types, e.g. **where**-clauses for type declarations.

Conclusions

We have our work cut out for us:

- Specify more commonly used external routines, e.g. for interpolation.
- Extend the dynamic programming example to a full model such as Remind.
- Improve notation for dependent-types, e.g. **where**-clauses for type declarations.
- Develop DSLs for specifications of economic, climate, etc. models.

Conclusions

We have our work cut out for us:

- Specify more commonly used external routines, e.g. for interpolation.
- Extend the dynamic programming example to a full model such as Remind.
- Improve notation for dependent-types, e.g. **where**-clauses for type declarations.
- Develop DSLs for specifications of economic, climate, etc. models.
- Implement interval analysis methods for validated numerics.

Conclusions

We have our work cut out for us:

- Specify more commonly used external routines, e.g. for interpolation.
- Extend the dynamic programming example to a full model such as Remind.
- Improve notation for dependent-types, e.g. **where**-clauses for type declarations.
- Develop DSLs for specifications of economic, climate, etc. models.
- Implement interval analysis methods for validated numerics.
- Prepare for the constructive mathematics revolution, e.g. results from projects such as ForMath.

Acknowledgments

Contributors:

Nicola Botta (PIK), Edwin Brady (University of St. Andrews), Patrik Jansson (Chalmers Univ. of Technology), Paul Flondor (Polytechnical Univ. of Bucharest), Carlo Jaeger (GCF), Johan Jeuring (Utrecht University), Leonidas Paroussos (Nat. Tech. Univ. Athens)

Members of the Lagom project (PIK)

Members of the Cartesian Seminar at the Univ. of Potsdam

A final word.

"The road to wisdom? Well, it's plain
and simple to express:
Err
and err
and err again,
but less
and less
and less."

Piet Hein (1905–1996), *The Road to Wisdom*, in Grooks (1966).