Solving Tree Problems using Category Theory

Rafik Hadfi

Monash University

18 July 2018

General Problem-solving

Introduction

- General Problem-solving is one of the main goals of AI since the early days of Computer Science.
- Progress in separate domains, using
  - Algorithms, Computational Game Theory, Computer Vision, Machine Leaning, Automated Artificial Agents, etc.
- Yet current AI is incapable of reaching the human ability to solve wide ranges of problems.
- Humans are good at solving problems because they can reason about unknown situations. They are capable of asking hypothetical questions that can effectively be answered through **analogical reasoning**.

General Problem-solving
Summary

1. Formalising analogical reasoning for general problem-solving.
2. Proposing a category-theoretic formalism for a class of problems represented as arborescences. Many real-world and AI problems are amenable to such structures.
3. Combing problems and solutions into two distinct categories, allowing us to define equivalence classes on problems (Metric).
4. Proving the existence of functors between the categories and its algorithmic interpretation.
5. Proposing an implementation of the functor as a Deep Neural Network.

- Analogical reasoning is when concepts from one space are mapped to the concepts of another space after noticing structural similarities or equivalences between the two.
  - General situations involving images, sounds, interactions, etc.
  - Complex tasks like puzzles, sports, etc.
- Solving problems using analogies requires the ability to identify relationships amongst complex objects and transform new objects accordingly.
- An analogy is usually described as «A is to B as C is to D»
  - Portugal : Lisbon :: Russia: ?

General Problem-solving

Analogical Reasoning

- Analogical reasoning is when concepts from one space are mapped to the concepts of another space after noticing structural similarities or equivalences between the two.
  - General situations involving images, sounds, interactions, etc.
  - Complex tasks like puzzles, sports, etc.
- Solving problems using analogies requires the ability to identify relationships amongst complex objects and transform new objects accordingly.
- An analogy is usually described as «A is to B as C is to D»
  - Portugal : Lisbon :: Russia: ?
  - Portugal : Lisbon :: Russia: Moscow

General Problem-solving
Analogical Reasoning

- Analogical reasoning is when concepts from one space are mapped to the concepts of another space after noticing structural similarities or equivalences between the two.
  - General situations involving images, sounds, interactions, etc.
  - Complex tasks like puzzles, sports, etc.
- Solving problems using analogies requires the ability to identify relationships amongst complex objects and transform new objects accordingly.
- An analogy is usually described as «A is to B as C is to D»
  - Portugal : Lisbon :: Russia: ?
  - Portugal : Lisbon :: Russia: Moscow
  - Bill: Hillary :: Barack : ?

- Analogical reasoning is when concepts from one space are mapped to the concepts of another space after noticing structural similarities or equivalences between the two.
  - General situations involving images, sounds, interactions, etc.
  - Complex tasks like puzzles, sports, etc.
- Solving problems using analogies requires the ability to identify relationships amongst complex objects and transform new objects accordingly.
- An analogy is usually described as «A is to B as C is to D»
  - Portugal : Lisbon :: Russia: ?
  - Portugal : Lisbon :: Russia: Moscow
  - Bill: Hillary :: Barack : ?
  - Bill: Hillary :: Barack : Michelle

General Problem-solving

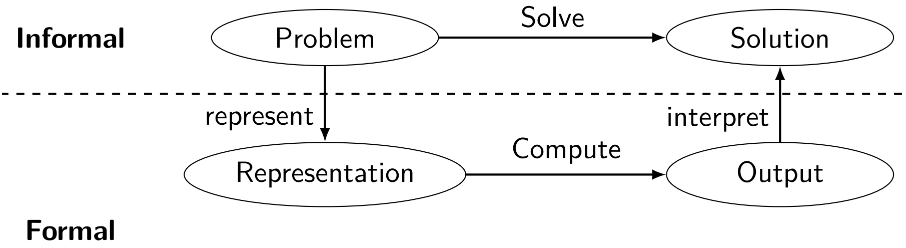
Problem representation

- Despite their intuitive appeal, analogies do have the drawback that, if the structure is not shared across the full problem space, we might end up with a distorted understanding of a new problem than if we had not tried to think analogically about it.
- It is therefore crucial to find a formalism that translates problems into the **representation** that allows comparisons and transformations on its structures.

General Problem-solving

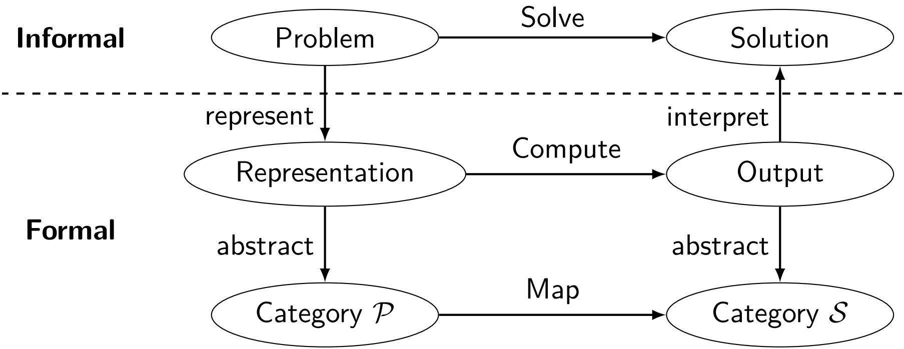
Problem representation

Role of Representation in Problem-solving



[1] Poole, David L., and Alan K. Mackworth. Artificial Intelligence: foundations of computational agents. Cambridge University Press, 2010.

Abstracting problems and solutions

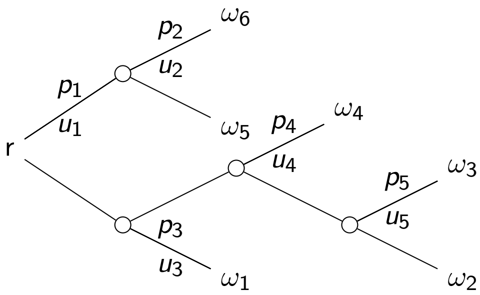


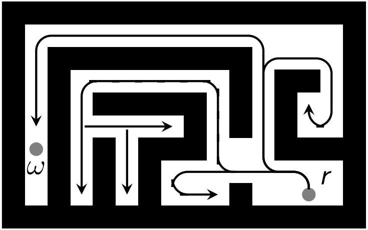
Tree Problems

How to represent problems in the most abstract fashion?

Tree Problems

How to represent problems in the most abstract fashion?

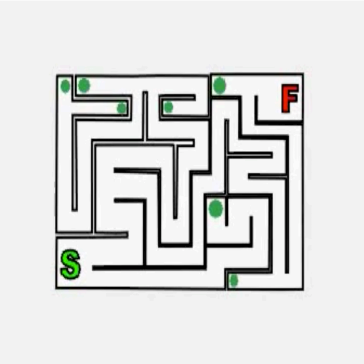




$$\pi^* = \arg\max_{\pi \in \mathscr{P}} \sum_{\substack{e \in \pi \\ \pi : r \to \omega}} u(e) \log p(e)$$

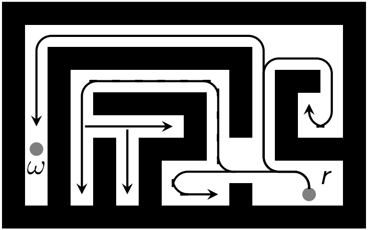
Tree Problems

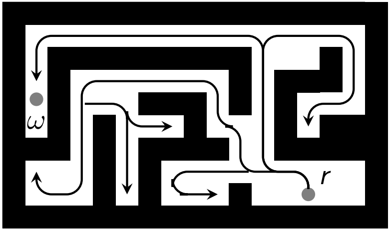
How to represent problems in the most abstract fashion?



Tree Problems

How to represent problems in the most abstract fashion?





Tree Problems

Definition

Definition

We define tree problems as an umbrella term for a class of problems in AI. Such problems are encountered in Decision-making, Maze Search, and Algorithmic Game Theory.
Given a directed rooted tree with predefined edge labels and a set of terminal vertices, the corresponding tree problem possesses at most one solution. The solution corresponds to a path from the root of the tree to one of its terminal nodes.

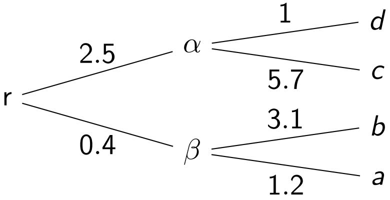
Formally, a problem $\mathcal{P}$ is represented by the tuple $T_{\mathcal{P}} = (T, \mathcal{L}, \mathcal{A})$, defined as following.

- The tuple $T = (r, V, E)$ is a labelled tree with root r, a set of nodes V, and a set of edges $E \subseteq V \times V$. The set V is partitioned into a set of internal nodes I and a set of terminal nodes Ω. We note $V(T)$ and $E(T)$ as shorthands for the vertices and edges of T.
- The tuple $\mathcal{L} = (\mathcal{L}_V, \mathcal{L}_E)$ defines the "labelling" functions $\mathcal{L}_V : V \to \mathbb{R}^n$ and $\mathcal{L}_E : E \to \mathbb{R}^m$. The numbers n and m are respectively the numbers of vertice and edge features.
- The algorithm $\mathcal{A} : T \mapsto S_{\mathcal{P}}$ implements an objective function that assigns solution $S_{\mathcal{P}}$ to T.

Tree Problems

Characteristic forms

- Tree T



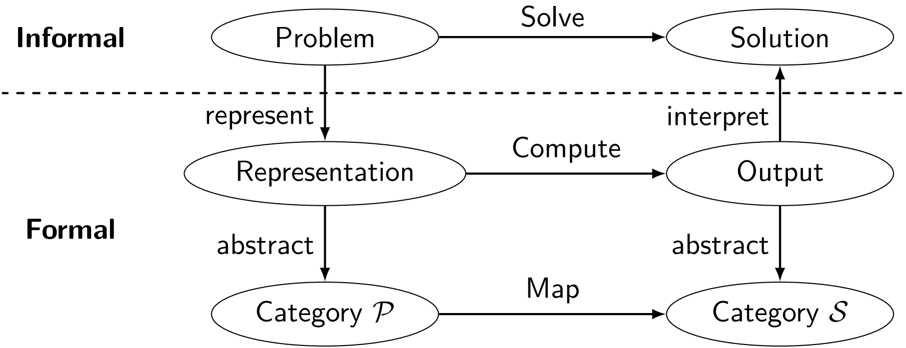
- Characteristic matrix of T

$$M_T = \begin{array}{c} \\ \end{array} \begin{matrix} ab & ac & ad & bc & bd & cd & p_a & p_b & p_c & p_d \end{matrix}$$
$$M_T = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0.4 & 0 & 0 & 0 & 0 & 2.5 & 1.2 & 3.1 & 5.7 & 1 \end{pmatrix}$$

- The characteristic form $\phi_\lambda(T) = \lambda M_T$ is parameterised by $\lambda \in [0,1]^{m+1}$ with $\sum_{j=1}^{m+1} \lambda_j = 1$.

From Problems to Categories

From Problems to Categories

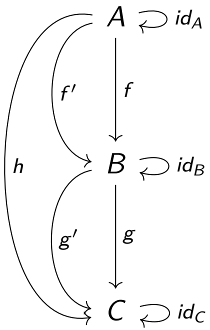


"An abstract way of representing things and ways to go between things."

A category $\mathcal{C}$ is defined by its

- **Objects** $Ob(\mathcal{C})$
- **Morphisms** $Mor_{\mathcal{C}}$
  - if $A, B \in Ob(\mathcal{C})$, then $Mor_{\mathcal{C}}(A, B)$ is a collection of morphisms, and $f : A \to B$ means $f \in Mor_{\mathcal{C}}(A, B)$
- **Composition law** $\circ$
  - associates to each $f : A \to B$ and $g : B \to C$ a morphism $g \circ f : A \to C$.
  - has a neutral element: $\forall X \in Ob(\mathcal{C}),\ \exists id_X : X \to X$ such that $\forall f : A \to B,\ id_B \circ f = f = f \circ id_A$
  - is associative: $(h \circ g) \circ f = h \circ (g \circ f)$

Commutative diagram:

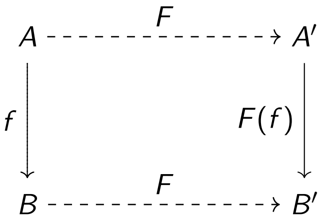


Categories are everywhere.

- *Set*: (Sets, Functions, $\circ$)
- *Mon*: (Monoids, Monoid morphisms, $\circ$)
- *Vec*: (Vector spaces, Linear functions, $\circ$)
- *Grp*: (Groups, Group morphisms, $\circ$)
- *Graph*: (Vertices, Paths, Path concatenation)
- *Hask*: (Haskell types, Functions, (.))
- Any deductive system: (Theorems, Proofs, Proof concatenation)
- ...

Another important notion is that of a (covariant) functor, which is a morphism of categories. A functor $F : \mathcal{C} \to \mathcal{C}'$ is made of

- A function mapping objects to objects, $F : Ob(\mathcal{C}) \to Ob(\mathcal{C}')$.
- For any pair of objects $A, B \in \mathcal{C}$, we have $F : Mor_{\mathcal{C}}(A, B) \to Mor_{\mathcal{C}'}(F(A), F(B))$ with the natural requirements of identity and composition:
  - Identity: $F(id_A) = id_{F(A)}$
  - Composition: $F(f \circ g) = F(f) \circ F(g)$

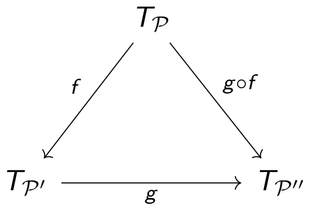


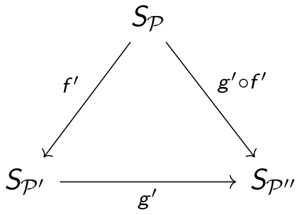
Functors will be later used to formalise analogies between problems.

Theorems

- Tree problems define a category $\mathcal{T}$.
- The solutions to tree problems define a category $\mathcal{S}$.

Proof: In order for $\mathcal{T}$ (resp. $\mathcal{S}$) to be a category, we need to characterise its objects $Ob(\mathcal{T})$, morphisms $Mor_{\mathcal{T}}$ and their laws of composition.



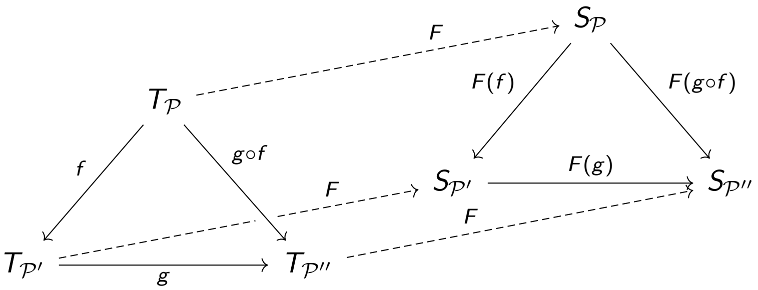


From Problems to Categories

Problem and Solution Categories

Theorem

There exists a functor F from the category of tree problems $\mathcal{T}$ to the category of solutions $\mathcal{S}$.



- The hypothesis is that given $\mathcal{T}$ and $\mathcal{S}$, it is possible to exploit analogies between solved and unsolved problems using functors.
- The analogy «S' is to S as P' is to P» translates to $P \xrightarrow{f} P' \xRightarrow{\mathcal{F}} S \xrightarrow{\mathcal{F}f} S'$.
- Mapping problems to solutions requires a level of identification between the two. Isomorphisms are in general rare and difficult to characterise. We need to "weaken" the isomorphism by descending from isomorphism of categories to equivalence of categories.

$$\underset{(Weaker)}{Functor} \succ Adjunction \succ Equivalence \succ Isomorphism \succ \underset{(Stronger)}{Identity}$$

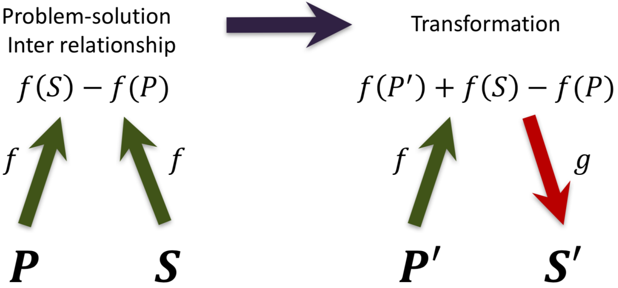
- The equivalences of categories of trees $(\mathcal{T}/\simeq)$ define what can be identified as the level of similarities or analogy between the problems that they represent. Similarly, the equivalence of categories of solutions $(\mathcal{S}/\simeq)$ defines the levels of similarities between solutions.
- If the tree $T_{\mathcal{P}} \in Ob(\mathcal{T})$ is analogous to other trees $\{T_{\mathcal{P}'}\}_{\mathcal{P}' \neq \mathcal{P}}$, it will be useful to find the "most" analogous ones.
- This could be done using the metrics
  - $d_\lambda(T_{\mathcal{P}}, T_{\mathcal{P}'}) = \|\phi_\lambda(T_{\mathcal{P}}) - \phi_{\lambda'}(T_{\mathcal{P}'})\|$ is a metric on $Ob(\mathcal{T})$.
  - $d(S_{\mathcal{P}}, S_{\mathcal{P}'}) = \|S_{\mathcal{P}} - S_{\mathcal{P}'}\|$ is a metric on $Ob(\mathcal{S})$, $S_{\mathcal{P}}, S_{\mathcal{P}'} \in \{0, 1\}^n$.
- Usage in learning and in accelerating the convergence when "training" the functor.

Implementing the functor
Idea

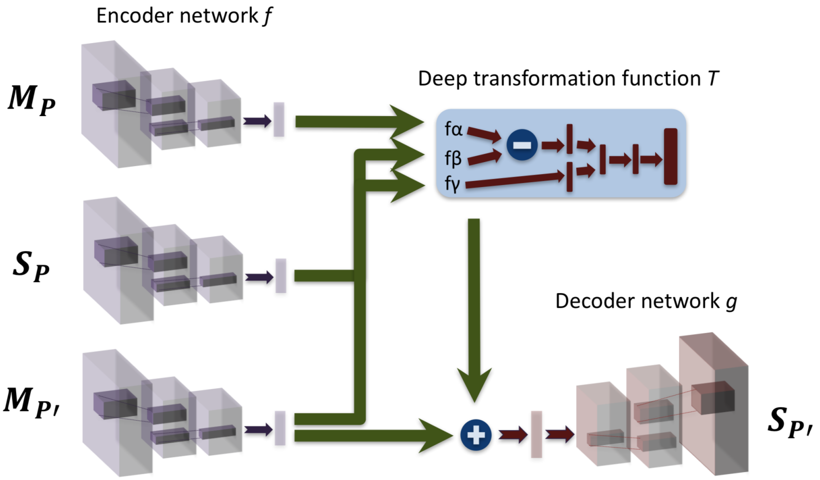
- Learning the functor as a function that reproduces an algorithm or a nonlinear system.
- An intuition from the **Universal Approximation Theorem**
  - A feed-forward network with a single hidden layer containing a finite number of neurones can approximate continuous functions on compact subsets of $\mathbb{R}^n$, under mild assumptions on the activation function.
- Can we construct the functor as a Deep Neural Network?

Implementing the functor
Sketch

1. Define the mappings in $P \xrightarrow{f} P' \xRightarrow{\mathcal{F}} S \xrightarrow{\mathcal{F}f} S'$
2. Embedding P, S, P', S' in the same space $\mathcal{E}$ to allow the encoding
3. Find the inter relationship between P and S as $f(S) - f(P)$
4. Transformation of $f(P') + f(S) - f(P)$
5. Decoding $g(f(P') + f(S) - f(P))$



Implementing the functor
Architecture



Conclusion and future directions

- General problem representation using arborescent structures.
- Category-theoretic formulation of problems and solutions.
- Solving problems using functors, constructed as Deep Neural Networks.
- Next
  - Complexity of the functor $O(1)$ vs. the actual algorithm $O(n)$.
  - Apply to Raven's Progressive Matrices as a general test of intelligence.
  - Neural basis for analogy-making?