Formal Methods for Probabilistic Systems
Annabelle McIver
Carroll Morgan

- Probabilistic temporal logic: qTL
- Probabilistic sequential-programming logic: pGCL
- Probabilistic modal mu-calculus: qMμ

- Motivating example (the stock market)
- Calculational approaches (we hope)
- Logical vs operational approaches
- Games, strategies and values
- The Game interpretation
- The Logical interpretation
- The congruence theorem (hard)
- Why aren’t we rich, then?

Investing in the futures market

1. The market value $v$ of the shares is a whole number of dollars between $0$ and $10$ inclusive; it has a probability $p$ of going up by $1$ in any month, and $1-p$ of going down by $1$ — but it remains within those bounds. The probability $p$ represents short-term market uncertainty.

2. Probability $p$ itself varies month-by-month in steps of $0.1$ between zero and one: when $v$ is less than $5$ the probability that $p$ will rise is $2/3$; when $v$ is more than $5$ the probability of $p$’s falling is $2/3$; and when $v$ is $5$ exactly the probability is $1/2$ of going either way. The movement of $p$ represents investors’ knowledge of long-term “cyclic second-order” trends.

3. There is a cap $c$ on the value of $v$, initially $10$, which has probability $1/2$ of falling by $1$ in any month; otherwise it remains where it is. (This modifies Item 1 above.) The “falling cap” models the fact that the company is in a slow decline.

4. If in a given month the investor does not reserve, then at the very next month he might find he is temporarily barred from doing so. But he cannot be barred two months consecutively.

5. If he never reserves, then he never sells and his return is thus zero.
Investing in the futures market

One month’s activity is described by this program fragment in pGCL, our small extension of Dijkstra’s guarded-command language GCL.

\[
\text{month} \triangleq (v + 1) \cap c \quad \nu \triangleq (v - 1) \cup 0;
\]

\[
\begin{align*}
\text{if} & \quad v < 5 \quad \text{then} \quad p := (p + 0.1) \cap 1 \quad 2/3 
\oplus (p - 0.1) \cup 0 \\
\text{elsif} & \quad v > 5 \quad \text{then} \quad p := (p - 0.1) \cup 0 \quad 2/3 
\oplus (p + 0.1) \cap 1 \\
\text{else} & \quad p := (p - 0.1) \cup 0 \quad 1/2 
\oplus (p + 0.1) \cap 1 \\
\fi;
\]

\[
c := (c - 1) \cup 0 \quad 1/2 \oplus c
\]

D. Kozen. Results on the propositional $\mu$-calculus. TCS 27, 333-54, 1983.

---

Investing in the futures market

The overall game is described by this “pseudo-code” loop, with a results construct that gives the final result.

If the loop is unending, the result is taken to be zero (as for a least fixed-point.)

\[
\text{Game} \triangleq (\mu X \cdot \{\text{month}\}(v) \sqcup \{\text{month}\}(X \sqcap \{\text{month}\}X))
\]

Calculating the expected return

In Mathematica

Futures game.nb

D. Kozen. Results on the propositional $\mu$-calculus. TCS 27, 333-54, 1983.
Calculating the expected return

In **Mathematica**

Constant expectations, and the example iterator functions `ROJ` etc:

- function `ROJ` applies the demonic choice late;
- function `f2` applies the demonic choice early;
- function `f3` applies a fixed investor strategy;
- function `f4` calculates probability; function `f5` removes the barring feature; function `f6` calculates probability using a strategy; function `f7` makes barring probability 0.5.

```
ROJ = Table[getV[s], {s, 1, numStates}]
Function[1] = Table[x, {x, 1, numStates}]
TBar = Random[Real, {0, 1}] 
condExp = condExp[vExp]
monthExp = monthExp[vExp]
initialIncreaseProb = initialIncreaseProb[vExp]
vestExp = vestExp[vExp]
initialShare = initialShare[vExp]
```

Calculating the expected return

In **Mathematica**

Fixed-point applied to all initial states `0 < v < maxV`, with c at the maximum and initial share increase probability shown.

```
initialShareProb = 0.5
```

Calculating the expected return

In **PRISM**

Using the game interpretation, we can generate a pr simulation by replicating states with multiple branching a minimising or maximizing choice as they are enclosed the maximum, whether to take whether or be other for each respective strategy.

For example, the investor's strategy (maximising) might be add until the share value `v` (rising) meets or exceeds the peaks. After a fixed value of the game's parameter `c`, and the further the number of states. We call this strategy's "least-of-the-paths" strategy.

The other hand, `W` strategy (minimising, the least possible, whenever the probability `p` of rising. As is usual in game theory, the actual strategy is that of the game's value, which is the only valid, and is the relative game's value is indeed well-defined.

We have used both Mathematica and PRISM's code scripts are available online from here and further detailed for the "best-of-the-paths" strategy given above.
Calculating the expected return

Futures Market Investor

The model follows a three-month strategy: the share value rises if the investor makes a maximising choice of whether to bar the investor or not. At the beginning of each month, the investor makes a maximising choice of whether to bar the investor. The investor then tries to maximize his or her return against the other player (the futures market) which attempts to minimize this return. The investor (I) does not know whether an investment is ‘futurist’; it is a specific number of shares in a specific company that he can receive at the first day of any month he chooses. Ideally, the investor will receive shares which increase in value — he can then sell the shares. The investor problem is to decide when to make the maximization so that the subsequent investor has maximized value. If the investor does not return, the investor’s return is zero.

www.cs.bham.ac.uk/~xjp/prism/casestudies/investor.html

Calculating the expected return

Modelling the system in PRISM

Below we give the PRISM code for this model. The model has 6,688 states and in PRISM it takes 0.169 seconds to construct this model.

```
// EXAMPLE: INVESTMENT IN THE FUTURES MARKET
// (Moller and Morgan 80)
nomdeterministic;
// mobile use to synchronize transitions
module parts

h t {1:1} // input

{[2008]} [0] => {0} // two transitions made at the start of the month synchronize on 'invest'
{[1998]} [0] => {0} // two transitions made during the month synchronize on 'invest'
{[0]} [1] => {0} // new investor has ranked in Share not making changes

endmodule

// the investor
module investor

i t {1:1} // i = no reservation and i = 1 made reservation

{[2008]} [0] => {1} // do nothing
{[1998]} [0] => {1} // make reservation
{[2008]} [1] => {0} // made reservation
{[1998]} [1] => {0} // made reservation
{[0]} [1] & {0} => {1} // paid in shares (not invested)

endmodule

Calculating the expected return

Value of the Game

Using the game representation, we can generate a probabilistic tree from the transition system above by performing moves with-matches moving and marking each node, and making appropriate assignments of values. The following table presents the results of calculating the expected return on the system for the above strategy. The expected return is calculated by adding the expected value of the states and multiplying by the probability of reaching the respective state and adding all these together. The expected return for the investor can be read off from the table below.

<table>
<thead>
<tr>
<th>Initial share value</th>
<th>Expected return</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1.03</td>
</tr>
<tr>
<td>2</td>
<td>2.06</td>
</tr>
<tr>
<td>3</td>
<td>3.09</td>
</tr>
<tr>
<td>4</td>
<td>4.12</td>
</tr>
<tr>
<td>5</td>
<td>5.15</td>
</tr>
<tr>
<td>6</td>
<td>6.18</td>
</tr>
<tr>
<td>7</td>
<td>7.21</td>
</tr>
<tr>
<td>8</td>
<td>8.24</td>
</tr>
<tr>
<td>9</td>
<td>9.27</td>
</tr>
<tr>
<td>10</td>
<td>10.30</td>
</tr>
</tbody>
</table>

We have used the PRISM and MM97 computer programs with Matlab for this calculation. The results are obtained from some code and further details are available in [5097]. For example, if in initially 0, the code in [5097]. Then the expected return and initial values for the investor is given in the table below, and at the end of each month the investor will have the same expected return as given above.
Calculating the expected return

For example, the investor’s strategy (maximising, he or she hopes) for dealing with the falling cap might be: wait until the share value $v$ (rising) meets the cap $c$ (falling), and then reserve. Waiting for $v$ to rise is a good idea, but when it has met the cap $c$ there is clearly no point in waiting further. We call this strategy the ‘seat-of-the-pants’ strategy.

For example, if $p$ is initially $0.5$ and the cap $c$ is 10, then the optimal expected sale-value for the investor is given in the table below. In the table we also include the results when the investor follows the ‘seat-of-the-pants’ strategy given above.

<table>
<thead>
<tr>
<th>Initial share value</th>
<th>Expected sale price:</th>
<th>Optimal strategy ‘seat-of-the-pants’ strategy</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4.20</td>
<td>3.68</td>
</tr>
<tr>
<td>1</td>
<td>4.30</td>
<td>3.79</td>
</tr>
<tr>
<td>2</td>
<td>4.55</td>
<td>3.97</td>
</tr>
<tr>
<td>3</td>
<td>4.88</td>
<td>4.17</td>
</tr>
<tr>
<td>4</td>
<td>5.24</td>
<td>4.29</td>
</tr>
<tr>
<td>5</td>
<td>5.52</td>
<td>4.17</td>
</tr>
<tr>
<td>6</td>
<td>5.00</td>
<td>4.16</td>
</tr>
<tr>
<td>7</td>
<td>7.00</td>
<td>4.65</td>
</tr>
<tr>
<td>8</td>
<td>8.00</td>
<td>5.61</td>
</tr>
<tr>
<td>9</td>
<td>9.00</td>
<td>6.78</td>
</tr>
<tr>
<td>10</td>
<td>9.50</td>
<td>9.50</td>
</tr>
</tbody>
</table>

But what’s the theory of all this !??

How do we know that these three things are the same?

$\textbf{Game} \equiv (\mu X \cdot \{\text{month}\}(v) \sqcup \{\text{month}\}(X \cap \{\text{month}\}X))$

But what’s the theory of all this !??

How do we know that these three things are the same?

This is a formula, with a fixed-point interpretation.

$\textbf{Game} \equiv (\mu X \cdot \{\text{month}\}(v) \sqcup \{\text{month}\}(X \cap \{\text{month}\}X))$

Stochastic games, strategies and values

There are two players, a maximising player and a minimising player. A turn in the game is one of the following:

- An immediate payoff (between 0 and 1), ending the game;
- A maximising turn;
- A minimising turn; or
- A probabilistic choice.

The maximising player strives to make the (expected) payoff as high as possible; the minimising player tries to make it as low as possible. Neither player has any control over probabilistic outcomes.
The “value” of a game — examples

This game has value .30

This game has value .45

This game has value .50

The “value” of a game

Max uses the strategy “go right”; Min uses the strategy “go left”.

A logic for two-player probabilistic games

The two-player games are formalised by a quantitative modal-mu calculus logic (extending Kozen), which is a superset of the qTL we met earlier. The principal theorem we prove is that the value of a formula can be determined in either of two equivalent ways:

- Use the formula, à la Stirling (but extended by us), to play a probabilistic minimax-over-strategies game as above. Operational reasoning is used.

- Interpret the formula, à la Kozen (but extended by us), denotationally in a lattice of real-valued functions. Least- and greatest fixed-points are used.

The equivalence means that we can reason operationally about whether a formula is appropriate for our application (Stirling), and then use a form of mathematical logic to manipulate it (Kozen).

Building a game from a formula in qM\(\mu\)

We operate over a state space \(S\) (usually countable, often finite), and a derived space \(\mathcal{R}S\) of probabilistic/demonic transitions over \(S\) in which we can express the tree-building nodes we saw earlier.

\[
\phi \equiv X \mid A \mid \{k\} \phi \\
| \phi_1 \cap \phi_2 \| \phi_1 \cup \phi_2 \| \phi_1 < G \triangleright \phi_2 \\
| (\mu X \cdot \phi) \| (\nu X \cdot \phi)
\]

- Variables \(X\) are of type \(S \rightarrow [0, 1]\), and are used for binding fixed points.
- Terms \(A\) stand for fixed functions in \(S \rightarrow [0, 1]\).
- Terms \(k\) represent probabilistic state-to-state transitions in \(\mathcal{R}S\).
- Terms \(G\) describe Boolean functions of \(S\), used in \(<\) (“if”) \(G \triangleright\) (“else”) style.

Building a game from a formula in qM\(\mu\)

We shall assume generally that \(S\) is a countable state space (though for the principal result we restrict to finiteness). If \(f\) is a function with domain \(X\) then by \(f.x\) we mean \(f\) applied to \(x\), and \(f.x.y\) is \((f.x).y\) where appropriate; functional composition is written with \(\circ\), so that \((f \circ g).x = f.(g.x)\).

We denote the set of discrete probability sub-distributions over a set \(X\) by \(X\): it is the set of functions from \(X\) into the real interval \([0,1]\) that sum to no more than one.

If \(A\) is a random variable with respect to some probability space, and \(\delta\) is some probability sub-distribution, we write \(\int_A \delta\) for the expected value of \(A\) with respect to \(\delta\).

The space of generalised probabilistic transitions \(\mathcal{R}S\) comprises the functions \(t\) in \(S \rightarrow S_S\) where \(S_S\) is just the state space \(S\) with a special “payoff” state \(S\) adjoining.

Thus \(S_S\) is the set of sub-distributions over \(\mathcal{R}S\) such that \(t\) of \(\mathcal{R}S\) give the probability of passage from initial \(s\) to final (proper) \(s'\) as \(t.s.s'\); any deficit \(1 - \sum_{s'} t.s.s'\) is interpreted as the probability of an immediate halt with payoff

\[
t.s.\delta/(1 - \sum_{s'} t.s.s').
\]
If the current game position is \((\phi_i, s_i)\), then play proceeds as follows:

1. Free variables \(X\) do not occur in the game — their role is taken over by “colours”.

A sequence of game positions is called a game path and is of the form \((\phi_0, s_0), (\phi_1, s_1), \ldots\) with (if finite) a payoff position \((y)\) at the end. The initial formula \(\phi_0\) is the given \(\phi\), and \(s_0\) is an initial state in \(S\). A move from position \((\phi_i, s_i)\) to \((\phi_{i+1}, s_{i+1})\) or to \((y)\) is specified by the following rules.

The game is between two players Max and Min. Play progresses through a sequence of game positions, each of which is either a pair \((\phi, s)\) where \(\phi\) is a formula and \(s\) is a state in \(S\), or a single \((y)\) for some real-valued payoff \(y\) in \([0, 1]\). We use “colours” to handle repeated returns to a fixed point.

Building a game from a formula in \(qM\)

If the current game position is \((\phi_i, s_i)\), then play proceeds as follows:

1. Free variables \(X\) do not occur in the game — their role is taken over by “colours”.

Building a game from a formula in \(qM\)

2. If \(\phi_i\) is \(A\) then the game terminates in position \((y)\) where \(y = VA.s_i\).

3. If \(\phi_i\) is \(\{k\}\) then the distribution \(V.k.s_i\) is used to choose either a next state \(s'\) in \(S\) or possibly the payoff state \(S\). If a state \(s'\) is chosen, then the next game position is \((\phi_i, s')\); if \(S\) is chosen, then the next position is \((y)\), where \(y\) is the payoff \(V.k.s_i.(1 + \sum_{s'} V.k.s'.s')\), and the game terminates.

4. If \(\phi_i\) is \(\phi' \land \phi''\) (resp. \(\phi' \lor \phi''\)) then \(\text{Min}\) (resp. \(\text{Max}\)) chooses one of the conjunctions (disjunctions): the next game position is \((\phi_i, s_i)\), where \(\phi\) is the chosen “and” \(\phi'\) or \(\phi''\).

5. If \(\phi_i\) is \(\phi' \land G> \phi''\), the next game position is \((\phi', s_i)\) if \(V.G.s_i\) holds, and otherwise it is \((\phi'', s_i)\).

6. If \(\phi_i\) is \((\mu X \cdot \phi)\) then a fresh colour \(C\) is chosen and is bound to the formula \(\phi_{[X \to C]}\) for later use; the next game position is \((C, s_i)\).

7. If \(\phi_i\) is \((\nu X \cdot \phi)\), then a fresh colour \(C\) is chosen and bound as for \(\mu\).

8. If \(\phi_i\) is a colour \(C\), then the next game position is \((\Phi, s_i)\), where \(\Phi\) is the formula bound previously to \(C\).
Building a game from a formula in \( q \text{M} \)

1. If \( \phi \) is a formula, then the next position is \((\phi, s_i)\), where \( \phi \) is the chosen "junct \( \lor \) or \( \land \)."
2. If \( \phi \) is \((\phi, s_i)\) then the next position is \((\phi', s_i)\) if \( V \cdot s, s' \) holds, and otherwise \((\phi, s_i)\)
3. If \( \phi \) is \((\phi', s_i)\) the next position is \((\phi', s_i)\) if \( V \cdot s, s' \) holds, otherwise \((\phi, s_i)\)
4. If \( \phi \) is \((\mu X \cdot \phi), s_i)\) the next position is \((\phi[X \rightarrow C], s_i)\)
5. If \( \phi \) is \((\phi[X \rightarrow C], s_i)\) the next position is \((\phi, s_i)\) if \( V \cdot s, s' \) holds, otherwise \((\phi[X \rightarrow C], s_i)\)
6. If \( \phi \) is \((\mu X \cdot \phi), s_i)\) the next position is \((\phi[X \rightarrow C], s_i)\)
7. If \( \phi \) is a colour \( C \), then the next position is \((\Phi, s_i)\), where \( \Phi \) is the formula bound previously to \( C \).
Building a game from a formula in qMμ

If the current game position is \((φ_i, s_i)\), then play proceeds as follows:

1. Free variables \(X\) do not occur in the game — their role is taken over by "colours".
2. If \(φ_i \equiv A\) then the game terminates in position \((s)\) where \(y = V.A.s_i\).
3. If \(φ_i \equiv (k)\) then the distribution \(V.K.s\) is used to choose either a next state \(s'\) in \(S\) or possibly the payoff state \(y\). If a state \(s'\) is chosen, then the next game position is \((s, y)\); if \(y\) is chosen, then the next position is \((φ, s)\), where \(y\) is the payoff \(V.K.s.y/(1 - \sum_{s'} V.K.s.s')\), and the game terminates.
4. If \(φ_i \equiv φ' \sqcap φ''\) (resp. \(φ' \sqcup φ''\)) then Min (resp. Max) chooses one of the minjuncts (maxjuncts): the next game position is \((φ, s)\), where \(φ\) is the chosen "junct" \(φ'\) or \(φ''\).
5. If \(φ_i \equiv φ' \sqsubseteq G\sqsupseteq φ''\), the next game position is \((φ', s)\) if \(V.G.s\) holds, and otherwise it is \((φ'', s)\).
6. If \(φ_i \equiv (μX \cdot φ)\) then a fresh colour \(C\) is chosen and is bound to the formula \(φ|_{X=C}\) for later use; the next game position is \((C, s)\).
7. If \(φ_i \equiv (νX \cdot φ)\), then a fresh colour \(C\) is chosen and bound as for \(μ\).
8. If \(φ_i\) is a colour \(C\), then the next game position is \((Φ, s)\), where \(Φ\) is the formula bound previously to \(C\).
Building a game from a formula in qMμ

If the current game position is \( \langle \phi, s_i \rangle \) then play proceeds as follows:

1. Free variables \( X \) do not occur in the game — their role is taken over by "colours".

2. If \( \phi \) is a then the game terminates in position \( y \) where \( y = V.A.s_i \).

3. If \( \phi \) is \( \{ k \} \phi \) then the distribution \( V.k.s_i \) is used to choose either a next state \( s' \) in \( S \) or possibly the payoff state \( S' \). If a state \( s' \) is chosen, then the next game position is \( \langle \phi, s' \rangle \); if \( S' \) is chosen, then the next position is \( (y), \) where \( y \) is the position \( y = V.k.s', \) and the game terminates.

4. If \( \phi \) is \( \phi' \land \phi'' \) (resp. \( \phi' \lor \phi'' \)) the game position \( \langle \phi, s_i \rangle \) where \( \phi \) is the chosen "conjunction (resp. disjunction)"

5. If \( \phi \) is \( \phi' \land \phi'' \lor \phi''' \) then \( \phi' \land \phi'' \lor \phi''' \) chooses the game position \( \langle \phi', s_i \rangle \) if \( \phi' \land \phi'' \) holds, and otherwise it is \( \langle \phi', s_i \rangle \).

6. If \( \phi \) is \( (\mu X \cdot \phi) \) then \( \phi \) is the chosen and is bound to the formula \( \phi(X_{\phi'} \cap C) \) for later use. The chosen position is \( \langle C.s_i \rangle \).

7. If \( \phi \) is \( (\nu X \cdot \phi), \) then a position is chosen and bound as for \( \mu \).

8. If \( \phi \) is a colour \( C \), then the next game position is \( \langle \phi, s_i \rangle \), where \( \phi \) is the bound position previously to \( C \).
Building a game from a formula in qM

If the current game position is \((φ_i, s_i)\), then play proceeds as follows:

1. **Free variables** \(X\). If \(φ_i\) is \(X\) then the variables \(Y\) are bound to \(Y = \forall A, s_i\).
2. If \(φ_i\) is \(A\) then the current game position is \((φ_i, s_i)\).
3. If \(φ_i\) is \(k\) the next game position is \((φ_i, s_i)\) if \(φ_i\) is chosen, or \((φ_i, s_i)\) if \(s_i\) is chosen, then the next game position is \((φ_i, s_i)\).
4. If \(φ_i\) is \(φ_i ∩ φ_i\) (resp. \(φ_i ∪ φ_i\)) then \(M_{\phi_i} \) (resp. \(M_{\phi_i} \)) chooses one of the minjuncts (maxjuncts): the next game position is \((φ_i, s_i)\), where \(φ_i\) is the chosen minjunct \(φ_i\) or \(φ_i\).
5. If \(φ_i\) is \(φ_i' ∩ \phi_i''\) the next game position is \((φ_i, s_i)\) if \(V.A, s_i\) holds, and otherwise it is \((φ_i, s_i)\).
6. If \(φ_i\) is \(μX, φ_i\) then a fresh colour \(C\) is chosen and is bound to the formula \(φ_iX = C\) for later use; the next game position is \((C, s_i)\).
7. If \(φ_i\) is \(ηX, φ_i\), then a fresh colour \(C\) is chosen and bound as for \(μ\).
8. If \(φ_i\) is a colour \(C\), then the next game position is \((φ_i, s_i)\), where \(φ_i\) is the formula bound previously to \(C\).
Playing the game

The maximising player must decide whether to “go left” or “go right”, although he does not yet know what the minimising player will do.

The minimising player must decide whether to “go left” or “go right”; he does know what the maximising player did.

Strategies are the key. With these strategies, the game has value \( \frac{1}{2}(0.60) + \frac{1}{2}(0.50) = 0.55 \).

Playing a game: a simpler example

Do strategies have memory?
A strategy is a function from game-paths to Boolean, decided in advance.

When the strategies are applied, the “reduced” game-tree is purely probabilistic, and has a value.

In this case, the value is
\[ \frac{1}{2} \times 0.60 + \frac{1}{2} \times (\frac{1}{2} \times 0.60 + \frac{1}{2} \times 0.50), \]
that is 0.575.

Thus a “full” game, with its maximising and minimising nodes, is a function...

...from strategy-pairs (one for Max and one for Min)...

...to [0,1] (which is the value of the reduced, purely probabilistic game remaining after the strategies have been applied).
Reasoning about strategies

Thus a “full” game, with its maximising and minimising nodes, is a function from strategy-pairs (one for Max and one for Min) to [0,1] (which is the value of the reduced, purely probabilistic game remaining after the strategies have been applied).

\[ \sigma \] is the maximising strategy \( \sigma \) is the minimising strategy

The value of a game played from formula \( \phi \) and initial state \( s \), with fixed strategies \( \sigma, \tau \), is given by the expected value

\[ \int \text{Val} \left[ \sigma ; \tau \right] s \]

of \( \text{Val} \) over the (probability distribution determined by the) game-tree \( \left[ \phi ; \sigma, \tau \right] s \) generated by the formula, the strategies and the initial state.

Reasoning about strategies: the game is well defined

Thus a “full” game, with its maximising and minimising nodes, is a function from strategy-pairs (one for Max and one for Min) to [0,1] (which is the value of the reduced, purely probabilistic game remaining after the strategies have been applied).

\[ \sigma \] is the maximising strategy \( \sigma \) is the minimising strategy

The value of a game played from formula \( \phi \) and initial state \( s \), with fixed strategies \( \sigma, \tau \), is given by the expected value

\[ \int \text{Val} \left[ \sigma ; \tau \right] s \]

of \( \text{Val} \) over the (probability distribution determined by the) game-tree \( \left[ \phi ; \sigma, \tau \right] s \) generated by the formula, the strategies and the initial state.
Reasoning about strategies: the game is well defined

\[ \bigcap_{\sigma} \bigcup_{\phi} \int Val_{[\phi]_V^s} = \bigcup_{\phi} \bigcap_{\sigma} \int Val_{[\phi]_V^s} \]

\( \sigma \) is the maximising strategy \( \sigma \) is the minimising strategy

It doesn’t matter who “goes first” in selecting an overall strategy.

On the left, for any strategy Min might select, we allow Max subsequently to choose the best “counter-strategy” from his point of view. The best Min can do under these circumstances is the minimax value of the game.

On the right we have the maximin value, where Max goes first.

Reasoning about strategies: the game is well defined

\[ \bigcap_{\sigma} \bigcup_{\phi} \int Val_{[\phi]_V^s} = \bigcup_{\phi} \bigcap_{\sigma} \int Val_{[\phi]_V^s} \]

\( \sigma \) is the maximising strategy \( \sigma \) is the minimising strategy

It doesn’t matter who “goes first” in selecting an overall strategy.

On the left, for any strategy Min might select, we allow Max subsequently to choose the best “counter-strategy” from his point of view. The best Min can do under these circumstances is the minimax value of the game.

On the right we have the maximin value, where Max goes first.

The game is well defined...

\[ \bigcap_{\sigma} \bigcup_{\phi} \int Val_{[\phi]_V^s} = \bigcup_{\phi} \bigcap_{\sigma} \int Val_{[\phi]_V^s} \]

\[ \sqcup_{\phi} [\phi]_V^s \]

\[ \sqcap_{\phi} [\phi]_V^s \]

... and is equal to the logical interpretation of the formula that generated the game.

The (logical) interpretation of \( q \mathbf{M} \)

\[ \| \{ k \} \phi \|_{v \cdot s} \triangleq \mathcal{V} \cdot k \cdot s \cdot s + \int \| \phi \|_{\mathcal{V} \cdot k \cdot s} \]

Denotations \( \| \phi \|_{v} \) are random variables over \( S \), that is of type \( S \rightarrow [0,1] \). Thus the effect of \( k \) — the function \( \{ k \} \) — to speak — is to transform one of these random variables into another. It can be considered to be of type

\[ (S \rightarrow [0,1]) \rightarrow (S \rightarrow [0,1]) \]

The random variables are a complete lattice with the order (pointwise-extended) \( \leq \) and so have least/greatest element the everywhere-zero/everywhere-one function respectively.

Least and greatest fixed points are taken within this lattice.

---

The transition \( k \)

\[ \| \{ k \} \phi \|_{v \cdot s} = \begin{cases} 2/5 \\ + & 1/4 \times \| \phi \|_{v \cdot s'_1} \\ + & 1/4 \times \| \phi \|_{v \cdot s'_2} \end{cases} \]
The “value” of a game: reprise

This game has value .55

Iterate to a solution

This game has value .55

Iterate to a solution

This game has value .55

Iterate to a solution
This game, with the strategies shown, has value 0.575; min has not played optimally.

This game has value 0.55.

Proof of equivalence

Note that we do not assume the minimax/maximin value of the game is well defined; that is a corollary of the proof to come.

Step 1: We augment the denotational semantics so that it too takes two strategy arguments, and show equivalence of the game- and denotational interpretations when the strategies are fixed.

For all closed $qM\mu$ formulae $\phi$, valuations $V$, states $s$ and strategies $x_1, x_2$, we have

$$\int V_{x_1} V_{x_2} = \|\phi\|_{x_1} V_{x_2} .$$

The argument uses routine denotational techniques — structural induction etc. — and is detailed only in the case of fixed points.

Proof of equivalence (Step 2)

Step 2: We show that memoriless strategies suffice in the logical interpretation. (For this we must restrict to finite state-spaces.)

For any formula $\phi$, possibly containing strategy operators $\sqcap/\sqcup$, and valuation $V$, there are state-predicate tuples $G/G$ — possibly depending on $V$ — such that

$$\|\phi\|_V = \|\phi\|_V = \|\phi\|_V .$$

We are stating that for any formula (containing maximum and minimum operators), there are two logically equivalent Boolean choices: one in which all minima have been replaced by Boolean choices (but leaving the maxima); and another in which all maxima have been replaced by Boolean choices (but leaving the minima).
**Proof of equivalence (Step 2)**

The argument for Step 2 is straightforward except in the case of maxima within least fixed-points (or minima within greatest fixed-points).

For those cases a considerably more involved analytical argument is used, based on techniques employed by Everett.


**Conclusion: The equivalence is the reason that for this...**

The overall game is described by this “pseudo-code” loop, with a `resultis` construct that gives the final result.

If the loop is unending, the result is taken to be zero (as for a least fixed-point.)
...we can do this.

In Mathematica

Fixed-point applied to all initial states \( 0 \leq v \leq \maxV \), with \( c \) at the maximum and initial share increase probability shown.

```mathematica
InitialIncreaseProb = 0.5

fp0 = FixedPoint[f0, zeroExp]
```

Initial share values:
- Initial share value 0 gives highest expected maturity value 4.156955.
- Initial share value 1 gives highest expected maturity value 4.295363.
- Initial share value 2 gives highest expected maturity value 4.553057.
- Initial share value 3 gives highest expected maturity value 4.877645.
- Initial share value 4 gives highest expected maturity value 5.250896.
- Initial share value 5 gives highest expected maturity value 5.523376.
- Initial share value 6 gives highest expected maturity value 6.000000.
- Initial share value 7 gives highest expected maturity value 7.000000.
- Initial share value 8 gives highest expected maturity value 8.000000.
- Initial share value 9 gives highest expected maturity value 9.000000.
- Initial share value 10 gives highest expected maturity value 9.500000.