The Cerebus Project

Ian Horswill, Rob Zubek, Aaron Khoo, Christopher Dac Le, and Shawn Nicholson

Northwestern University Computer Science Department
1890 Maple Avenue, Evanston, IL 60201
{ian, rob, khoo, le, nicholson}@cs.northwestern.edu

Abstract

The Cerebus project is an attempt to extend parallel-reactive architectures to higher-level cognitive tasks. Cerebus is intended to be a "self-demoing" robot in the sense that it is able to give a short talk about itself and it is able to field (slightly) more detailed questions (asked by typing English on a keyboard) about itself, in addition to demonstrating specific capabilities on demand. We will describe the motivation and techniques underlying the system and present preliminary results.[1]

Introduction

The Cerebus Project[2] is an effort to build a limited artificial person using a Society-of-Mind-like (Minsky 1984) parallel cognitive system. By "artificial person", we do **not** mean a system with human-level capabilities. Nor do we mean an accurate simulation of human cognitive processing. For reasons to be explained shortly, we believe that parallel collections of agencies are better way of implementing high-level cognition on robots than conventional LISP-based methods. Our goal is to demonstrate this by building a robot with all the standard reactive sensory-motor capabilities that can also converse in Pidgin English and reflectively answer queries about its own capabilities. Cerebus is effectively a "self-demo'ing" robot. You can ask it questions like "what can you do?" to which it will reply "Cerebus can follow freespace, follow colors, answer questions, …," listing off its capabilities. You can then tell it "show me freespace following," and it will drive around demonstrating freespace following. We hope that lab members will eventually come to think of it as a member of the community, albeit a rather stupid one, hence the term "artificial person."

[1] Support for this work was provided in part by the National Science Foundation under award #IRI-9625041 and in part by the Defense Advanced Research Projects Agency Mobile Autonomous Robot Software Program under award N66001-99-1-8919 and by the DARPA Distributed Robotics Program and the U.S. Army Soldier Systems Command under award #DAAN02-98-C-4023/MOD P3.

[2] Cerebus is the name of a barbarian aardvark in a saga by Canadian comic artists Dave Sim and Gerhard. Like our robot, Cerebus is short, brutish, stupid, and only speaks of himself in the third person.

Motivation

Robot control systems have traditionally been built out of two distinct kinds of computational "stuff": On the one hand, there are the LISP-like symbolic systems from which theorem provers, planners, and expert systems are traditionally built. On the other hand, there are the parallel, reactive systems popular in behavior-based robotics and connectionist circles. The two classes of machine have complementary strengths and weaknesses. Our ultimate goal is to extend parallel-reactive systems to support higher the higher-level cognitive tasks for which symbolic systems are typically used.

Parallel-reactive systems

Sensory-motor loops are typically built from parallel systems with circuit semantics (Nilsson 1994). They consist of parallel networks of finite-state or zero-state components communicating over fixed connections. Circuit nodes continually recomputed their values over time, allowing the circuit to track changes in the world as sensory data changes. For the purposes of this discussion, we will call this class of computational machinery "parallel-reactive systems."

While parallel-reactive systems are generally implemented on serial hardware, what matters to this discussion is that they are equivalent to parallel circuits and therefore are restricted to the class of computations expressible as such circuits. They are efficient, easy to interface to sensors and effectors, and provide hard real-time performance guarantees.

The problem with these parallel-reactive systems is their limited representational power. Since all representations are ultimately based on wires transmitting scalar activation levels, they are in some sense limited to propositional logic, meaning that it is difficult to represent predicate/argument structure, term expressions, or any other kinds of data that require dynamic graph and tree-structures. Building dynamic graph structures ultimately requires something like a pointer mechanism. While these can be supported in parallel networks by introducing

a switching network, the switching network is expensive and a potential bottleneck. Pointers also threaten to require implementing garbage collection, dynamic type dispatch, and all the other facilities of LISP within a simple circuit-parallel computational model. This is a daunting prospect at best. Moreover, our colleagues in neuroscience always get giggle fits when computer scientists talk about LISP and the brain in the same sentence.

The inability to cleanly represent predicate/argument structure can lead to painfully large networks. One behavior-based dialog system, for example, required separate behavior modules for each possible utterance of each possible speaker (Hasegawa et al. 1997); the behavior-network solution to the blocks-world problem requires distinct nodes for each ground instance of each predicate and action (Maes 1990). While such networks are polynomial in the number of blocks, they're exponential in the order of the domain theory's highest-order predicate or action.

Symbolic systems

Symbolic systems use much more flexible and expressive representations. "Symbolic" is a vague term used in the literature to mean many different things. We use it here to mean serial search algorithms operating on mutable graph structures. In other words: non-deterministic, LISP-like systems. We do *not* use it to mean "language-like," "traditional," or "anything the reader can recognize as traditional AI." Thus truth-maintenance systems, for example, are not symbolic, by this definition, but rather parallel-reactive systems (at least in the steady-state[1]). A better term for this category might therefore be "non-deterministic graph manipulators."

While symbolic systems are expressive, they are expensive ways to support situated activity in a dynamic world. This is usually attributed to the well-known **intractability problem**. Search algorithms are typically non-polynomial. Thus computing an inference can take exponential time or worse. While this is the most common criticism made of symbolic systems, it is not clear that it is the most important, or even a particularly fair one. Worst-case complexity, after all, is worst-case. Many problems can be solved in polynomial time with these techniques; those that cannot probably can't be solved by parallel-reactive systems at all.

A more apt criticism is what we might call the **model-coherency problem**. When we talk about "models" or "world models," we typically think of the centralized database of logical assertions that a symbolic system uses as its source of knowledge about the world. However, robots have *lots* of models of the world – each sensory or motor system typically has its own limited model of the world. These models are all being updated asynchronously by their respective modules. In order for the robot to exhibit coherent behavior, these models need to be consistent with one another across asynchronous updates. Thus, if a tracking system updates its representation of the distance to some object, say a tiger, something must make sure that the symbolic system's representation of the distance to the tiger gets updated too. At present, most systems leave this up to the programmer – the domain theory must be salted with rules to specify when to run epistemic actions.

What's worse is that the symbolic system must also update its representation of whether or not the tiger is a threat. If it had previously been facing away from the agent, it may not have been considered a threat. However, if it has suddenly turned toward the agent, it may have become a threat. Not only must the proposition `not(threat(tiger504))` be retracted and the proposition `threat(tiger504)` asserted, but any inferences based on these propositions must also be revised. Goals may be created, abandoned, or reordered. New plans may need to be computed.

Ideally, a reasoning system should be a kind of tracking system; it should smoothly adjust its representation to track the state of the environment as it evolves. Parallel-reactive systems are successful, when they're successful, precisely because they're good at solving this **model-tracking problem**; as new data comes in through the sensory systems, it flows through the rest of the system, smoothly updating the agent's models of the world. This kind of pipelined data flow is much more difficult to implement in symbolic systems.

Tiered systems

One approach is to use a multi-level or "tiered" architecture in which one or more symbolic systems are run in parallel with a behavior-based system. The symbolic system is responsible to high-level reasoning and planning, while the behavior-based system is responsible for execution of low-level actions and handling of short time-scale contingencies. Systems based on this design have been very successful (Fikes *et al.* 1974, Bonasso *et al.* 1997, Arkin 1998). They provide a way of building a system that support flexible representations and control structures while still being able to respond to many environment contingencies in a timely manner.

Unfortunately, adjoining a symbolic system to a behavior-based system does not solve either the model-coherency problem or the model-tracking problem. There symbolic system still uses a separate database of stored logical assertions that must somehow be kept in sync with the other representations in the system and with the world. Moreover, if there are multiple symbolic systems, such as

[1] By "steady-state," I mean the TMS after all nodes have been added and the system is simply cycling the nodes in and out.

in the 3T architecture (Bonasso *et al.* 1997), the world models of the different symbolic systems must be kept consistent with each other. Coherency and tracking are typically left up to the programmer, who must add rules to the domain model to trigger updates of specific facts in specific cases.

The fact that the planners in three-level architectures (planner, symbolic executive, and behavior-based system) are rarely called in practice is a reflection of this problem.

Supporting inference in parallel-reactive systems

An alternative is to extend parallel-reactive systems to perform the kinds of operations for which symbolic systems are typically used. If successful, this would allow us to build systems with the flexibility of symbolic systems and the responsiveness of reactive systems.

The most important task in extending parallel-reactive systems is making their representations more flexible. There are two general strategies for doing this. On the one hand, we can find techniques for reducing instances of reasoning problems into parallel-reactive systems. In other words, we compile symbolic systems into behavior-based systems. The other strategy is to build a single parallel-reactive system that can simulate a traditional symbolic system, in other words, an interpreter. As with programming language implementation, we can expect the compilation-based strategies to be more efficient and the interpretation-based strategies to be more flexible.

There are clearly limits to how much expressiveness we can add to parallel-reactive systems before they stop being especially parallel or reactive. Exactly what those limits are isn't nearly as clear, however. We could implement a Pentium emulator in the subsumption architecture (Brooks 1986, 1990), but it would be pointless; we'd just have an exceptionally slow Pentium. We could attempt to build a "parallel LISP machine," but since pointer chasing is an inherently serial operation, we would be in danger of negating much of the advantage of using parallel computation in the first place.

As far as we know, most AI algorithms are difficult or impossible to map into neuron-like parallel circuits. For example, unification has been shown to be *P*-complete, meaning that if it's possible to efficiently compile it into a compact feed-forward circuit, then anything in *P* is too, a fact which is generally taken to mean that it isn't "parallelizable."[1] If it isn't parallelizable, then problems like *SAT* (*NP*-complete) or STRIPS planning (*PSPACE*-complete) are presumably right out.

However, the fact that we can't implement all of LISP in behavior-based systems doesn't mean that we can't implement the parts of symbolic AI that we really need in practice. It's very rare to see truly complex reasoning performed on robots in the first place, so in many ways, the parallel-reactive systems don't have that far to catch up.

The key questions, then, in extending behavior-based systems to more complicated reasoning tasks are:

1. What representational capabilities are required in practice for guiding embodied activity?
2. How can those representations be efficiently implemented in parallel hardware with pipelined dataflow?
3. How can any serial inference operations that must be performed be implemented and interface to the parallel systems so as to minimize the impact of their seriality?

Approach

In our work, we have used a combination of the compilation and interpretation strategies, however, we have generally preferred compilation techniques because of their efficiency. We typically have reserved interpretation-based techniques for problems like controlling serial visual search where the problem itself imposes some seriality to begin with and so parallelism is less of an issue.

In the course of our work, we have developed a number of useful tools for building parallel-reactive reasoning systems.

GRL

GRL (**G**eneric **R**obot **L**anguage, a.k.a. "GiRL") is a functional programming language for designing behavior-based systems (Horswill 2000). It provides most of the conveniences of LISP, including polymorphism, functional semantics, higher-order procedures, and sequence functions. Unlike LISP however, all of these features are partially evaluated at compile-time to reduce the program to a network of parallel finite-state machines. The compiler then emits the network as a while loop containing the update rules for each node in the network. The resulting code is statically typed, performs no run-time storage allocation, and can be rendered any a range of languages from BASIC to Scheme. The object code generated by the compiler is typically better than a naïve C programmer would generate. It is also typically more reliable since the semantics of the language make a number of common programming errors impossible.

[1] It is important to take this result with a grain of salt. The fact that a problem can't be solved by a compact feed-forward network does not mean that it can't run considerably faster on a shared-memory multiprocessor or a recurrent network than on a standard serial uniprocessor. It simply means that it can't be made to run in constant- or polylog-time through the mere application of parallelism.

GRL can be viewed as a meta-language implementing application-specific compilers for different architectural features. Compilers for motor Schemas (Arkin 1998), the original subsumption architecture (Brooks 1986), spreading activation (Maes 1989, 1990), plan sequencers, and forward- and backward-chaining inference systems have all been written in GRL. These techniques can then be mixed and matched as appropriate.

GRL is similar in spirit to Kaelbling's REX system (1987) and Schaad's parallel-functional decision trees (1998). However, GRL attempts to provide a cleaner semantics as a programming language.

Role passing

Again, one of the principal problems with extending behavior-based systems is their inability to deal with parameters, predicate/argument structure, and so on. Agre has argued that this problem is ultimately inescapable and that alternative accounts of representation and abstraction are necessary (Agre 1997). His solution to the problem, due also to David Chapman, is based on replacing logic variables bound to logical constants, which are in turn grounded in the world by sensory systems, with indexical constants grounded in the world by sensory systems (Agre and Chapman 1987). Originally called *indexical/functional representation*, the technique is now known as *deictic representation*. The argument for deictic representation is roughly that the constraints of physical embodiment – limited sensory bandwidth, constrained viewpoint – already force upon the designer the need for sensory attention rather than an omniscient perceptual engine; but perceptual attention (e.g. the fact that a robot's color tracker is tracking one thing rather than another) is already a form of variable binding. More glibly: you eyeball already *is* a pointer – you assign the pointer by pointing it at an object and you dereference it by reading out the data from the sensory system.

The major problem with deictic representation is the proliferation of indexical constants. Agre and Chapman's system had indexicals like `the-bee-that-is-chasing-me` and `the-block-I'm-going-to-kick-to-kill-the-bee-that-is-chasing-me`. While this provides a vast improvement of having separate network nodes for every block, bee, and every possible block/bee pair, it's still rather unwieldy. Moreover, it is considerably larger than the number of object trackers that can reasonably be supported by an active vision system. Agre and Chapman therefore had to allocate indexicals to trackers manually at design-time, making sure that conflicts could not arise at run-time when one tracker was needed to represent two different indexicals.

Role passing is a form of deictic representation that solves most of these problems (Horswill 1997). Rather than postulating an unlimited number of indexicals, role-passing implements a fixed set of generic indexical terms named after linguistic roles such as `agent`, `patient`, and `object`. It is also similar to Minsky's *pronomes* (Minsky 1984) and Shastri and Ajjanadadde's SHRUTTI system (Shastri and Ajjanadadde's 1993) except that binding is performed in the sensory systems rather than inside the central system.

When the control system assigns a tracker or other sensory system to an object, it specifies a role to which the object is to be bound. The sensory system is tagged with that role, and it can then be accessed associatively by the tag. More importantly, sets of roles can be represented as compact bit-vectors contained in a single machine word. This allows forward-chaining modal Horn clause inference over a finite domain of roles to be compiled to simple feed-forward Boolean networks. These networks can be simulated on conventional hardware by compiling them to straight-line code consisting only of bit-mask instructions. In effect, it is a lifted version of Kaelbling and Rosenschein's techniques for automated compilation of domain axioms to digital logic (Kaelbling and Rosenschein 1991).

It should be noted that compiling role-passing inference requires hardware exponential in the arity of the highest arity predicate in the rule base, so in practice it is restricted to unary predicates. While this is a severe restriction, it is worth noting that:

1. In many cases, higher-arity relations can be expressed implicitly. When representing a concept like PTRANS, it is obvious that the agent of the PTRANS is whatever object is bound to the `agent` role.
2. In practice, we have found cases where this is insufficient to be relatively uncommon for mobile robot control programs.
3. A slower, serial, symbolic system can always be adjoined to handle the more complex cases, (although this requires care in handling model consistency and tracking).
4. Anything that can be expressed in this language can be compiled into a form that can track all inferences at sensor rates. A rule base of 1000 inference rules can be updated at 100Hz, completely recomputing each predicate on each cycle, using less than 1% of a contemporary desktop computer.

Role passing allows behavior-based systems to abstract over both objects and sensory systems. It allows rules to reason about objects without needing to know which tracker the information came from, or even which sensory

modality; an object can have multiple representations across several different modalities.

The current version of the role-passing compiler is written in GRL.

Higher-order behaviors

As stated so far, roles are typically bound to objects in the world via sensory systems. However, they can also be bound to other kinds of representations or to entities that are internal to the robot such as behaviors. If one provides a mechanism for binding behaviors to roles, one can create behaviors that take other behaviors as arguments.

An example of such a higher-order behavior is the generic search behavior. The search behavior successively enumerates elements from some set. It takes as arguments (through role bindings) actions for resetting the search, selecting the next element of the set, and operating on the element. It alternately triggers the next-element action and the operate action until all the members of the set are exhausted. One can then call the search behavior with the find-door behavior as the next-element argument to get the system to drive to all the offices in the lab. Alternatively, one can call it with the find-person behavior to get it to work a crowd at a conference. (We are ignoring here the issue of realizing when the robot has already talked to a person; this is a difficult issue, but no more difficult with role passing than with a symbolic system).

This sort of systematic enumeration is trivial to implement in a symbolic system provided that all the elements of the set are already represented as data structures internal to the system. However, it is considerably more complicated to implement when the elements of the set are real objects in the world whose identities have yet to be determined, such as our case of working a crowd.

Preliminary implementation

Until recently, most of our work has been focused on building infrastructure like the GRL compiler. We have now begun work on the Cerebus system, as a demonstration of the techniques discussed above. To date, there have only been approximately six person-weeks devoted to implementing the system, however, a proof-of-concept system was demonstrated at the AAAI Robot Challenge in Austin, Texas earlier this year. In addition to the basic techniques discussed above, Cerebus consists of the following components. With the exception of a few specialized out-calls to LISP for the parser and semantic nets, all the components are parallel-reactive systems written in GRL. The exceptions could have been written in GRL, but it would have been somewhat cumbersome. Since we are running the compiled GRL code within Scheme anyhow, it was simpler to write the key portions of the subcategorization frame matching and marker passing directly in Scheme. There are also a number of primitives written in C for image processing and graphics.

Sensory-motor systems

Cerebus' hardware consists of a standard Pentium laptop mounted on a Magellan base from Real World Interface. The Magellan is a 2DOF base arranged in a differential-drive configuration with front- and back-casters for stability. The Magellan is equipped with a sonar ring, infrared proximity sensors (not presently used), bump switches and shaft encoders for velocity and position estimation. All input and output devices are controlled by an on-board 68000-class machine which communicates with the host processor via a serial port. A standard NTSC security camera mounted is mounted on top of the robot and connected to the Pentium via a NogaTech USB frame grabber.

Vision is the robot's primary sensory modality. It performs appearance-based collision avoidance using the Polly algorithm (Horswill 1993) and adaptive color blob tracking using a variant of the k-means clustering algorithm. A post-processing step also searches for leg-like shapes in the depth map generated by the Polly algorithm and tracks them when they're found. These sensory modalities support the standard suite of low-level behaviors, such as freespace following and object tracking (both colored objects and legs), which can be triggered by the higher-level control systems as needed. The robot performs tracking and wandering at approximately 1m/s using 10fps vision processing. All processing, perceptual and otherwise, is performed on-board the Pentium. These vision behaviors are backed up by a wrapper behavior that inhibits forward motion when the sonars detect a nearby obstacle ahead and backward motion when the sonars detect an obstacle behind.

Parser

Cerebus contains a rudimentary finite-state parser for English. The parser contains a lexicon within which roles can be bound at run-time. As words are clocked into the system, the parser looks up their lexical categories and binds them to roles. In the case of words that function as heads of phrases, it also looks up their subcategorization frame(s) for use in determining the roles of their constituents.

Binding of constituents in verb phrases is performed in two phases. First, constituents are bound to the "syntactic roles" `subject`, `object1`, and `object2`. When the end of a phrase is detected, either by the presence of another head or by reaching a full stop, the subcategorization frame(s) of the current head are used to

map the syntactic roles to normal, "semantic" roles. The two-phase mapping makes it easier to handle phenomena such as wh-words, which flip the order of subject and object.

Subcategorization frames can be conditionalized on the set of roles previously bound and/or on the types of the objects to which they're bound. They can also contain defaults within the lexicon for unbound roles. For example, the utterances "bring rob," "bring me rob," and "bring rob to me" all parse as:

```
action=bring
agent=cerebus
patient=rob
destination=user
```

However, the first noun after the verb sometimes parses as the destination role, and other times as the patient.

Roles in prepositional phrases are bound in a single pass, although it would ultimately be useful to treat each preposition as a separate syntactic role, which could then be mapped by the subcategorization frames of the verb. Nouns are not presently allowed to take arguments.

NL Generator

In addition to the motor behavior stack used to implement navigation and locomotion, Cerebus also has a speech behavior stack consisting of behaviors to output fixed strings, to generate noun phrases describing objects bound to roles, and an "um" behavior that generates appropriate grunts when it is the robot's conversational turn but it hasn't generated any speech recently. We anticipate other ambient speech behaviors, such as an "uh-huh" behavior to handle the case where it is the human's turn in the conversation.

The speech behavior stack is sequenced by `generate-VP`, an FSM that generates a description of the current state of the robot's role bindings as a verb phrase.

Semantic net

Cerebus also has a simple symbolic memory in the form of a marker-passing semantic net. The semantic net is used to store information about the robot so that it can answer questions like "what can you do?"

Roles are used as the markers passed between nodes. Each node remembers the set of roles to which it is bound. The basic operations of the system are to mark nodes, to enumerate nodes, and to clear nodes of roles. Nodes that correspond to a word in the lexicon automatically inherit any role bindings from their lexical entry. Further nodes can be marked with an operation that assigns a specified role to all nodes related to a selected node by a specified relation. The selected node is specified by role. Nodes are enumerated with a mark-first operation, that bindings the first node (within a more or less random but static ordering) with a specified role to another specified role. The mark-next operation moves the marker to the next selected node. This gives the system a way of enumerating all the nodes related to a specified node in a specified way.

The semantic net is static in the sense that new nodes and links cannot presently be added to it.

Conclusion

Cerebus is an on-going project designed to scale parallel-reactive components to higher-level cognitive tasks. To date, we have implemented initial versions infrastructure we need for the system. However, there is presently little to flesh it out. The system can answer any English language query, provided it is one of the five or so queries for which is has answers in its knowledge base. Likewise, it can follow any command, provided it's one of the four or so commands it understands. Nevertheless, the system does demonstrate that it is possible to integrate symbolic capabilities such as NL question answering and instruction following using the moral equivalent of a behavior-based system. In the next few months, we plan to extend the system considerably. We hope to give a live demo at the symposium.

References

Agre, P. 1997. *Computation and Human Experience*. Cambridge University Press, Cambridge.

Agre, P. and Chapman, D. 1987. "Pengi: An Implementation of a Theory of Activity," in *Proceedings of the Sixth National Conference on Artificial Intelligence*, 268-272. Seattle, WA.

Arkin, A. 1998. *Behavior-Based Robotics*. MIT Press, Cambridge, MA.

Bonasso, P., Firby, R. J., Gat, E., and Kortenkamp, D. 1997. "Experiences with an Architecture for Intelligent Reactive Agents." In *Journal of Theoretical and Experimental Artificial Intelligence* special issue on software architectures for physical agents, Hexmoor, Horswill, and Kortenkamp, eds., 9:2-3. Taylor and Francis, Ltd.

Brooks, R. 1986. "A Robust Layered Control System for a Mobile Robot," *IEEE Journal of Robotics and Automation*, Vol. RA-2, No. 1, pp. 14-23.

Brooks, R. 1990. "The Behavior Language," *A.I. Memo No. 1227*, MIT AI Laboratory, April.

Fikes, R, Hart, P., and Nilsson, N. 1972. "Learning and Executing Generalized Robot Plans," *Artificial Intelligence* Vol. 3, No. 4, pp. 251-288.

Horswill, I. 1993. "Polly: A Vision-Based Artificial Creature." In *Proceedings of the Eleventh National Conference on Artificial Intelligence*. pp. 824-829. AAAI Press.

Horswill, I. 1998. "Grounding Mundane Inference in Perception," *Autonomous Robots*, Vol. 5, pp. 63-77.

Horswill, I. 2000. "Functional Programming of Behavior-Based Systems." *Autonomous Robots.* In press.

Kaelbling, L. 1987. "Rex: A Symbolic Language for the Design and Parallel Implementation of Embedded Systems," *Proceedings of the AIAA Conference on Computers in Aeorospace VI*, Wakefield, MA, pp. 255-60.

Kaelbling, L. and Rosenschein, S. 1991. "Action and Planning in Embedded Agents," in *Designing Autonomous Agents*, ed. P. Maes, MIT Press, Cambridge, MA, pp. 35-48.

Maes, P. 1989. "The Dynamics of Action Selection," *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, MI, pp. 991-997.

Maes, P. 1990. "Situated Agents Can Have Goals," *Robotics and Autonomous Systems*, Vol. 6, pp. 49-70.

Minsky, M. 1984. *The Society of Mind*. Simon and Schuster, New York.

Nilsson, N. 1994. "Teleo-Reactive Programs for Agent Control," *Journal of Artificial Intelligence Research*, Vol. 1, pp. 139-158.

Schaad, R. 1998. *Representation and Execution of Situated Action Sequences.* Dissertation Der Wirtschaftswissenschaftlichen, Universität Zürich.