Erlang Solutions Ltd.

Log Analysis With Exago

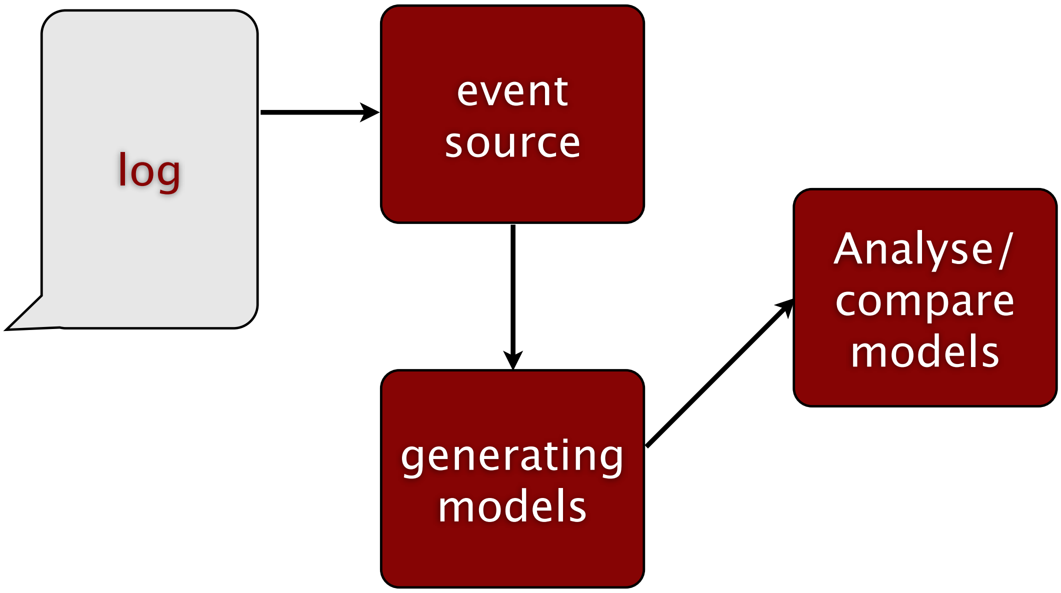
Get Exago

- https://github.com/et4te/**ExagoE**
- You might want to install GraphViz to generate schemas

Logs: they are important

- The best test is sending things in production
  - Nothing represents production conditions like production itself
- Logs are often the only way to know what happens in production
- Logs can be problematic
  - What format to keep them in
  - how to interpret them
  - How to find the source of problems

How Exago Works



log

event source

generating models

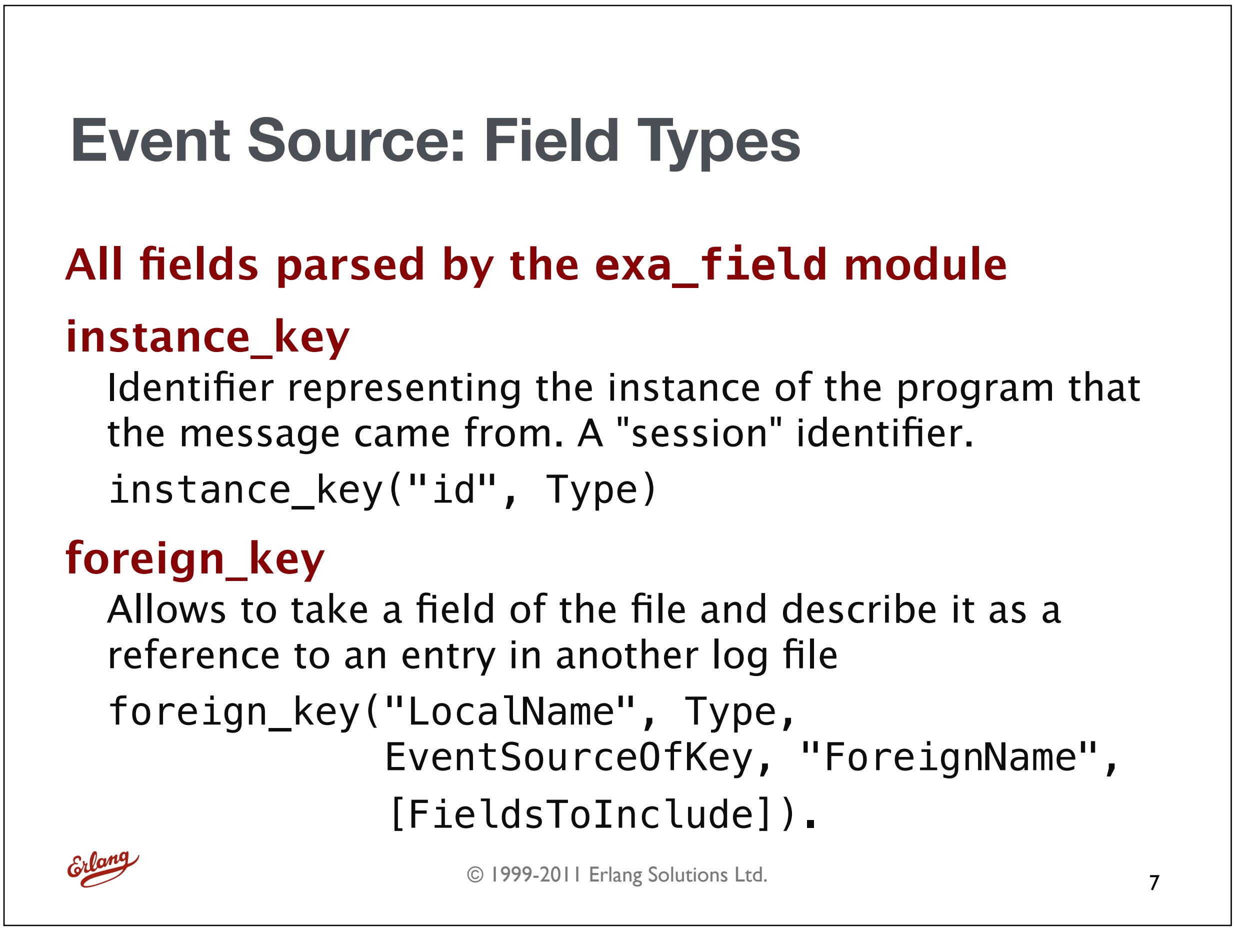
Analyse/ compare models

Logs

- Incredible variety in the wild
- Text: ASCII, Unicode, Latin-1, Constants, etc.
- Separators: white space, -, #, byte length, etc.
- Fields: ID, IP, Host, names, domains, time
- Time Stamps
- Purpose:
  - human readability
  - write speed, space efficiency
  - indexing, search.

Event Source

- Event sourcing is about telling Exago how to parse logs (text-based) and read them
- Exago accepts lists of lists as a final format
  - Format is Lines = [Fields=[A,B,...]]
- For now, Exago only provides basic CSV as a format supported out of the box, but it is possible to add more.
- Exago also provides basic data types for id's, foreign keys, etc.



Event Source: Field Types

All fields parsed by the exa_field module

instance_key

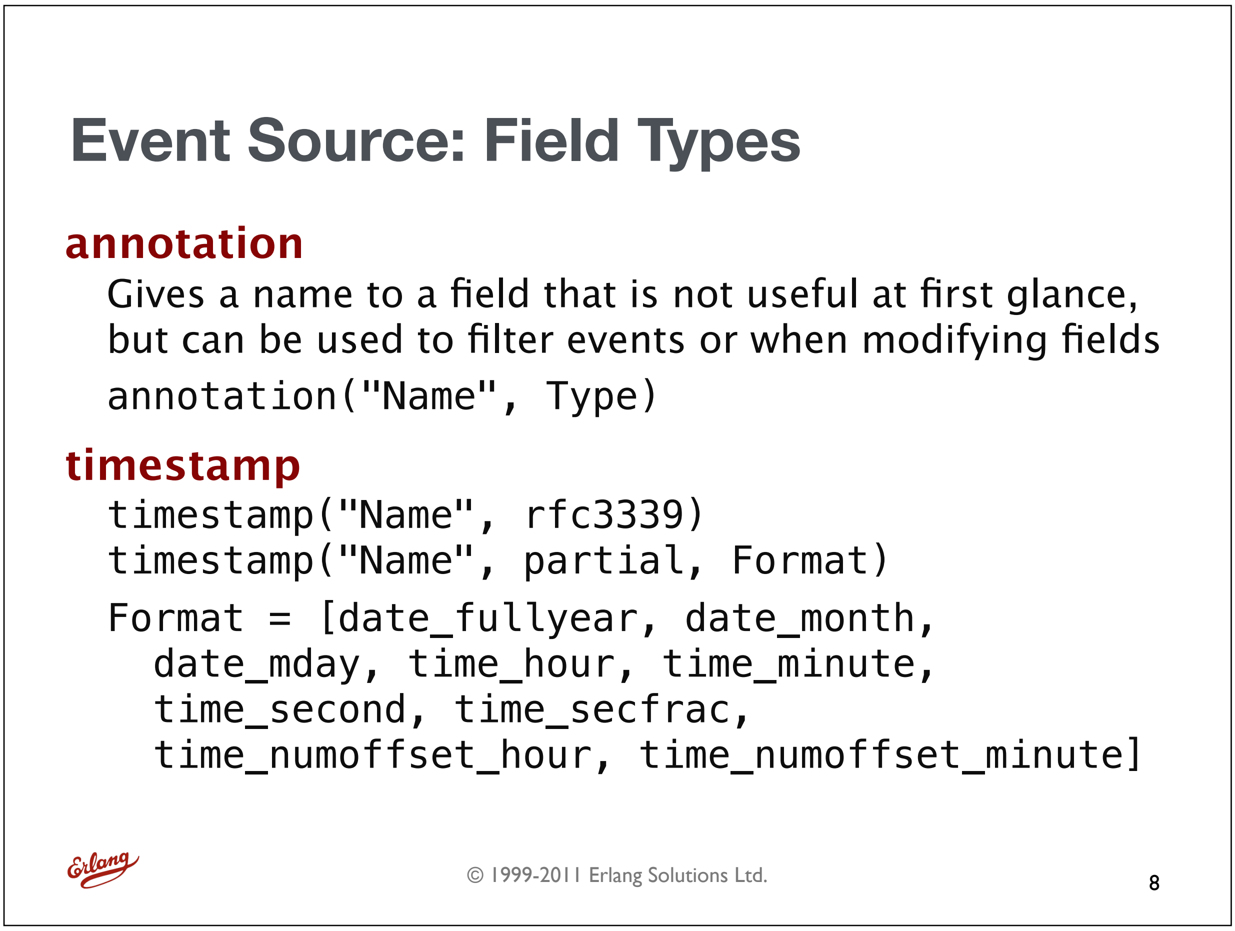
Identifier representing the instance of the program that the message came from. A "session" identifier.

```
instance_key("id", Type)
```

foreign_key

Allows to take a field of the file and describe it as a reference to an entry in another log file

```
foreign_key("LocalName", Type,
            EventSourceOfKey, "ForeignName",
            [FieldsToInclude]).
```



Event Source: Field Types

annotation

Gives a name to a field that is not useful at first glance, but can be used to filter events or when modifying fields

```
annotation("Name", Type)
```

timestamp

```
timestamp("Name", rfc3339)
timestamp("Name", partial, Format)
```

```
Format = [date_fullyear, date_month,
  date_mday, time_hour, time_minute,
  time_second, time_secfrac,
  time_numoffset_hour, time_numoffset_minute]
```

Event Source: Field Types

transition

An event that makes a program move to one state to the other.

Setting transitions allows Exago to transform the logs into a finite-state machine.

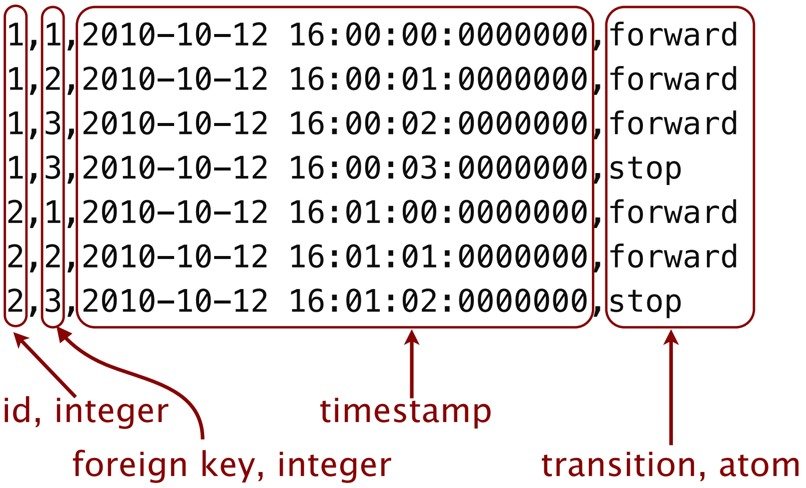
```
transition("EventName", Type)
```

state

Define a state name to be used in a finite state machine defining the current status.

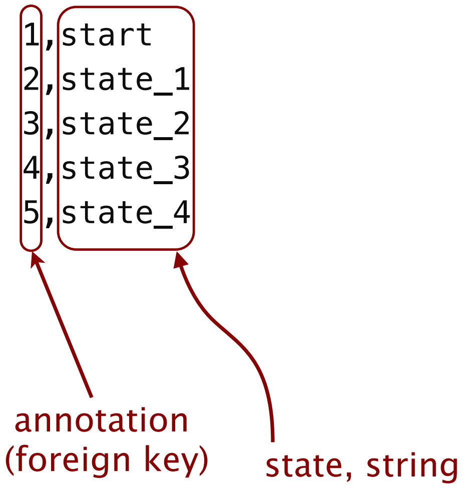
```
state(Name, Type)
```

Event Source: Row Formats



```
1,1,2010-10-12 16:00:00:0000000,forward
1,2,2010-10-12 16:00:01:0000000,forward
1,3,2010-10-12 16:00:02:0000000,forward
1,3,2010-10-12 16:00:03:0000000,stop
2,1,2010-10-12 16:01:00:0000000,forward
2,2,2010-10-12 16:01:01:0000000,forward
2,3,2010-10-12 16:01:02:0000000,stop
```

Event Source: Row Formats



```
1,start
2,state_1
3,state_2
4,state_3
5,state_4
```

annotation
(foreign key)

state, string

Event Source: Row Formats

```
event_source_1() ->
    {"sample_1",
     [{csv, absolute, "./log_files/sample_1.log"}],
     row_format_1()}.
event_source_2() ->
    {"sample_2",
     [{csv, absolute, "./log_files/sample_2.log"}],
     row_format_2()}.
```

- Standard way to define event sources is to:
- Give them a name
- define how to open the file(s)
  - **absolute** for precise filenames, **wildcard** to match on multiple file names
- A row format specification

Event Source: Row Formats

```
row_format_1() ->
  [exa_field:instance_key("id", integer),
   exa_field:foreign_key("foreignKey",integer, "sample_2",
                         "linkKey", ["state"]),
   exa_field:timestamp("timestamp", partial,
                       [date_fullyear, date_month,
                        date_mday, time_hour, time_minute,
                        time_second, time_secfrac]),
   exa_field:transition("move", atom)].

row_format_2() ->
    [exa_field:annotation("linkKey", integer),
     exa_field:state("state", string)].
```

Event Source: Combining Sources

```
combined_event_source() ->
{"sample_combined",
 exa_es:collect([event_source_1(), event_source_2()],
                absorb, implicit_state)}.
```

- An event source is of the form {"NameOfSource", Source}
- The source itself is given by `exa_es:collect([ListOfSources], absorb|append, implicit_state|source_state).`
  - **absorb**: used for logs with external keys; like a table join between files.
  - **append**: used for logs of the same format; useful for things like log rotation.
  - **implicit_state**: the state is defined in the log file
  - **source_state**: the event source's name acts as the state

Event Source: Result

```
{"sample_combined",
 [{complete_result,
  [{instance_key,{field_identifier,"id"},{field_value,1}},
   {foreign_key,field_identifier,"foreignKey"},
   {field_value,{foreign_reference,
                 {"sample_2",1,"linkKey".["state"]}}}},
                 {state,{field_identifier,"state"},
   {field_value,"start"}},
   {timestamp,{field_identifier,"timestamp"},
              {field_value,[{2010,date_fullyear}, ...}]},
   {transition,{field_identifier,"move"},
               {field_value,forward}}],
   field_format_eq},
   ...
```

Event Source

- Only a few fields are mandatory: instance_key, timestamp, transition. State can be derived, but it is useful to specify it.
- The Event Source allows to build an intermediary format that Exago can understand to base its analysis on.
- Exago can absorb event sources and generate finite-state machine models based on them.
- Exago can generate graphical representations of the models (using graphviz)

Generating Basic State Machines

- Using the event source, Exago can generate two kinds of state machines:
  - uniques: each sequence of logs (based on instance keys) creates one state machine
  - combined: all sequences of logs are combined into one large state machine.
- Exago can return abstract state machines defined in Erlang terms or a graphical state machine representations

Generating Basic State Machines

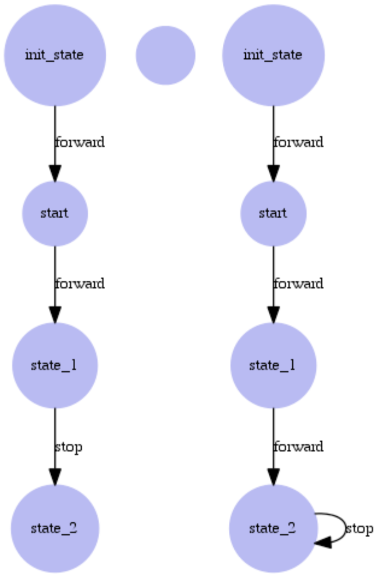
```
2> exa:generate_combined(Source, []).
{[{1,init_state},{2,start},{3,state_1},{4,state_2}],
 [{1,forward,2}, {2,forward,3}, {3,forward,4}, {3,stop,4},
  {4,stop,4}],
 {autogen_possible,true}}

3> exa:generate_uniques(Source, []).
[{[{1,init_state},{2,start},{3,state_1},{4,state_2}],
  [{1,forward,2},{2,forward,3},{3,forward,4},{4,stop,4}],
  {autogen_possible,true}},
 {[{4,state_2},{1,init_state},{2,start},{3,state_1}],
  [{1,forward,2},{2,forward,3},{3,stop,4}],
  {autogen_possible,true}}]
```

Generating Basic State Machines

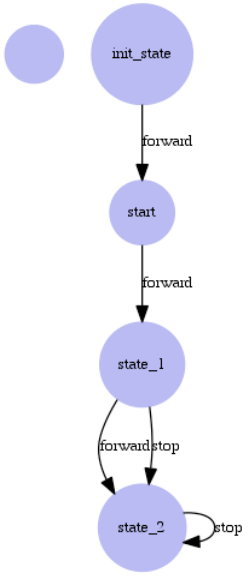
```
3> exa:generate_uniques(Source,
[{visualize, {true, "."}}]).
[{[[{1,init_state},...,
  [{1,forward,2},{2,forward,3},
   {3,forward,4},{4,stop,4}],
  {autogen_possible,true}},

 {[[{4,state_2},
   {1,init_state},
   {2,start},
   {3,state_1}],
  [{1,forward,2},{2,forward,3},
   {3,stop,4}],
  {autogen_possible,true}}]
```



Generating Basic State Machines

```
3> exa:generate_combined(Source,
[{visualize, {true, "."}}]).
{[[{1,init_state},{2,start},
  {3,state_1},{4,state_2}],
 [{1,forward,2},
  {2,forward,3},
  {3,forward,4},
  {3,stop,4},
  {4,stop,4}],
 {autogen_possible,true}}
```



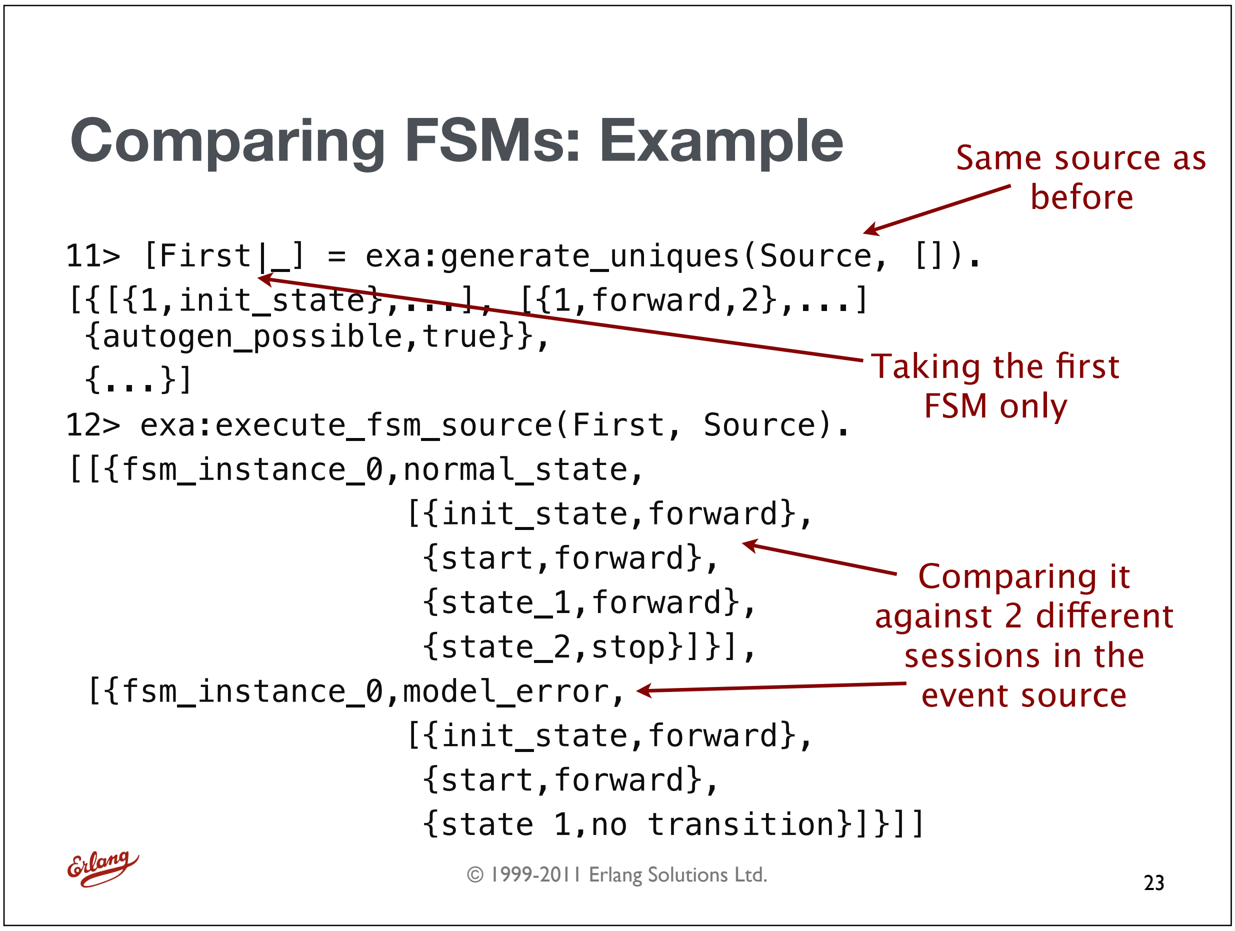
Writing State Machines by Hand

- Writing state machines by hand can be done by re-using the format output when generating them automatically

-
```
{[{StateId, StateName},...],
 [{InState, Event, ChangeToState}],
 {autogen_possible, true}}.
```

Running FSMs against event sources

- Finite State Machines correspond to a model of the behaviour the application should take.
- An expected state machine can be ran against an actual event source to compare its behaviour to what was expected
  - exa:execute(FSM, EventSource).
- A given state machine can be ran against another state machine to verify whether the first one is a subset of the second one
  - exa:execute(FSM1, FSM2).



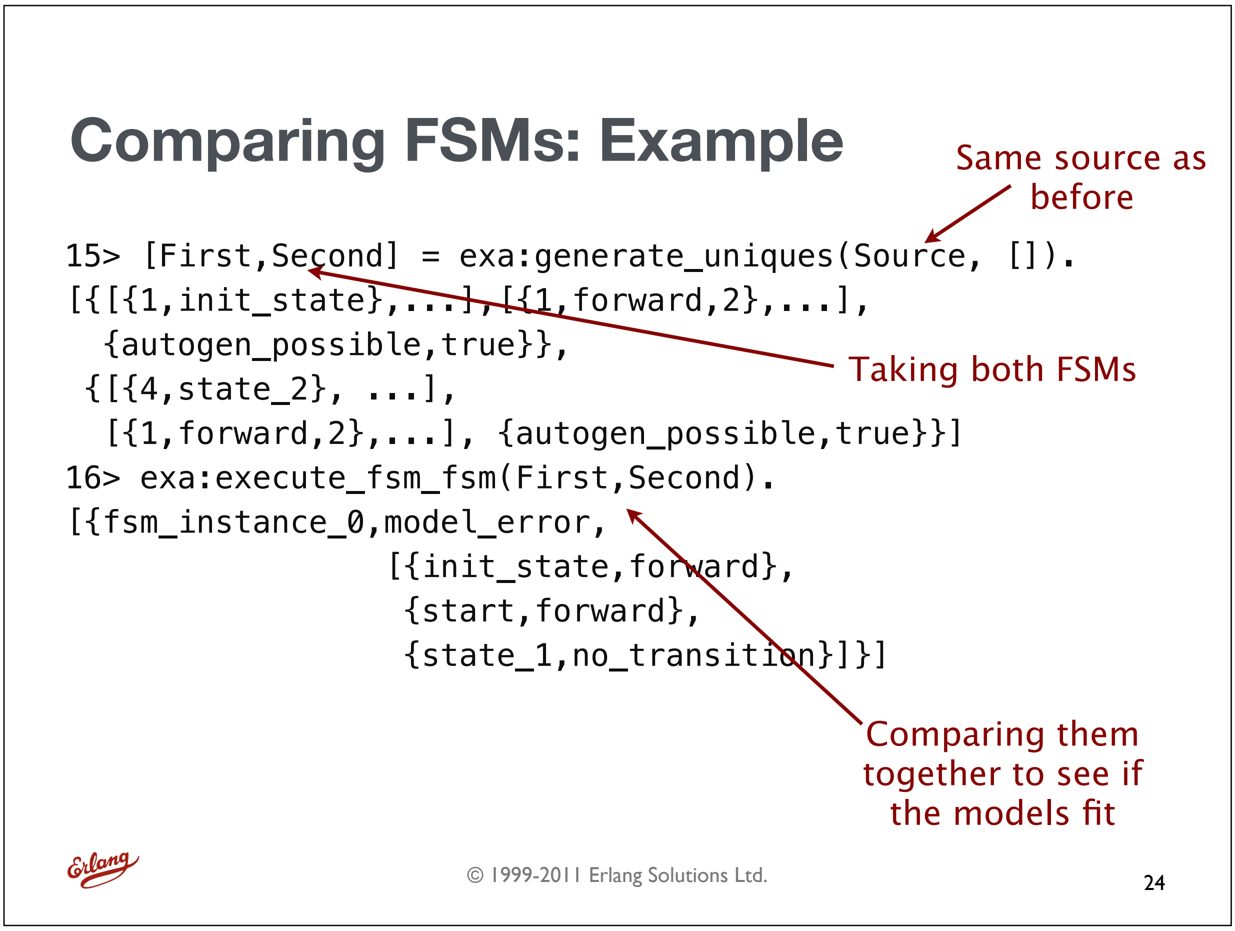
Comparing FSMs: Example

Same source as before

```
11> [First|_] = exa:generate_uniques(Source, []).
[{[{1,init_state},...], [{1,forward,2},...]
 {autogen_possible,true}},
 {...}]
12> exa:execute_fsm_source(First, Source).
[[{fsm_instance_0,normal_state,
                  [{init_state,forward},
                   {start,forward},
                   {state_1,forward},
                   {state_2,stop}]}],
 [{fsm_instance_0,model_error,
                  [{init_state,forward},
                   {start,forward},
                   {state 1,no transition}]}]]
```

Taking the first FSM only

Comparing it against 2 different sessions in the event source



Comparing FSMs: Example

Same source as before

```
15> [First,Second] = exa:generate_uniques(Source, []).
[{[{1,init_state},...],[{1,forward,2},...],
  {autogen_possible,true}},
 {[{4,state_2}, ...],
  [{1,forward,2},...], {autogen_possible,true}}]
16> exa:execute_fsm_fsm(First,Second).
[{fsm_instance_0,model_error,
                 [{init_state,forward},
                  {start,forward},
                  {state_1,no_transition}]}]
```

Taking both FSMs

Comparing them together to see if the models fit

Comparing FSMs: Example

Same source as before

```
18> General = exa:generate_combined(Source, []).
{[[{1,init_state}, ...],
 [{1,forward,2}, ...],
 {autogen_possible,true}}
19> exa:execute_fsm_fsm(First, General).
[{fsm_instance_0,normal_state,
                  [{init_state,forward},
                   {start,forward},
                   {state_1,forward},
                   {state_2,stop},
                   {state_2,stop}]}]
```

Using a combined FSM

Comparing one of the unique FSM to the general one

Refining FSMs

```
state_format() ->
    [exa_state:state(start, error),
     exa_state:state(state_1, normal),
     exa_state:state(state_2, accept)].
```

- Not all states of a finite-state machine are born equal
- The current FSMs are making no distinction between what is a good state or a bad state to finish in.
- We must augment the FSMs to be able to give meaning to such states.
- The meaning can be error, normal or accept.

Annotating FSMs: Examples

```
33> exa:execute_fsm_source(
33>    exa_sm:augment_model(
33>       First,
33>       [exa_state:state(state_2, error)]),
33>    Source).
[[{fsm_instance_0,error_state,
                  [{init_state,forward},
                   {start,forward},
                   {state_1,forward},
                   {state_2,stop}]}],
 [{fsm_instance_0,model_error,
                  [{init_state,forward},
                   {start,forward},
                   {state_1,no_transition}]}]]
```

Transition Modifiers

- Outside of the scope of the tutorial
- They allow to modify the event source during different stages of parsing
- They let you combine many fields or modify them to give a new event source
- Allows for smarter control of the event source

Transition Modifier: What's possible

```
2010-10-12 16:50:00:0821546,1286898600821546,close,1
2010-10-12 16:50:00:0821866,1286898600821866,move,1,up
2010-10-12 16:50:01:0822515,1286898601822515,approaching,1,2
2010-10-12 16:50:02:0214074,1286898602214074,close,2
2010-10-12 16:50:02:0214403,1286898602214403,move,2,up
```

Can give states such as

```
close_elevator_doors (first elevator)
move_elevator_up (first elevator)
approaching_floor_2 (first elevator)
close_elevator_doors (second elevator)
move_elevator_up (2nd elevator)
etc.
```

Exercises

```
{[{4,logged},{0,''},{1,locked},{2,unlogged},
  {3,init_state}],
 [{1,lock,1}, {1,unlocked,2}, {2,admin_locked,1},
  {2,denied,2}, {2,lock,1}, {2,logged,4}, {3,start,2}],
 {autogen_possible,true}}
```

- Get the logs at https://gist.github.com/2c477f8d837bb784cf87
- parse the logs into a FSM that is compatible with Exago
- Is the FSM deterministic? Can it be used to automatically run models?

```
UserName, TimeStamp, State, Event

Dwigth,2011-10-31 22:24:56:0950918,unlogged,start
Carl,2011-10-31 22:24:56:0952629,unlogged,start
Mike,2011-10-31 22:24:56:0954020,unlogged,start
Carl,2011-10-31 22:24:56:0954945,locked,admin_locked
Mike,2011-10-31 22:24:57:0052107,unlogged,denied
Mike,2011-10-31 22:24:57:0153185,unlogged,denied
Mike,2011-10-31 22:24:57:0254167,unlogged,denied
Mike,2011-10-31 22:24:57:0355174,locked,lock
Dwigth,2011-10-31 22:24:57:0451151,unlogged,denied
Dwigth,2011-10-31 22:24:58:0452175,unlogged,denied
Carl,2011-10-31 22:24:58:0951183,locked,lock
Mike,2011-10-31 22:24:58:0952313,unlogged,unlocked
Dwigth,2011-10-31 22:24:59:0203108,logged,logged
Mike,2011-10-31 22:25:00:0356109,unlogged,denied
```