Temporal Kodkod (WIP)

Nuno Macedo Alcino Cunha

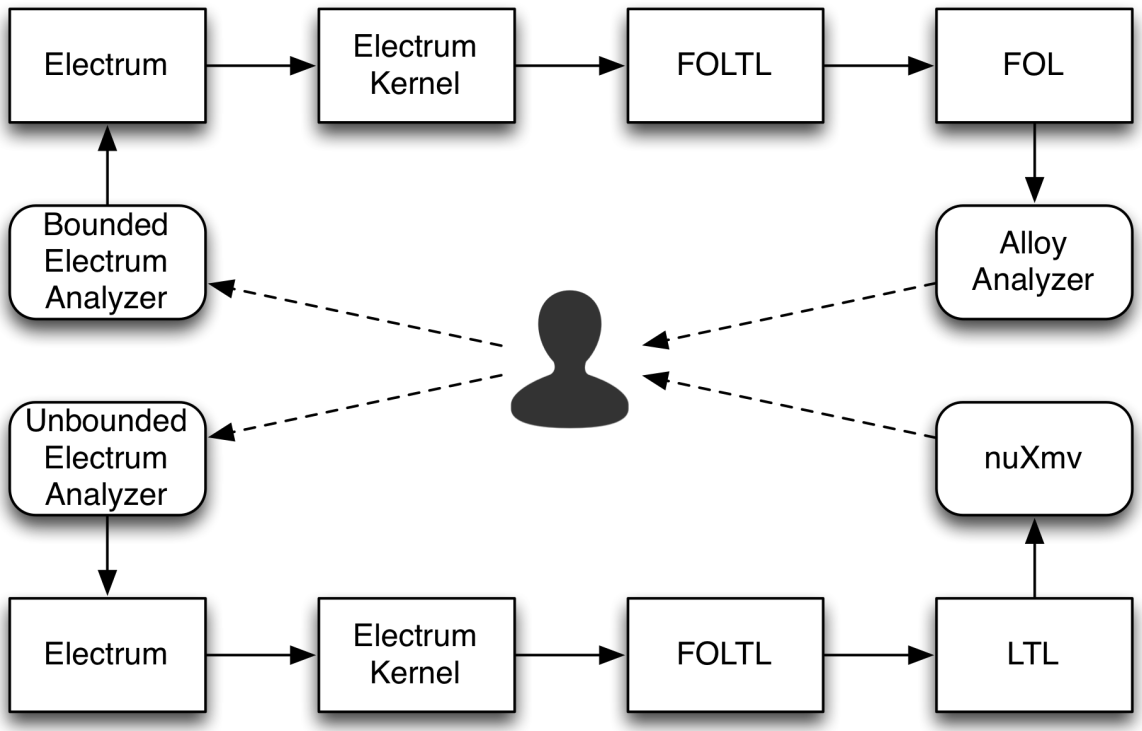
TRUST Workshop 2016
September
Braga, Portugal

Electrum

- A **lightweight formal specification language**, inspired by Alloy and TLA that simplifies the specification of dynamic systems with rich configurations
- A bounded and an unbounded **model-checking technique** to verify such systems, i.e., whether temporal properties hold for every possible configuration

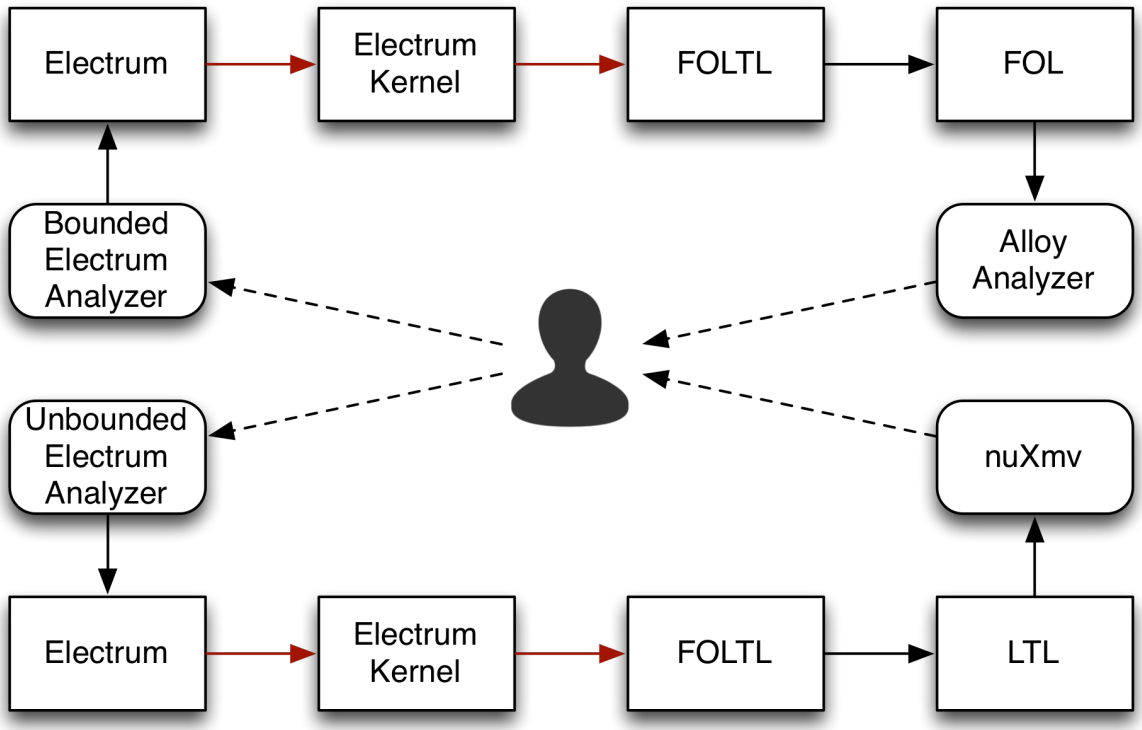
Electrum

Actual architecture:



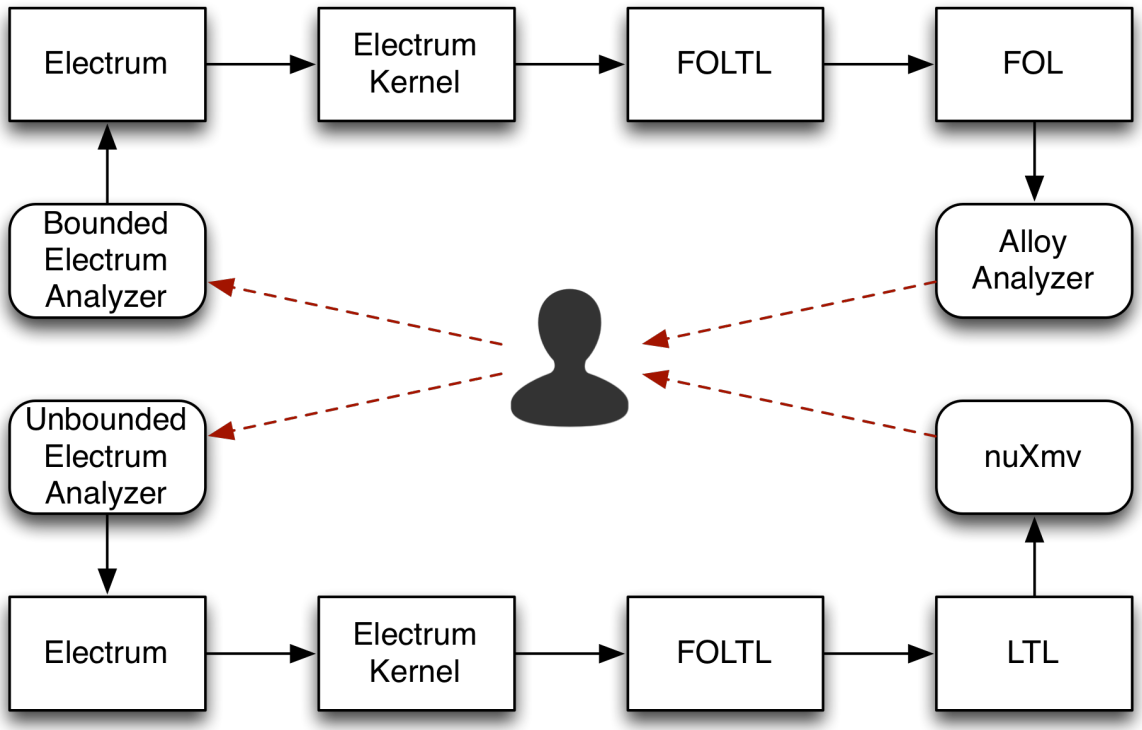
Electrum

Actual architecture:



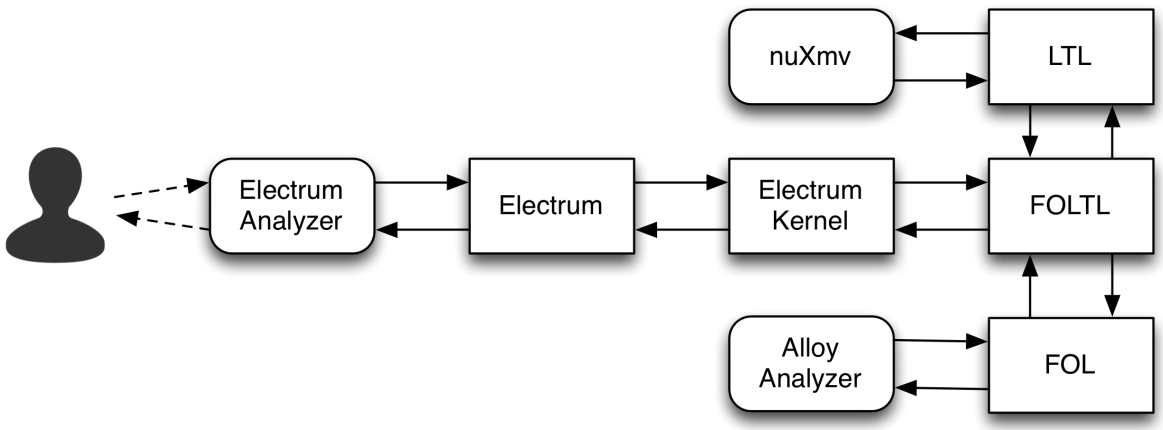
Electrum

Actual architecture:



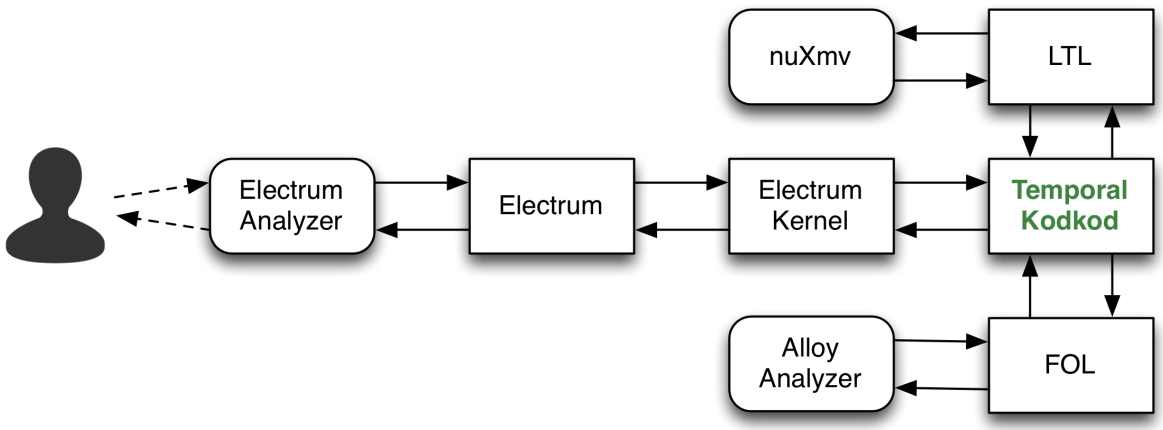
Electrum

Planned architecture:



Electrum

Planned architecture:



Model Finding

- Bounded search for models that satisfy given constraints
- Abstracts several consistency management tasks
- Simulation, verification, model checking...

Model Finding

- Archetype: **Kodkod**
- Bounded universe of atoms $\mathcal{U}$
- Relational variable declarations B (upper- and lower-bounds)
- Relational first-order-logic formulas ϕ
- Solutions bind relations in B with tuples from $\mathcal{U}$ and satisfy ϕ

Model Finding

- **Automated** finding via SAT solving
- Bound declarations allow **partial** solutions
- Solutions can be **iterated** by using incremental SAT solvers
- Powerful **symmetry** breaking algorithm removes isomorphic solutions

Temporal Model Finding

- No native support for **dynamic** systems
- A temporal extension is in order
- Characteristic features must be preserved:
  - automated solving
  - partial solutions
  - solution iteration
  - symmetry breaking

Temporal Kodkod: Bounds

- Solutions are now (potentially infinite) sequences of binds
- Bounds must accept potentially infinites traces
- Transition relation is total and deterministic (essentially a stream)
- Would a more flexible structure be useful?
- Allows the definition of **partial** solutions traces

Temporal Kodkod: Formulas

- Formulas ϕ are now temporal
- LTL operators + primed variables
- Define the properties that the temporal solutions are expected to hold

Example

- Farmer river crossing puzzle

Example

$\mathcal{U}$ = {Left,Right}

```
B = left   : [{Left}]
    right  : [{Right}]

    farmer : [{}, {Left, Right}]
    fox    : [{}, {Left, Right}]
    goose  : [{}, {Left, Right}]
    beans  : [{}, {Left, Right}]
```

ϕ = **always** (**one** farmer **and one** fox **and one** goose **and one** beans)

```
    farmer + fox + goose + beans = left

    always (
      farmer' != farmer and
      (farmer = fox and fox' != fox and goose' = goose and beans' = beans) ||
      (farmer = goose and fox' = fox and goose' != goose and beans' = beans) ||
      (farmer = beans and fox' = fox and goose' = goose and beans' != beans)
    )

    always (
      not (farmer != fox and fox = goose) and
      not (farmer != goose and goose = beans)
    )

    eventually ( farmer + fox + goose + beans = right )
```

Example

$\mathcal{U}$ = {Left,Right}

```
B = left   : [{Left}]
    right  : [{Right}]

    farmer : [{}, {Left, Right}]
    fox    : [{}, {Left, Right}]
    goose  : [{}, {Left, Right}]
    beans  : [{}, {Left, Right}]
```

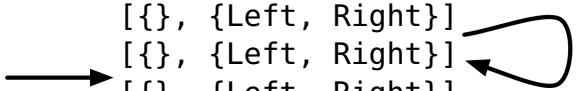
ϕ = **always** (**one** farmer **and one** fox **and one** goose **and one** beans)

```
    farmer + fox + goose + beans = left

    always (
      farmer' != farmer and
      (farmer = fox and fox' != fox and goose' = goose and beans' = beans) ||
      (farmer = goose and fox' = fox and goose' != goose and beans' = beans) ||
      (farmer = beans and fox' = fox and goose' = goose and beans' != beans)
    )

    always (
      not (farmer != fox and fox = goose) and
      not (farmer != goose and goose = beans)
    )

    eventually ( farmer + fox + goose + beans = right )
```

Example

$\mathcal{U}$ = {Left,Right}

```
B = left   :S [{Left}]
    right  :S [{Right}]

    farmer :V [{Left}]      [{}, {Left, Right}]
    fox    :V [{Left}]      [{}, {Left, Right}]
    goose  :V [{Left}]      [{}, {Left, Right}]
    beans  :V [{Left}]      [{}, {Left, Right}]
```



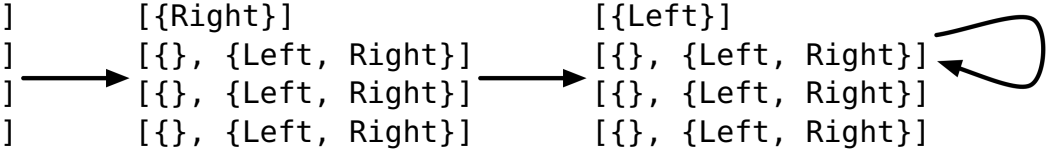
ϕ = **always** (**one** farmer **and one** fox **and one** goose **and one** beans)

```
    always (
      farmer' != farmer and
      (farmer = fox and fox' != fox and goose' = goose and beans' = beans) ||
      (farmer = goose and fox' = fox and goose' != goose and beans' = beans) ||
      (farmer = beans and fox' = fox and goose' = goose and beans' != beans)
    )

    always (
      not (farmer != fox and fox = goose) and
      not (farmer != goose and goose = beans)
    )

    eventually ( farmer + fox + goose + beans = right )
```

Example

$\mathcal{U}$ = {Left,Right}

B = left :S [{Left}]
right :S [{Right}]

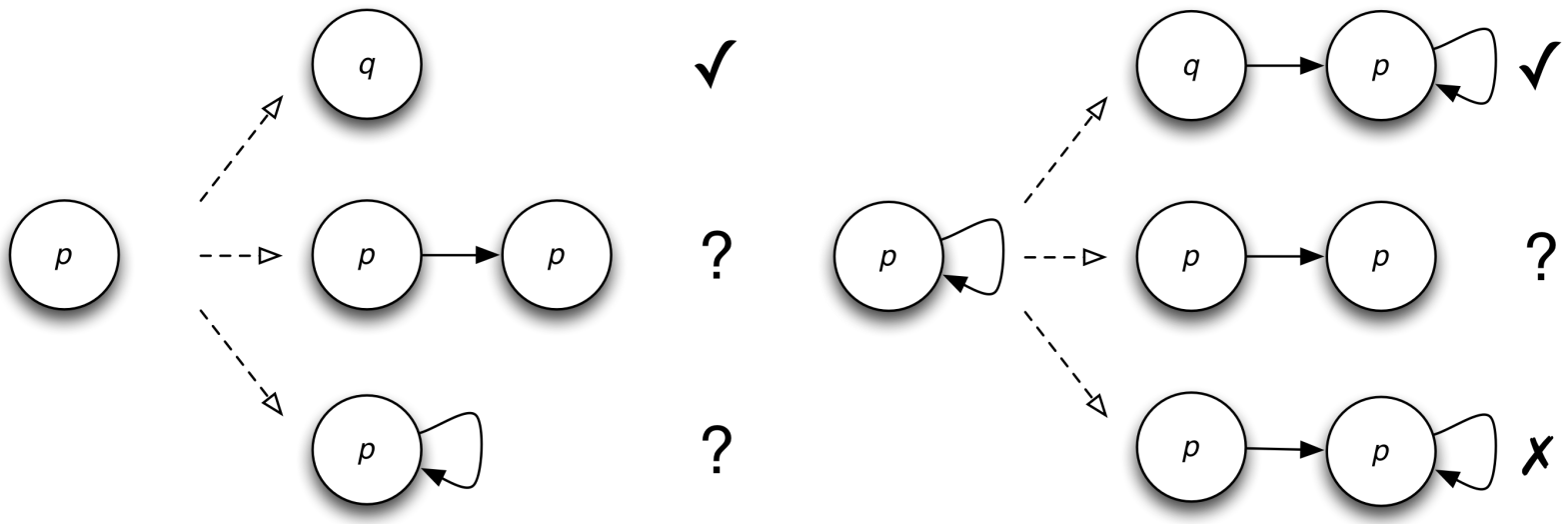


ϕ = **always** (**one** farmer **and one** fox **and one** goose **and one** beans)

```
always (
  (farmer = fox and fox' != fox and goose' = goose and beans' = beans) ||
  (farmer = goose and fox' = fox and goose' != goose and beans' = beans) ||
  (farmer = beans and fox' = fox and goose' = goose and beans' != beans)
)

always (
  not (farmer != fox and fox = goose) and
  not (farmer != goose and goose = beans)
)

eventually ( farmer + fox + goose + beans = right )
```

Solution Iteration

- Should finite prefixes be considered valid solutions?
- Which solutions should be removed when iterating?



- Suitable temporal symmetry breaking should address these issues

Temporal Kodkod: Solving

- Currently deployed iteratively over SAT
- Temporal bounds are unfolded as needed
- Guarantees minimal traces

State of the Work

- Implemented as an extension of Kodkod, deployed over SAT
- (Bounded) Electrum built over it
- Derived bounds are "constant"
- Alloy Analyzer adapted to present temporal solutions
- Several open questions

Plan

- Deployment over unbounded model checkers (SMV)
- Advanced scenario exploration
- Decomposed, parallel solving of configurations
- Symmetry breaking?
- Derive stricter bounds from Electrum (initial state)