Computational Linguistics
From Zen to Aum
Computational Linguistics
A few specific applicative techniques:

The Zen toolkit - Generic technology
We represent finite-state automata by a mixed structure - an automata Mista-AuM.

The first component is a forest of lexical trees, used as covering trees of the state transitions graph. The rest of the transitions is represented as annotations stating that on a certain input (a word possibly empty, allowing ε-transitions), the automaton goes to a state designated by a virtual address. There are two kinds of addresses, local and global. A local address has the same structure, but now acts as a differential word. A global address is given by an integer (indexing into the forest array), and a word. A local address is indexed by an integer (indexing into the forest array) and a word. Necessary because of sharing.

The first component is a deterministic skeleton decorated by non-deterministic transitions.
A differential word is a notation permitting to retrieve a word from and another word \( w' \) sharing a common prefix. It denotes the minimal path connecting the words in a tree, as a sequence of ups and downs.

A differential word is a notation permitting to retrieve a word from another word \( w \).

\[ \text{type: delta \rightarrow word \rightarrow word} = \text{patch \_ \_ word} \]
The automaton structure:

- Input: `word`
- Delta: `int * word`
- Global address: `delta`
- Local address: `delta`

Type `auto`:
- State: `bool * deter * choices`
- Deterministic: `list letter * auto`
- Choice: `list input * address`

Type `automaton`:
- Array of `auto` * `delta`

Type `backtrack`:
- Input: `input`
- Delta: `delta`
- Choice: `choices`
- Resumption: `list backtrack`

Type `resumption`:
- Coroutine: `list backtrack`

The automaton structure:
Completeness

Every non-deterministic automaton (possibly with $\epsilon$ transitions) may be represented as a flat automaton (possibly with $\epsilon$ transitions) whose choice annotations do not give rise to backtrack. N.B. Sharing the local virtual addresses does not necessarily correspond by equivalence by bisimulation. Sharing the local virtual addresses does not necessarily correspond by equivalence by bisimulation. Sharing the local virtual addresses does not necessarily correspond by equivalence by bisimulation.

Complementness
The transducer structure

(\* coroutine resumption \* (list backtrack)
and resumption = list backtrack
(type backtrack = (input * output * delta * choices)

(type transducer = (array trans * delta)

(type backtrack = (input * output * delta * choices)
and choices = list (input * output * address)

(type deter = list (letter * trans)
and deter = list deter
)

(type trans = list deter
[ State of (bool * deter * choices)

(type address = (global of delta | local of delta)

(type delta = (int * word)

(type input = word and output = word)

The transducer structure
- 6 -

dérégement

mot

an

at

dér rég ém ent

AUM

K
The access stack \([sn; sn−1; ...; s0]\) is necessary to interpret local virtual addresses. It may be convenient to store as well the current access stack, stacked and unstacked along the local addresses. The access stack \([s0; s1; ...; sn]\) is necessary to interpret local virtual addresses.

Applications:

-Inflected forms dictionary used as lemmatizer (regular plural):
-Absolute of word

Absolute of word of relative of det: In the last case, output is computed by patch applied to word.

Applications. We may thus distinguish two output constructors:

- \((([s_i], 1) = q)\) •
- \((([[]], 0) = q)\) •
- \(((n, 0) = q)\) •

Memorisation of the current access
Modular aums are given by a pair in (array auto * delta), where we put $a + q + 1$, where we put

$B + A$ by a continuation by relocating by $B + A$ is obtained by relocating $(arrayA, deltaA)$ starting at $B$ is of size $a$ and $(arrayB, deltaB)$ is of size $q$. If $A = B$ •

$A + deltaB$, starting at deltaA, of size $a + q - 1$.

$A \cdot B$ is obtained by relocating $B$ by $a$, continuing $A$ by $a + deltaB$, starting at deltaA, of size $a + q$.

The base case is any aum, its size the size of its array.

Now it is easy to compile regular expressions into aums, as follows:

- With empty input.
- Continuations are implemented as e-transitions, i.e., extra choices.
- Extra transitions and possibly interpreting success states by continuations.
- We make them modular by making the global addresses relocatable.

An aum is given by a pair in (array auto * delta).
These transformations ought to be effected before sharing.

If $A = (a, \delta A)$ is of size $a$, then $A^*$ is obtained by continuing $A$ by $\delta A$, making its starting node accepting, of size $a$.

State \( \Delta \) is of size $q + a + 1$. If $A^* \Delta = A \Delta$ then $A^*$ is obtained by

- If $A = (a, \delta A)$ is of size $a$ then $A^*$ is obtained by continuing $A$ by $\delta A$, making its starting node accepting, of size $a$. …
Conclusion

Automata offer an elegant applicative solution to many finite-state processing problems, typically the treatment of lexicon, phonology, morphology and segmentation in computational linguistics. The deterministic spanning tree of the state space is then naturally the dictionary of inflected forms of words, which is thus placed at the center of the computer treatment of language.