#4430--268

Shortest Path in a Combined Street and Public Transportation Network: Cognitive Agent Simulation in a Product of Two State-Transition Networks

Andrew U. Frank,
frank@geoinfo.tuwien.ac.at
TU Wien, Department of Geoinformation and Cartography,
Gusshausstrasse 27-29/E127,
A-1040 Vienna, Austria

Note: figures will be improved for final publication

Abstract: Location Based Services to assist travelers in wayfinding are a prime application for expert system techniques. The use of public transportation leads nearly always to a combination of different services from different providers (multi-modal transportation). Information systems must combine data for the different services and produce advice to navigate in space and obtain the right tickets, reservations, etc. This information can be seen as two (or more) state-transition diagrams: one for the spatial navigation and one for the business (ticketing, validation, reservation) rules.
A (categorical) product combines two state-transition diagrams. The implementation is immediate using an intuitionistic logic reasoner built into the programming language, which infers typing for second order, polymorphic functions and allows their safe execution. The shortest path algorithm in this combined network produces sound advice and reminds the user to acquire tickets and plans the necessary navigation to ticket vending machines, etc.
The approach combines typical expert system technologies like inference engines with object-oriented programing; recent advances increase the level of reasoning possible during compilation. The use of a high-level programming language with substantial inference power facilitates the formalization of domain knowledge and is a viable alternative to the classical expert system architecture.

"(Playing Chopin's etudes) taught me a lifelong lesson: that phenomena perceived to be magical are always the outcome of complex patterns of nonmagical activities taking place at a level below perception. In other words, the magic behind magic is pattern."
Douglas Hofstadter, SCIENTIFIC AMERICAN, April, 1982, p. 17.

1 Improve the quality of mobility

It is widely accepted that individual car transportation is not a viable solution to increase and improve mobility in European cities, but to entice people to use public transportation is notoriously difficult. The standard argument is that public transportation is not available when and where it is required, which is certainly true for some mobility demands, but investigations show that many potential passengers do not use public transportation even when it is available. The simple reason: potential users do not know when and where public transportation is available and how they could use it; they lack information about schedule and the—often complex—tariff and ticketing rules. Location based services provide the information when and where it is necessary.

drop next para if not enough space:

Economically speaking, the total cost of using public transportation consists of the cost and time used for transportation plus the cost for collecting the information on how to use it; the later cost is high and makes the car an unbeatable alternative for all but repeated trips—for example, commuting to work—where the information cost is amortized over many trips. Effectively, only regular users are found on public transportation, primarily commuters; this gives credibility to studies, which indicate that the number of potential users that do not use public transportation for lack of information is about as large as the number actually using them; these additional users would use available capacity in off hours!

In most cases, the information is available and can be found on the web; most public transportation companies publish their schedules and line maps with stops. Public transportation combines in all but the most simple cases several services from different providers. I considered recently a trip from Vienna to Tangier and a combination of a direct flight to Malaga, a bus to Algeciras, and the ferry to Tangier seemed more attractive than the unreliable multi-stop flight to Tangier. The schedules and line information for the different transportation companies involved in a trip are not coordinated and the effort to collect the information and patch the trip together is a complicated task. An intelligent method to combine the available data and produce the information necessary for the user for the whole trip—door to door—and including ticketing rules, reservation system, etc. is a perfect application for artificial intelligence.

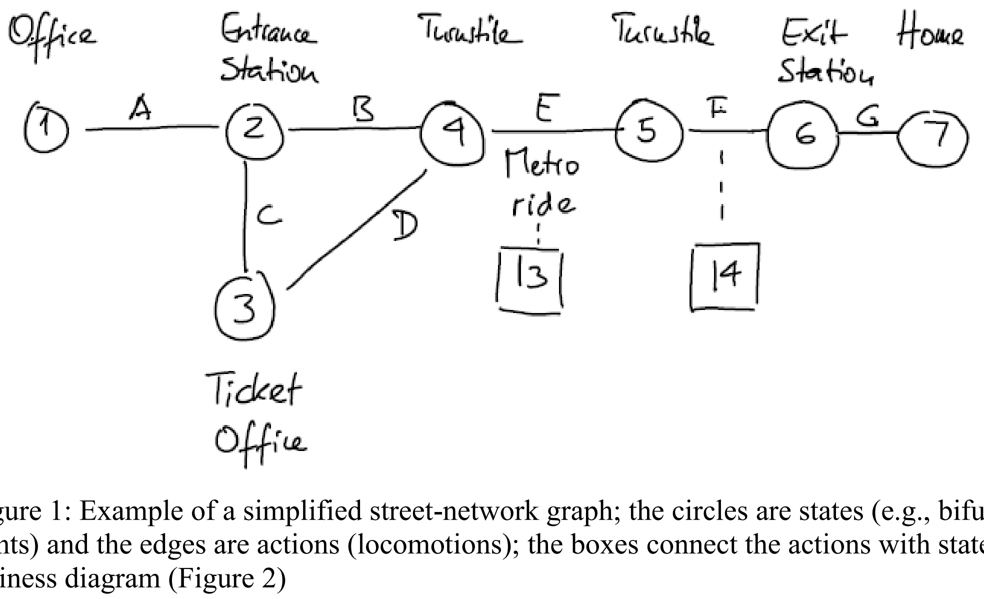

Figure 1: Example of a simplified street-network graph; the circles are states (e.g., bifurcation points) and the edges are actions (locomotions); the boxes connect the actions with states in the business diagram (Figure 2)

The scientific goal of Pontikakis (2006) was to understand how to combine spatial navigation (Figure 1) with the "navigation" of the state-transition diagram of the public transportation business rules (Figure 2). An agent simulation was constructed and demonstrated clearly the typical problems; e.g., plan navigation such that travelers stop at convenient places to buy tickets they need later. Travel agents, railway information staff and similar are the experts that know how to paste together the various data pieces found in published schedules. Some expert system exists for such applications, for example the widely used web pages of European national trains services, which seem to use a common European knowledge base of scheduled transportation and comparable rules for extracting relevant information: schedules for train transportation across Europe are mostly available, including hints about distances and time necessary to change between stations, etc.—lacking are the reservation and ticketing rules in these expert systems.

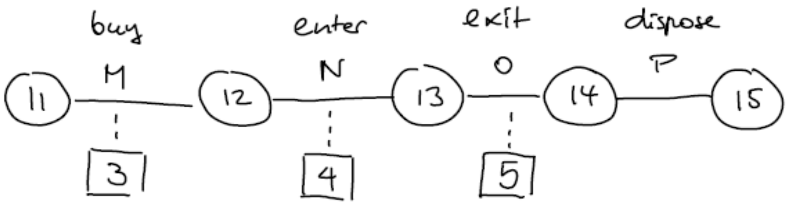

Figure 2: Simplified graph, which describes actions necessary for a legal ride on Vienna's metro; the boxes indicate the places (from Figure 1) where actions can be carried out.

shorten above para if needed

To construct an intelligent advisor for multi-modal, public transportation requires data about schedules, ticketing, etc. and a method to combine these for use by a shortest path algorithm. The contribution here focuses on an abstract description of the combination method and how it is accessed by the shortest path algorithm. Descriptions of the form in which the data must be provided, or specifications for routines to translate existing data to this form, follow from this high level analysis.

2 Expert Systems for Geographic Information

An information system to produce advice for potential users of public transportation is an expert system, which, in its classical architecture, consists of a knowledge base containing formalized expert knowledge and a computerized inference engine to derive conclusions from the stored knowledge, comparable to the advice a human expert would deliver. The famous PROSPECTOR (McCammon 1984) expert system was built according to this paradigm to imitate human experts prospecting for minerals; commercially successful were systems to diagnose or configure technical systems. These systems used logic reasoning based on forward or backward chaining, e.g., Prolog (Clocksin et al. 1981). It was expected that expert system shells would simplify programming to achieve more 'intelligent' programs than were common at the time (accounting, engineering calculations, etc.).

In 1984 we extended our network database (Panda) (Egenhofer 1989) with a Prolog reasoner to experiment with expert systems in GIS (Clocksin et al. 1981), but found the limitations of logic-based programming too restrictive for most GIS problems. The subset of logic used in Prolog (and similar systems) interprets failure to prove as negation—, which is a variant of the closed world assumption: everything of the world is known and what is not known is assumed to be false (not unknown), which is not an acceptable simplification of reasoning for a GIS.

In the 1980s object-oriented programming emerged as an off-spring from the Abstract Data Type research. Programming in a computationally complete language avoids the 'failure as negation' simplification and separates *false* from *unknown*. Expert knowledge is captured as a lattice of objects with operation (e.g., multiple inheritance of C++ (Stroustrup 1991)). This approach focuses on the structure of the domain and not so much on the inference. From HOPE (Bailey 1985) over CAML (Fletcher 2008) we moved to HUGS (Hugs 2008) and use today Haskell (Peterson 1997) with the customary extensions.

Experience shows that a second order, polymorphic, lazy, functional language (Backus 1978) is very well suited to capture expert knowledge: the differences in operations are mapped to overloaded operations, which are selected depending on the type to which the operation is applied. Modeling cognitive agents, which decide on actions based on their simulated perception of their simulated environment, is straight forward (Frank 2000; Raubal 2001; Krek 2002).

3 Navigation

Computerized car navigation systems are probably the most popular application of geographic information systems. They go back to one of the first thorough analyses of an algorithm by computer pioneer Edsger Dijkstra (1976): the task to find the shortest path in a (street) network. Dijkstra's shortest path, or the faster A*, algorithm (Hart et al. 1968) are at the core of the programs running in millions of car navigation systems and produce navigation advice for drivers all over the world.

drop next if not enough space

Car navigation needs, besides the algorithm and the hardware to run it on, a schematic representation of the street-network, usually provided by surveyors and cartographers. Car navigation systems became feasible with the computerized street-network graphs produced in the 80s by the U.S. administration; comparable data became available in Europe not much later. The question how people cognize navigation is a special case of how they cognize space in general (Egenhofer 1995). Reports of (1) cars ending in a river because the navigation system included a bridge where only a ferry is available (Kuhn 1994), (2) slowly narrowing roads were fire brigades regularly unstuck cars and (3) towns complaining about increased truck traffic are all the result of incongruences in the representation of reality in the road database and their use by navigation systems.

For some regular graphs (e.g., complete graphs) analytical solution for some questions are known; in general, however, solutions for applied graph problems cannot be given by analytical methods and require some sort of traversal, for which we use the conceptual framework of multi-agent simulation. In a computational agent simulation a programmed agent traverses a simulated network (graph) and performs at each node and edge some cognitive functions that determine the next action. The behavior of a person navigating in a street-network maps naturally to such a computational agent.

4 Generalization: State Transition Diagram

Navigation in a street-network by an agent (human or simulated) consists of a series of actions that each leads to a new state; when a bifurcation point is reached, a decision about the next action is taken, following some rule incorporated in the agent. The same schema of *perception—decision—action* is applicable to other problems, where a sequence of steps is necessary to achieve a goal. In using a public transportation system, a passenger follows the graph in Figure 2, which describes the business rules; realistic rules may contain bifurcations, when users have options, e.g., buying different types of tickets.

5 Graph products

A manual integration of the two sets of rules is possible to produce a single state transition diagram, but fails automation. Mathematical category theory gives a general solution that allows the integration of two (or more) state transition diagrams formally. Given two navigation problems with state spaces W (e.g., wayfinding, Figure 1) and B (e.g., business, Figure 2), and locomotion and business actions V and T respective, with state-transition diagrams described by two (partial) functions,

$$w: W \times V \rightarrow W$$
$$b: B \times T \rightarrow B,$$

where w is a function encoding the city map with streets and public transportation lines, and b encodes the "business rules" about tickets, how they are obtained, validated, etc. Needed is a combined state-transition diagram with states S and actions P described as function $f: S \times P \rightarrow S$.

The question is how to construct f from w and b? In mathematical category theory (Asperti et al. 1991; Mac Lane 1998) the combined state S is written as a product (pair) of the wayfinding and the business states W and B, and the combined actions P as the sum (union) of the locomotion V and business actions T:

$$S = W \times B$$
$$P = V + T$$

Then the desired function

$$f: S \times P \rightarrow S$$

becomes

$$f: (W \times B) \times (V + T) \rightarrow (W \times B).$$

To construct f, we need two isomorphic functions h and k, which change the pairing, such that

$$h: (W \times B) \times V \rightarrow (W \times V) \times B$$
$$k: (W \times B) \times T \rightarrow W \times (B \times T).$$

From the given functions w and b we construct functions w' and b'

$$w': (W \times V) \times B \rightarrow W \times B$$
$$w' = w \times id \cdot h$$
$$b': (W \times B) \times T \rightarrow W \times B$$
$$b' = id \times b \cdot k$$

and a function l,

$$l: (W \times B) \times (V + T) \rightarrow (W \times B) \times V) + (W \times B) \times T).$$

The function l is isomorphic in the distributive category of sets (Cockett 1992), which we operate in. Combined this gives for *f = [w' , b'] · l = [(w x id) · h, (id x b) · k] · l.*

The second order functions used here are:

id: A → A denotes the function that does nothing: *id a = a*;

(f x g) is the function, which applies *f* to the first part of a pair and *g* to the second part of the pair: *(f x g) (a,b) = (f a, g b)*; and

[f,g] means the application of *f* or *g* to the two parts: *[f,g] (a+b) = f (a) + g (b).*

Two or more navigation problems in real space and "business logic" space can be directly combined with this formula and we produce a state transition function that can be used for shortest path and similar algorithms.

A modern, high-level, functional language realizes very closely such expressions. The implementation of the above described solution for combining the two state-transition diagrams translates directly to an executable computer program (*cross* stands for *x* and *either* for *[]*):

```
h ((w,b),v) = ((w,v), b)
k :: ((W,B), T) -> (W, (B,T))
k ((w,b),t) = (w, (b,t))
w' = cross (w, id) . h
b' = cross (id, b) . k
f = either w' b' . l
```

The representation of the problem of merging two state-transition diagrams must be extended by *a*, the set of rules, which connects the two diagrams: for example the restriction that tickets can only be purchased at the ticket office, or that the turnstile cannot be passed without having a ticket, etc. Such rules are a restriction on the function *f* constructed and represented as a Boolean function *c: (W x B) x (V + T) -> Bool.*

6 Shortest Path

Advice for travelers is produced by an algorithm to find the shortest (optimal) path through the combined graph. An optimal path algorithm requires functions (*[V]* means here a list of *V*)

a: given a state find possible actions: *a : W -> [V]*

w: get next state for given state and action: *w : (W,V) -> W*

c: get cost for action given a state: *c : (W, V) -> C*

The functions *a* and *w* can be derived from *f* (costs is ignored for simplicity). With these functions in place, a shortest path (e.g., with minimal number of locomotions) is computed by Dijkstra's shortest path algorithm.

An advice to travelers must plan ahead and have the agent add a detour to acquire a token before the necessity is discovered when standing in front of the turnstile and the agent has to backtrack to a place passed before without acquiring the token. A path 1, 2, 4, 3, 4, 5, 6, 7 (in Figure 1 and 2) is not desirable, but how to identify at point 2 the need for a ticket, which becomes only noticeable to the traveler at point 4? The optimal path in the combined (product) graph is, of course, 1, 2, 3, 4, 5, 6, and 7—produced by Dijkstra's shortest path algorithm automatically! The “intelligent” combination achieved the magic.

7 Conclusions

Artificial intelligence is here included in the powerful type checker, which makes programming at this high level of abstraction possible. The development from early Prolog and similar reasoners to the type reasoning system, which is behind the type inference in the code shown (note: not all constructs have programmer written types—most are inferred!). The type system, an extension of the Hindley-Millner (1978) type system based on an intuitionistic (Martin-Löf) type theory (Nordström 1990). These theories were first influential in AI and lead to construction of automatic proof systems, but are now built into the programming languages.

cut next para if space is tight

With such powerful type system in a programming language, new opportunities to write programs capturing more of the semantics of the application domain (i.e., the expert knowledge) appear. The compiler of a language like Haskell can be used by the programmer to do some simple logic, e.g., counting, during compilation (not during execution!). This is useful to write type save programs of matrix operations, where the dimensions of the matrices are fixed at run time and type checking for dimensions is carried out statically—but without writing separate operations for any possible matrix dimension: the compiler can increase or decrease the dimension of a matrix as it may be demanded by the operation (Lämmel 2005). The following fragment of code for defining two types for zero and successor demonstrates how numbers are represented following Peano's axiomatization of natural numbers—and the compiler (not at run time) is carrying this out!

```
data HZero; instance HNat HZero
data HSucc n; instance HNat n => HNat (HSucc n)
```

Such inferences can be used in database programming such that the application program and the database engine use the same language. The gap in current practice, where SQL statements are written as strings and included in programs, can thus be overcome by “artificial intelligence” included in the programming language.

The goal to construct portable navigation aid devices for pedestrians using public transportation systems requires the integration of navigation in space with “navigation in business logic”. Our approach to include “artificial intelligence” in the form of a very powerful type theory and reasoning system and write the application domain (expert) knowledge in an object-oriented style in a functional programming language was useful to clarify step by step notions related to spatial behavior, in particular wayfinding, based on cognitive agent simulation (Frank 2000; Raubal 2001; Krek 2002), but also in other investigations of spatial cognition (Twaroch 2007). Kuhn has used the same type theory to construct ontologies which are not only *is_a* taxonomies but define taxa through applicable affordances, represented as operations (functions) in the class (Kuhn 2007).

The use of a mathematically sound theory (category theory) has produced a method to combine relatively simple pieces and to achieve “intelligent” behavior; confirming Hofstadter's motto.

Acknowledgments

I am grateful for the dissertation work of Alenka Krek, Martin Raubal, and Elissavet Pontikakis, which investigated each one of the topics recorded. Till Mossakowski produced the crucial hint to apply distributive categories.

References (incomplete)

Asperti, A. and G. Longo (1991). Categories, Types and Structures - An Introduction to Category Theory for the Working Computer Scientist. Cambridge, Mass., The MIT Press.

Backus, J. (1978). "Can Programming Be Liberated from the von Neumann Style? A Functional Style and Its Algebra of Programs." CACM 21: 613-641.

Bailey, R. (1985). A Hope Tutorial. BYTE: 235-258.

Cockett, J. R. B. and R. A. G. Seely (1992). Weakly Distributive Categories. Applications of Categories in Computer Science: Proc. LMS Symp. on the Applications of Category Theory in Computer Science, Durham, Cambridge University Press.

Clocksin, W. F. and C. S. Mellish (1981). Programming in Prolog. Berlin, Springer-Verlag.

Dijkstra, E. W. (1976). A Discipline of Programming. Englewood Cliffs, NJ, Prentice Hall.

Egenhofer, M. J. and D. M. Mark (1995). Naive Geography. Spatial Information Theory - A Theoretical Basis for GIS. A. U. Frank and W. Fletcher, D. (2008). "OCAML 2.00."   Retrieved 8. January, 2008, from www.dcs.ed.ac.uk/home/djf/ocaml/.

Frank, A. U. (2000). Communication with Maps: A Formalized Model. Spatial Cognition II (Int. Workshop on Maps and Diagrammatical Representations of the Environment, Hamburg, August 1999). C. Freksa, W. Brauer, C. Habel and K. F. Wender. Berlin Heidelberg, Springer-Verlag. 1849: 80-99.

Hart, P. E., N. J. Nilsons and B. Raphael (1968). "A formal basis for the heuristic determination of minimum cost path." IEEE SSC 4: 100 - 107.

Hugs. (2008). "Hugs 1.4." from http://haskell.systemsz.cs.yale.edu/hugs/.

Krek, A. (2002). An Agent-Based Model for Quantifying the Economic Value of Geographic Information. Vienna, Technical University Vienna. PhD Thesis.

Kuhn, W. (1994). Zur Verwendbarkeit von Geographischen Informationssystemen (GIS). UTA (Umwelt Technologie Aktuell). 5: 88-93.

Kuhn, W. (2007). An Image-Schematic Account of Spatial Categories. 8th International Conference, COSIT 2007, Melbourne, Australia, Springer.

Lämmel, R. and E. Meijer (2005). Mappings Make Data Processing Go' round, Redmond, USA, Microsoft Corp.

Mac Lane, S. (1998). Categories for the Working Mathematician. New York, Berlin, Springer.

McCammon, R. B. (1984). User Guide to the USGS Prospector/KAS Expert System. US Geological Survey: 149.

Milner, R. (1978). "A Theory of Type Polymorphism in Programming." Journal of Computer and System Sciences 17: 348-375.

Nordström, B., K. Petersson and J. M. Smith (1990). Programming in Martin-Löf's Type Theory, Oxford University Press.

Peyton Jones, S., J. Hughes and L. Augustsson. (1999). "Haskell 98: A Non-Strict, Purely Functional Language." from http://www.haskell.org/onlinereport/.

Pontikakis, E. (2006). Wayfinding in GIS: Formalization of Basic Needs of a Passenger When Using Public Transportation. Vienna, Technical University Vienna. Doctor of Technical Science.

Raubal, M. (2001). Agent-Based Simulation of Human Wayfinding -A Model for Unfamiliar Buildings. Vienna, Technical University Vienna. PhD.

Stroustrup, B. (1991). The C++ Programming Language. Reading, Mass., Addison-Wesley.

Twaroch, F. (2007). Sandbox Geography How to Structure Space in Formal Models. Wien, Technische Universität Wien. PhD.

White, R. M. and A. U. Frank (1985). Graphics Programming in Prolog. Orono, ME, Surveying Engineering Program, University of Maine.