Implementation Concepts in Vaucanson 2

Akim Demaille[1], Alexandre Duret-Lutz[1], Sylvain Lombardy[2], Jacques Sakarovitch[3]

[1]EPITA/LRDE, [2]LaBRI, Université de Bordeaux, [3]LTCI, CNRS / Télécom-ParisTech
http://vaucanson.lrde.epita.fr
http://vaucanson-project.org

CIAA 2013; July, 17th 2013

(2013-03-08 12:22:26 +0000 94dae6f)

Vaucanson

Features and Objectives

General-purpose platform for weighted automata and transducers.

- Genericity
  - Acceptors and transducers
  - Rational expressions
  - Weights: $\mathbb{B}$, $\mathbb{Z}$, $\langle \mathbb{Z}, \min, + \rangle$, $\mathbb{Q}$, $\mathbb{R}$, Rational expressions...
  - Letters: `char`s, `int`s...
  - Labels: letter, word, ε...
- Performance
  - C++ templated library
  - No dynamic polymorphism (`virtual`)
  - Efficient algorithms and data structures
- Flexibility
  - A dynamic API on top of the templated library
  - A dynamic typing system for automata, rational expressions, etc.
  - TAF-Kit, a shell API

Features and Objectives

General-purpose platform for weighted automata and transducers.

- Genericity
  - Acceptors and transducers
  - Rational expressions
  - Weights: $\mathbb{B}$, $\mathbb{Z}$, $\langle \mathbb{Z}, \min, + \rangle$, $\mathbb{Q}$, $\mathbb{R}$, Rational expressions...
  - Letters: `char`s, `int`s...
  - Labels: letter, word, ε...
- Performance
  - C++ templated library
  - No dynamic polymorphism (`virtual`)
  - Efficient algorithms and data structures
- Flexibility
  - A dynamic API on top of the templated library
  - A dynamic typing system for automata, rational expressions, etc.
  - TAF-Kit, a shell API

Features and Objectives

General-purpose platform for weighted automata and transducers.

- Genericity
  - Acceptors and transducers
  - Rational expressions
  - Weights: $\mathbb{B}$, $\mathbb{Z}$, $\langle \mathbb{Z}, \min, + \rangle$, $\mathbb{Q}$, $\mathbb{R}$, Rational expressions...
  - Letters: `char`s, `int`s...
  - Labels: letter, word, ε...
- Performance
  - C++ templated library
  - No dynamic polymorphism (`virtual`)
  - Efficient algorithms and data structures
- Flexibility
  - A dynamic API on top of the templated library
  - A dynamic typing system for automata, rational expressions, etc.
  - TAF-Kit, a shell API

Features and Objectives

General-purpose platform for weighted automata and transducers.

- Genericity
  - Acceptors and transducers
  - Rational expressions
  - Weights: $\mathbb{B}$, $\mathbb{Z}$, $\langle \mathbb{Z}, \min, + \rangle$, $\mathbb{Q}$, $\mathbb{R}$, Rational expressions...
  - Letters: `char`s, `int`s...
  - Labels: letter, word, ε...
- Performance
  - C++ templated library
  - No dynamic polymorphism (`virtual`)
  - Efficient algorithms and data structures
- Flexibility
  - A dynamic API on top of the templated library
  - A dynamic typing system for automata, rational expressions, etc.
  - TAF-Kit, a shell API

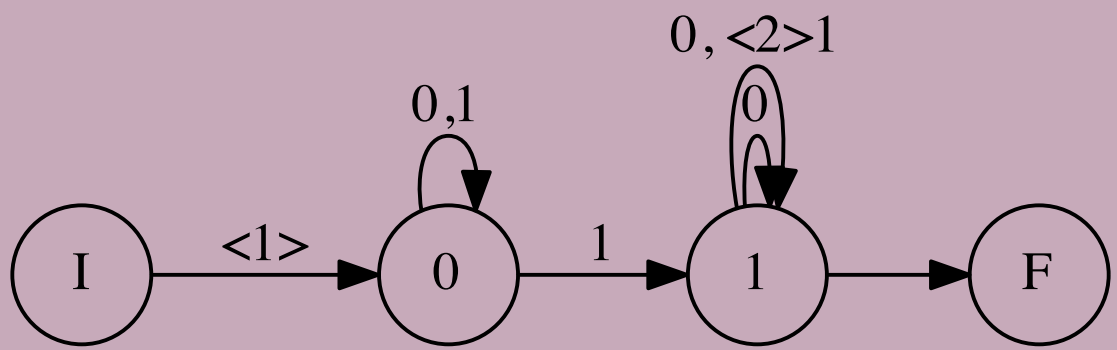
TAF-Kit

'binary.gv': (0+1)*1(<2>0+<2>1)*

```
digraph binary {

  I -> 0 [label = "<1>"]
  0 -> 0 [label = "0,1"]
  0 -> 1 [label = "1"]
  1 -> 1 [label = "0"]
  1 -> 1 [label = "0, <2>1"]
  1 -> F
}
```

```
$ dotty binary.gv
```

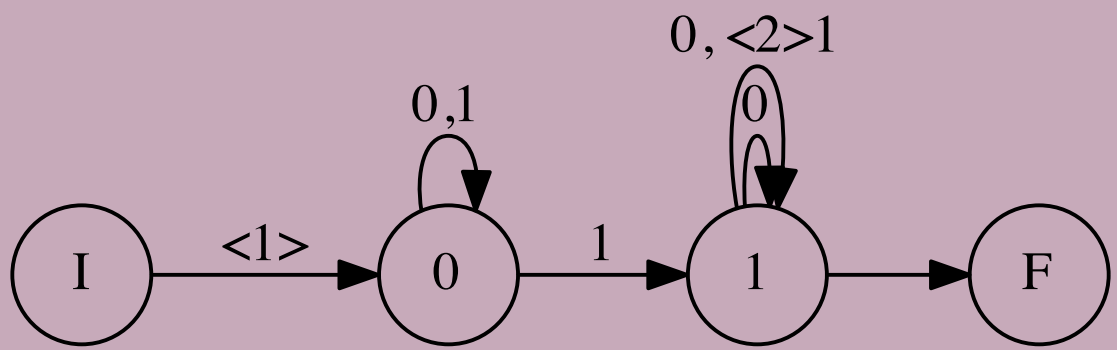


Vaucanson: Typed I/O of Automata in GraphViz

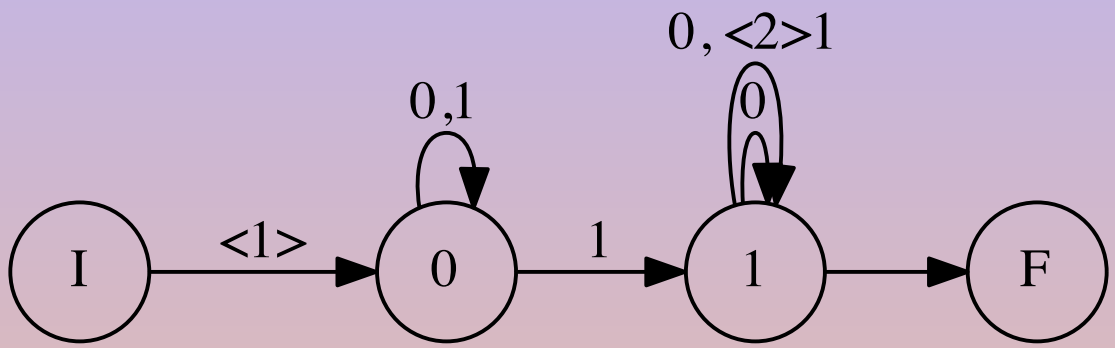
'binary.gv': (0+1)*1(<2>0+<2>1)*

```
digraph binary {
  vcsn_context = "lal_char(01)_z" // transition type: {0,1} → ℤ
  I -> 0 [label = "<1>"]
  0 -> 0 [label = "0,1"]
  0 -> 1 [label = "1"]
  1 -> 1 [label = "0"]
  1 -> 1 [label = "0, <2>1"]
  1 -> F
}
```

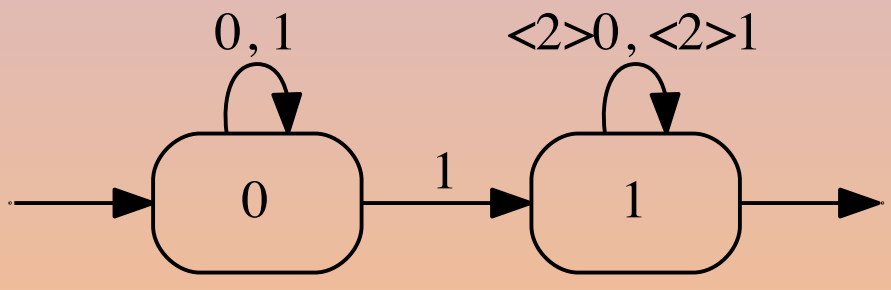
```
$ dotty binary.gv
```



Using TAF-Kit

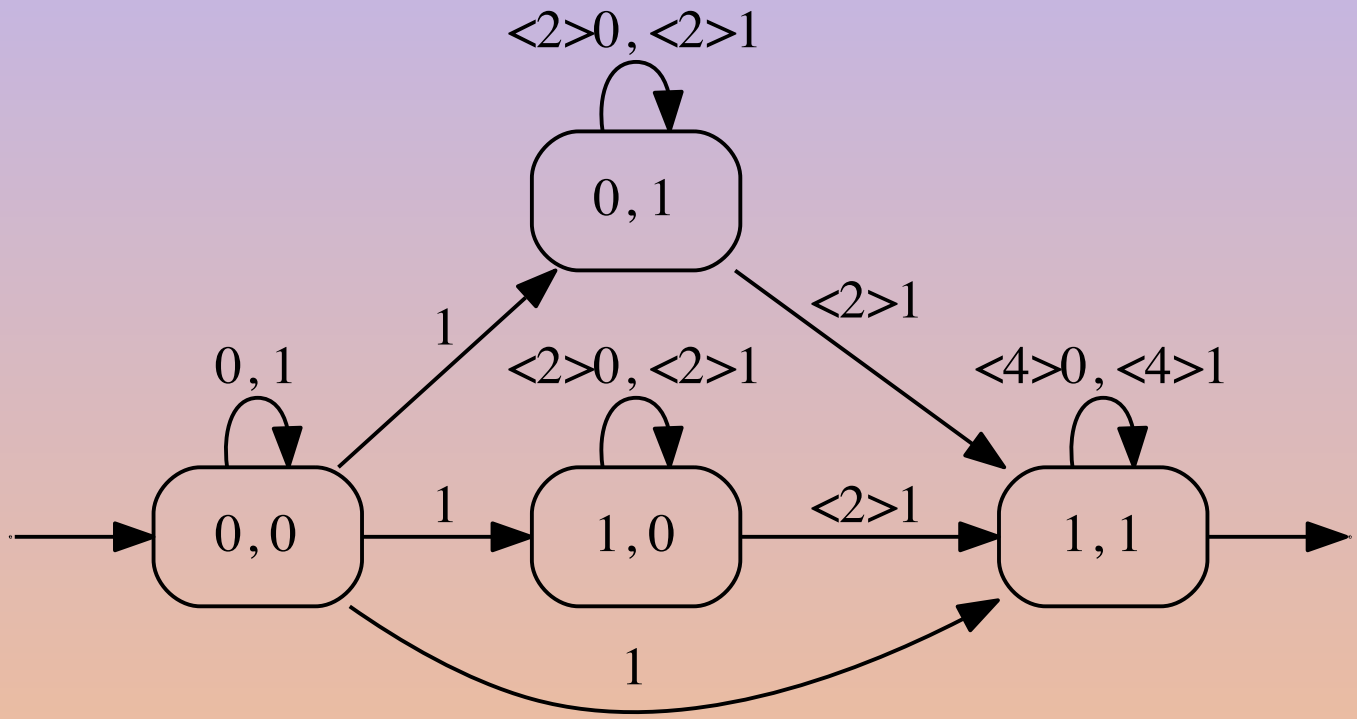



```
$ vcsn cat -f binary.gv -o binary-catted.gv
```



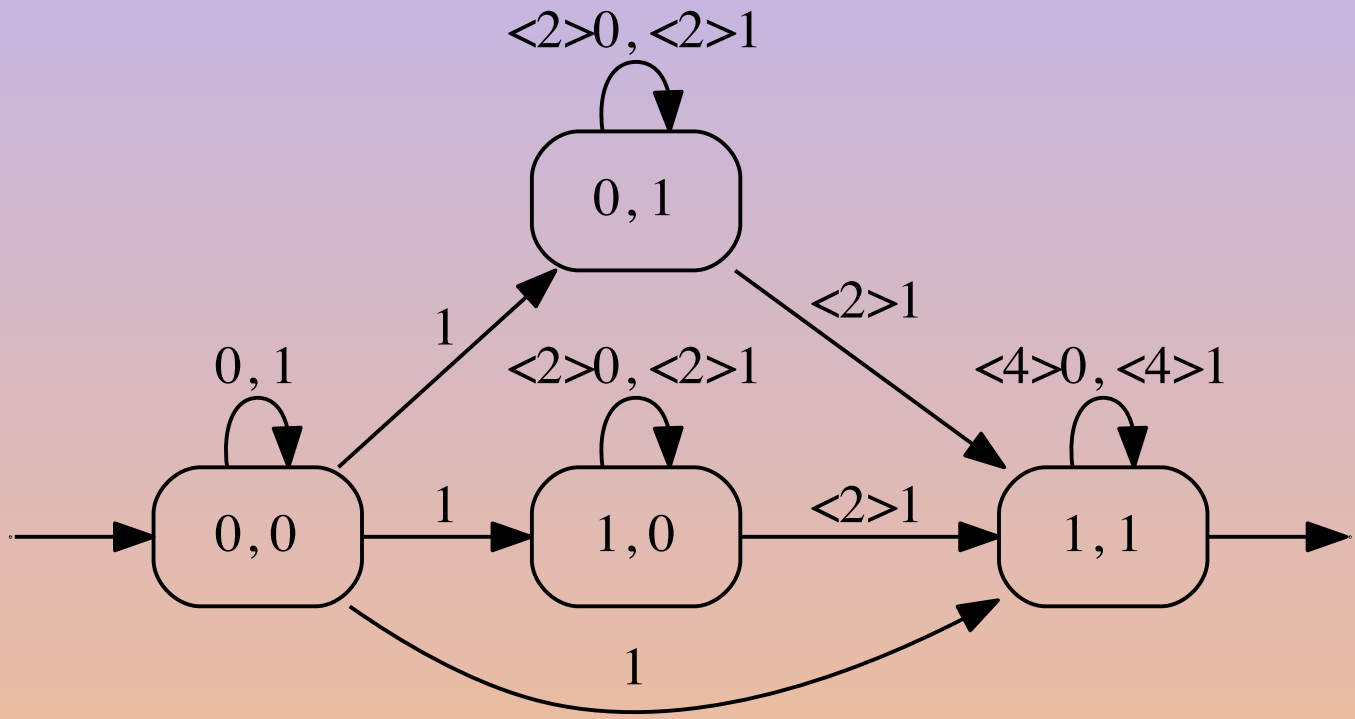
Using TAF-Kit

```
$ vcsn product -f binary.gv binary.gv -o squared-binary.gv
```



Using TAF-Kit

```
$ vcsn product -f binary.gv binary.gv -o squared-binary.gv
```

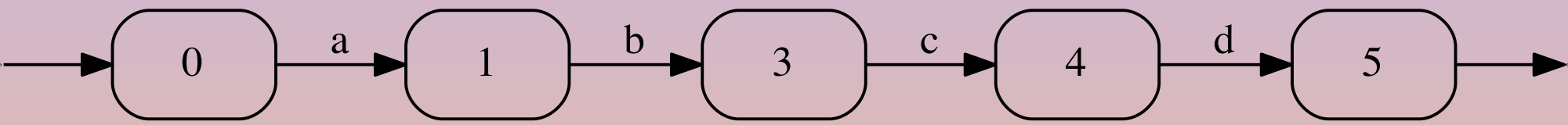



```
$ vcsn evaluate -f binary.gv 101
5
$ vcsn evaluate -f squared-binary.gv 101
25
```

Typing the Algebraic Context: Labels

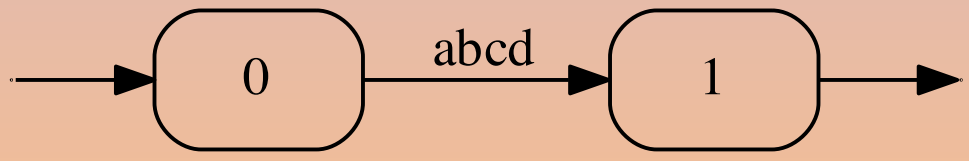
Labels Are Letters:

```
$ vcsn standard -C 'lal_char(abcd)_b' -e abcd
```



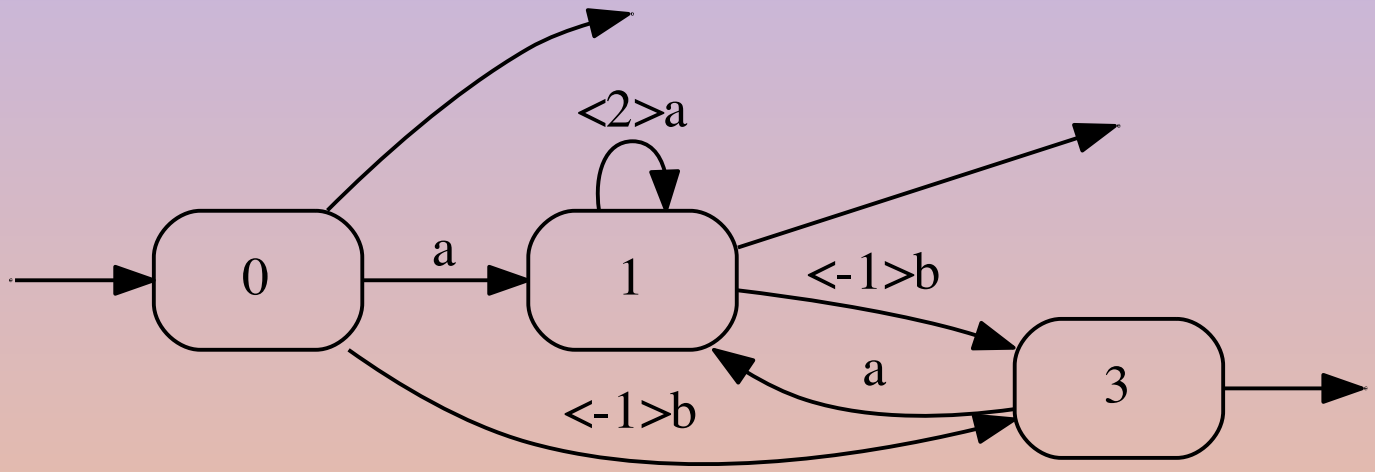
Labels Are Words:

```
$ vcsn standard -C 'law_char(abcd)_b' -e abcd
```



Typing the Algebraic Context: Weights

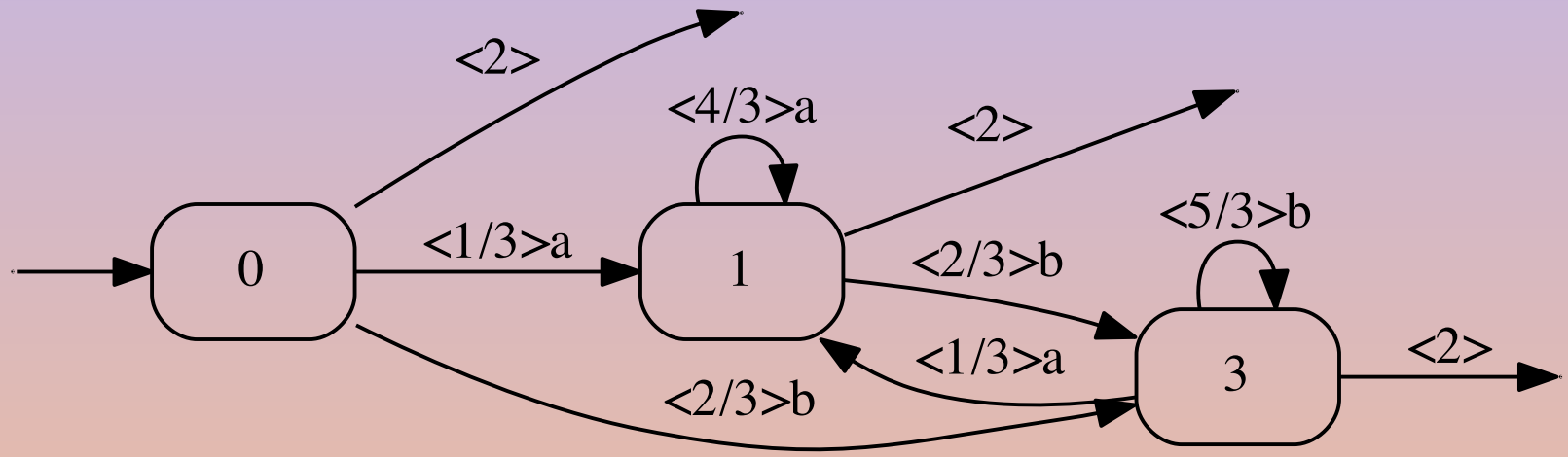
```
$ vcsn standard -C 'lal_char(ab)_z' -e '(a*+<-1>b*)*' -o lalz.gv
```




```
$ vcsn evaluate -f lalz.gv aba
-1
$ vcsn evaluate -f lalz.gv aabaabaa
8
```

Typing the Algebraic Context: Weights

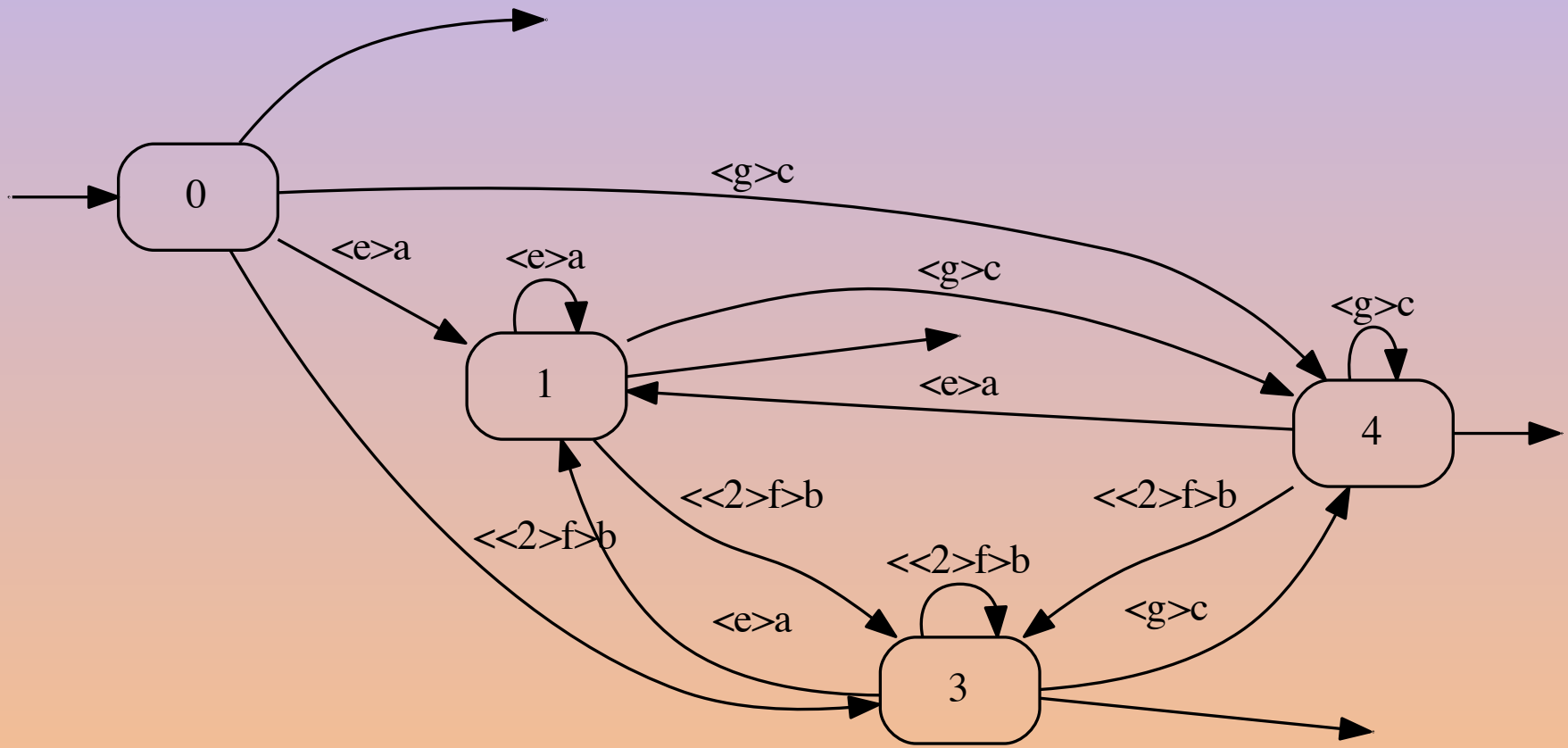
```
$ vcsn standard -C 'lal_char(ab)_q' -e '(<1/6>a*+<1/3>b*)*' -o lalq.gv
```




```
$ vcsn evaluate -f lalq.gv aba
4/27
$ vcsn evaluate -f lalq.gv aabaabaa
512/6561
```

Typing: $\{a, b, c, d\} \to \{f, g, h\} \to \mathbb{Z}$

```
$ vcsn standard -C 'lal_char(abcd)_ratexpset<lal_char(efg)_z>' \
                -e '(<e>a+<<2>f>b+<g>c)*' -o lallal.gv
```

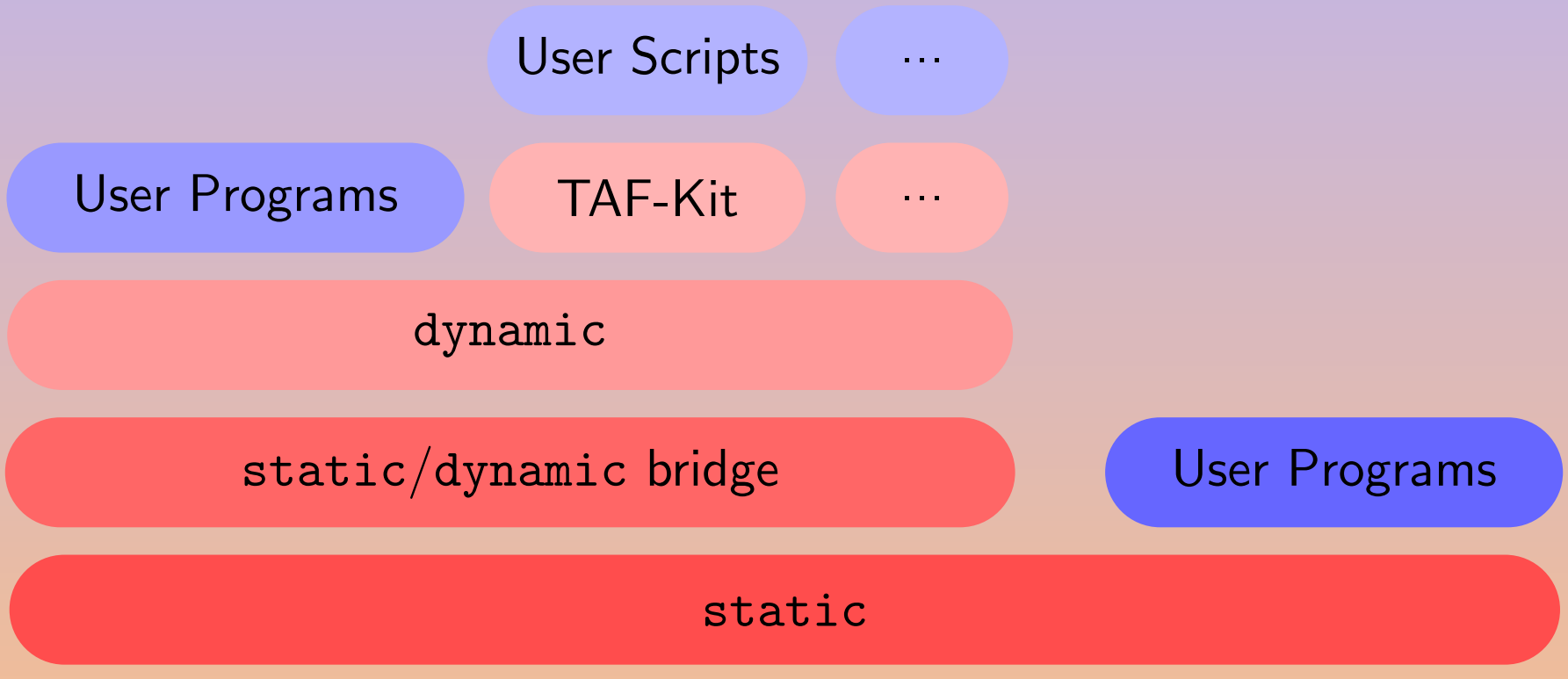



```
$ vcsn evaluate -f lallal.gv abbc
e.<2>f.<2>f.g
```

```
$ vcsn evaluate -f lallal.gv d
\z
```

Static/Dynamic Bridge

Birdview Architecture



The Static Layer: Type Safety and Efficiency

```
template <typename Ctx>
void
sta_prod_eval(const std::string& lhs, const std::string& rhs,
              const std::string& word)
{
  using automaton_t = vcsn::mutable_automaton<Ctx>;
  automaton_t l = vcsn::read_automaton_file<automaton_t>(lhs);
  automaton_t r = vcsn::read_automaton_file<automaton_t>(rhs);
  automaton_t prod = vcsn::product<automaton_t, automaton_t>(l, r);
  typename Ctx::weight_t w = vcsn::eval<automaton_t>(prod, word);
  prod.context().weightset()->print(std::cout, w);
}
```

```
sta_prod_eval<vcsn::ctx::lal_char_z>(argv[1], argv[2], argv[3]);
```

The Dynamic Layer: Flexibility

```
static void
dyn_prod_eval(const std::string& lhs, const std::string& rhs,
              const std::string& word)
{
  using namespace vcsn::dyn;
  automaton l = read_automaton_file(lhs);
  automaton r = read_automaton_file(rhs);
  automaton prod = product(l, r);
  weight w = eval(prod, word);
  print(w, std::cout, "text");
}
```

```
dyn_prod_eval(argv[1], argv[2], argv[3]);
```

The Dynamic Layer: Flexibility

The static version is written for `lal_char_z` only:

```
$ vcsn standard -C 'lal_char(ab)_b' -e 'a+b' -o 'a+b.gv'
$ vcsn standard -C 'lal_char(ab)_b' -e 'a*'  -o 'a*.gv'
$ sta-prod-eval 'a+b.gv' 'a*.gv' a
sta-prod-eval: a+b.gv: invalid type: mutable_automaton<lal_char_b>,
                       expected: mutable_automaton<lal_char_z>
```

The dynamic version accepts any type:

```
$ dyn-prod-eval 'a+b.gv' 'a*.gv' a
1
```

```
$ dyn-prod-eval 'a+b.gv' 'a*.gv' b
0
```

Or almost:

```
$ dyn-prod-eval 'binary.gv' 'a+b.gv' a
dyn-prod-eval: product: no implementation available for
          mutable_automaton<lal_char_z> x mutable_automaton<lal_char_b>
```

The Dynamic Layer: Flexibility

The static version is written for `lal_char_z` only:

```
$ vcsn standard -C 'lal_char(ab)_b' -e 'a+b' -o 'a+b.gv'
$ vcsn standard -C 'lal_char(ab)_b' -e 'a*'  -o 'a*.gv'
$ sta-prod-eval 'a+b.gv' 'a*.gv' a
sta-prod-eval: a+b.gv: invalid type: mutable_automaton<lal_char_b>,
                       expected: mutable_automaton<lal_char_z>
```

The dynamic version accepts any type:

```
$ dyn-prod-eval 'a+b.gv' 'a*.gv' a
1
```

```
$ dyn-prod-eval 'a+b.gv' 'a*.gv' b
0
```

Or almost:

```
$ dyn-prod-eval 'binary.gv' 'a+b.gv' a
dyn-prod-eval: product: no implementation available for
          mutable_automaton<lal_char_z> x mutable_automaton<lal_char_b>
```

The Dynamic Layer: Flexibility

The static version is written for `lal_char_z` only:

```
$ vcsn standard -C 'lal_char(ab)_b' -e 'a+b' -o 'a+b.gv'
$ vcsn standard -C 'lal_char(ab)_b' -e 'a*'  -o 'a*.gv'
$ sta-prod-eval 'a+b.gv' 'a*.gv' a
sta-prod-eval: a+b.gv: invalid type: mutable_automaton<lal_char_b>,
                       expected: mutable_automaton<lal_char_z>
```

The dynamic version accepts any type:

```
$ dyn-prod-eval 'a+b.gv' 'a*.gv' a
1
```

```
$ dyn-prod-eval 'a+b.gv' 'a*.gv' b
0
```

Or almost:

```
$ dyn-prod-eval 'binary.gv' 'a+b.gv' a
dyn-prod-eval: product: no implementation available for
          mutable_automaton<lal_char_z> x mutable_automaton<lal_char_b>
```

The Static/Dynamic Bridge

Eventually

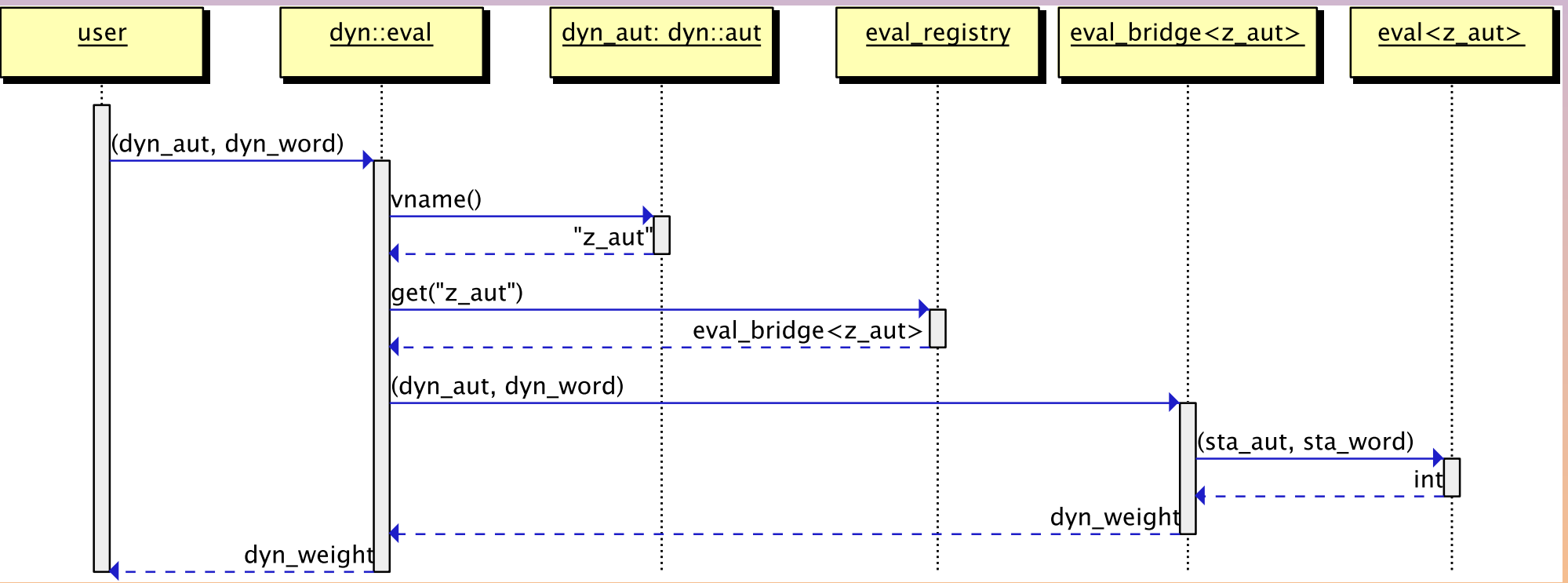
```
dyn::weight dyn_weight = dyn::eval(dyn_aut, dyn_word);
```

calls

```
auto sta_weight = vcsn::eval<automaton_t>(sta_aut, sta_word);
```

dyn::eval eventually calls eval<z_aut>

```
dyn::automaton dyn_aut = {...};
dyn::word      dyn_word = "abcd";
dyn::weight    dyn_weight = dyn::eval(dyn_aut, dyn_word);
```



The Static/Dynamic Bridge

```
dyn::weight dyn_weight = dyn::eval(dyn_aut, dyn_word);
```

```
namespace dyn
{
  template <typename Automaton>
  weight
  eval(const automaton& dyn_aut, const word& dyn_word)
  {
    const auto& sta_aut  = dyn_aut->as<const Automaton&>();
    const auto& sta_word = dyn_word->as<const Automaton::word_t&>();

    auto sta_res = vcsn::eval<Automaton>(sta_aut, sta_word);

    const auto& dyn_ctx = sta_aut.context();
    return make_weight(dyn_ctx.weightset(), sta_res);
  }
}
```

```
auto sta_weight = vcsn::eval<automaton_t>(sta_aut, sta_word);
```

Conclusion

Static: Type Safety and Efficiency

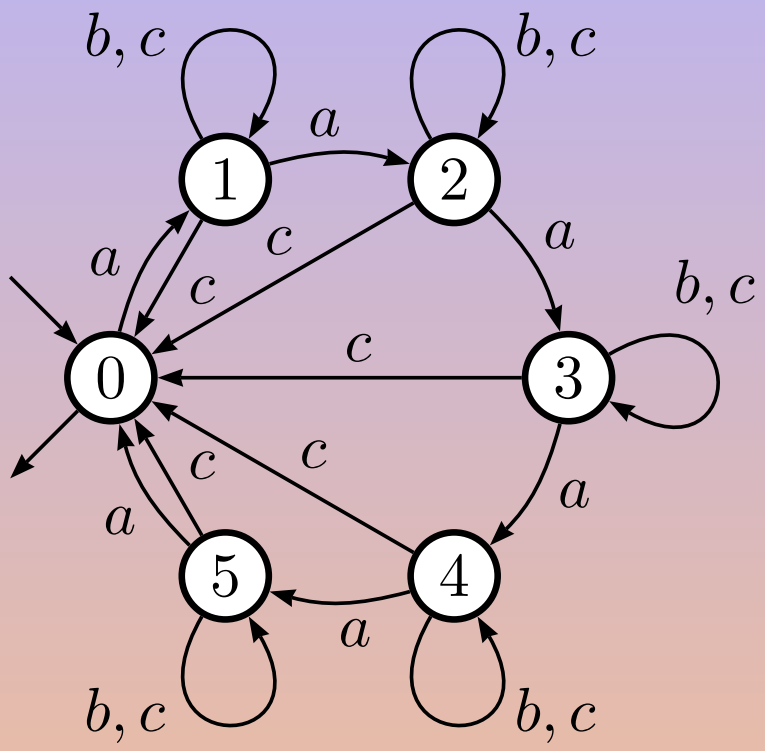
ε-removal on Thompson$((\varepsilon + a)^n)$

Duration (s)	100	200	500	1000
Vaucanson 1	0.55	4.27	84.49	783.98
Vaucanson 2	0.06	0.19	3.58	53.45
OpenFst	0.05	0.10	0.47	1.79

Grail 3.4.1
Vaucanson 1.4.1
Vaucanson 2 July 2013
OpenFst 1.3.3
MacBook Pro, i7 2.9GHz, 8GB RAM
GCC 4.8 -O3

Static: Type Safety and Efficiency

Ladybird Determinization



	11	13	15	17	19	21
Grail	0.03	0.46	6.21	140		
Vaucanson 1	0.10	0.44	1.95	8.93	40.17	183.46
Vaucanson 2	0.05	0.19	0.85	3.79	16.82	73.84
OpenFst	0.04	0.07	0.27	1.12	4.82	20.24

Conclusion

State

- The direct heir of VAUCANSON 1
- Much simpler static API
- Faster too
- Brand new dynamic API

Transition

- Transducers
- Runtime compilation
- Python
- ...

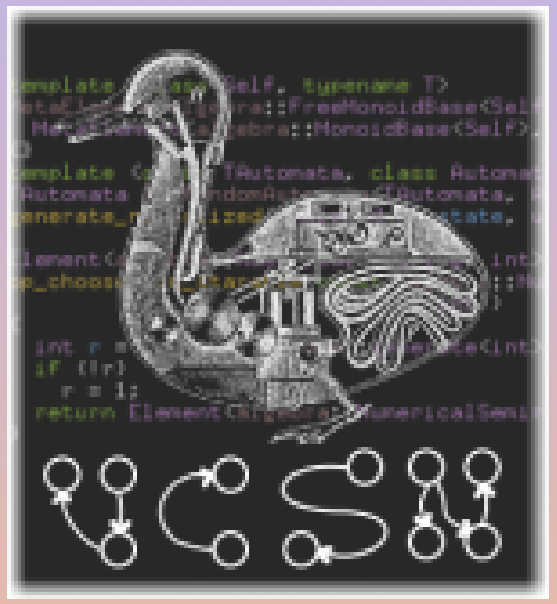
Conclusion

State
- The direct heir of VAUCANSON 1
- Much simpler static API
- Faster too
- Brand new dynamic API

Transition
- Transducers
- Runtime compilation
- Python
- ...

Questions?

1. Vaucanson
2. TAF-Kit
3. Static/Dynamic Bridge
4. Conclusion



http://vaucanson.lrde.epita.fr
http://vaucanson-project.org