Nuno Macedo[1] Julien Brunel[2] David Chemouil [2] Alcino Cunha [3]

VERIFYING TEMPORAL RELATIONAL MODELS WITH PARDINUS

[1]University of Porto, Portugal

[2]ONERA/DTIS, Toulouse

[3]University of Minho, Portugal

June 2, 2023, ABZ 2023, LORIA, France

CONTEXT: ANALYSIS OF ALLOY MODELS

Alloy Language (MIT, 2006)

- Specification language based on First-Order Logic and Temporal Logic
- Additional relational operator : transitive closure
- Inspired by UML, user-friendly

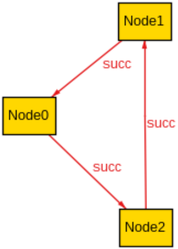
Alloy Analyzer

- Bounded verification $\rightarrow$ Decidability
- Use of SAT solvers $\rightarrow$ Efficiency, quick feedback

SPECIFYING THE STRUCTURE OF A SYSTEM

Example: a ring-shaped network

```
sig Node {
  succ : one Node,
}
fact ringShaped {
  all n: Node |  Node in n.^succ
}
```



TEMPORAL REASONING IN ALLOY

- Included in a Alloy 6 (November 2021)
- First proposed as Electrum [FSE 2016]

Main features

- Fields and signatures can be *mutable* (declared with a **var** keyword)
- The language includes *LTL* connectives and primed variables

Different backends

- Bounded Model Checking
- Unbounded Model Checking

TEMPORAL REASONING IN ALLOY

- Included in a Alloy 6 (November 2021)
- First proposed as Electrum [FSE 2016]

Main features

- Fields and signatures can be *mutable* (declared with a **var** keyword)
- The language includes *LTL* connectives and primed variables

Different backends

- Bounded Model Checking
- Unbounded Model Checking

EXAMPLE

```
sig Id {}
sig Node {
  succ : Node,
  id : Id,
  var inbox, outbox: set Id,
}

fact distinctIds {
  all i: Id | lone id.i
}

fact ring {
  all n: Node | Node in n.^succ
}

fun elected : set Node {
  {n : Node | once (n.id in n.inbox)}
}

pred init [] { ... }
pred skip [] { ... }
pred compute[n : Node] {...}
```

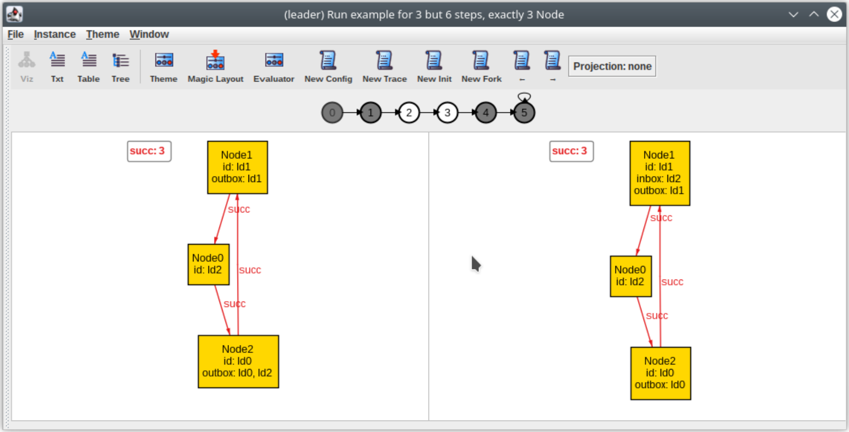
```
pred send [n : Node] {
  some i: n.outbox {
    outbox' = outbox - n->i
    inbox' = inbox + n.succ->i
  }
}

fact traces {
  init
  always (some n: Node | send[n] or compute[n]}
                         or skip)
}
assert Safety {
  always lone elect
}
assert Liveness {
    eventually some elect
}
run {} for 3
check Safety for 4
check Liveness for 4
```

EXAMPLE

```
sig Id {}
sig Node {
  succ : Node,
  id : Id,
  var inbox, outbox: set Id,
}

fact distinctIds {
  all i: Id | lone id.i
}

fact ring {
  all n: Node | Node in n.^succ
}

fun elected : set Node {
  {n : Node | once (n.id in n.inbox)}
}

pred init [] { ... }
pred skip [] { ... }
pred compute[n : Node] {...}
```

```
pred send [n : Node] {
  some i: n.outbox {
    outbox’ = outbox – n→i
    inbox’ = inbox + n.succ→i
  }
}

fact traces {
  init
  always (some n: Node | send[n] or compute[n]}
                               or skip)
}
assert Safety {
  always lone elect
}
assert Liveness {
    eventually some elect
}
run {} for 3
check Safety for 4
check Liveness for 4
```

TRACE OF THE SYSTEM

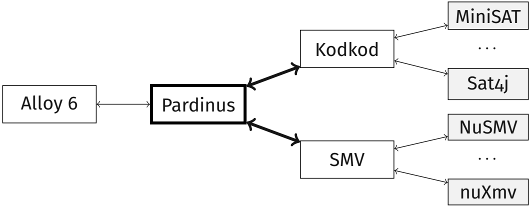


IN THE PAPER

Pardinus: an efficient backend for Alloy

- A unified backend for bounded and unbounded verification of Alloy models
- A path iteration mechanism: returns non-isomorphic solutions, efficiently implemented using incremental SAT solving
- A decomposed analysis technique that relies on symbolic bounds and parallel execution to speed up verification

[JAR 2022] N. Macedo, J. Brunel, D. Chemouil, A. Cunha. Pardinus: A Temporal Relational Model Finder

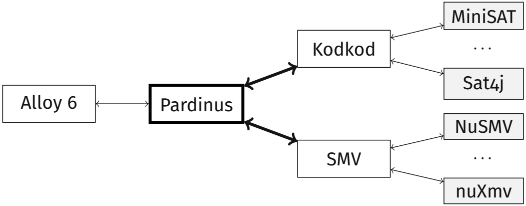


IN THE PAPER

Pardinus: an efficient backend for Alloy

- A unified backend for bounded and unbounded verification of Alloy models
- **A path iteration mechanism**: returns **non-isomorphic solutions**, efficiently implemented using incremental SAT solving
- A decomposed analysis technique that relies on symbolic bounds and parallel execution to speed up verification

[JAR 2022] N. Macedo, J. Brunel, D. Chemouil, A. Cunha. Pardinus: A Temporal Relational Model Finder



A PARDINUS PROBLEM

```
univ = {I0,I1,I2,I3,N0,N1,N2,N3}

Node       :1 {} {(N0),(N1),(N2),(N3)}
Id         :1 {(I0),(I1),(I2),(I3)} {(I0),(I1),(I2),(I3)}
id         :2 {} {(N0,I0),(N0,I1),(N0,I2),(N0,I3),...,
                 (N3,I0),(N3,I1),(N3,I2),(N3,I3)}
succ       :2 {} {(N0,N0),(N0,N1),(N0,N2),(N0,N3),...,
                 (N3,N0),(N3,N1),(N3,N2),(N3,N3)}
var outbox :2 {} {(N0,I0),(N0,I1),(N0,I2),(N0,I3),...,
                 (N3,I0),(N3,I1),(N3,I2),(N3,I3)}

id in Node → Id and
all n : Node | one n.id and
all i : Id | lone id.i and
...
always (some n: Node | some i: n.outbox ...)
...
```

A PARDINUS PROBLEM

```
univ = {I0,I1,I2,I3,N0,N1,N2,N3}

Node       :1 {} {(N0),(N1),(N2),(N3)}
Id         :1 {(I0),(I1),(I2),(I3)} {(I0),(I1),(I2),(I3)}
id         :2 {} {(N0,I0),(N0,I1),(N0,I2),(N0,I3),...,
                  (N3,I0),(N3,I1),(N3,I2),(N3,I3)}
succ       :2 {} {(N0,N0),(N0,N1),(N0,N2),(N0,N3),...,
                  (N3,N0),(N3,N1),(N3,N2),(N3,N3)}
var outbox :2 {} {(N0,I0),(N0,I1),(N0,I2),(N0,I3),...,
                  (N3,I0),(N3,I1),(N3,I2),(N3,I3)}

id in Node → Id and
all n : Node | one n.id and
all i : Id | lone id.i and
...
always (some n: Node | some i: n.outbox ...)
...
```

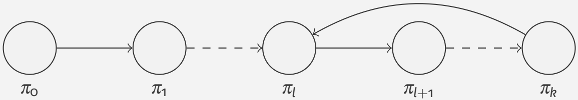
SEMANTICS OF PARDINUS

A Pardinus solution

A solution of a Pardinus problem is a path, *i.e.*, an infinite sequence of bindings from the declared relations to constants that

- always respects the declared bounds
- and satisfies the temporal formula.

Small model property of LTL



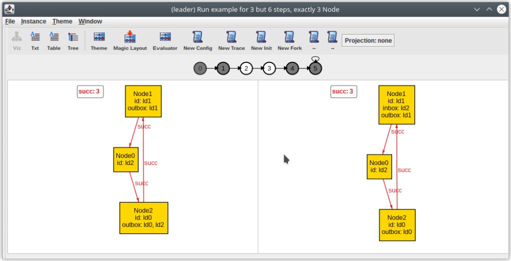
We can consider bounded witnesses of infinite sequences w.l.o.g.

SEMANTICS OF PARDINUS

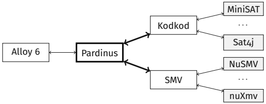
A Pardinus solution

A solution of a Pardinus problem is a path, *i.e.*, an infinite sequence of bindings from the declared relations to constants that

- always respects the declared bounds
- and satisfies the temporal formula.

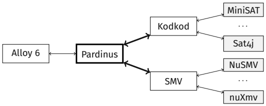


TRANSLATION TO RELATIONAL LOGIC



$$
\begin{aligned}
\langle \Gamma \textbf{ in } \Delta \rangle_s &= \langle \Gamma \rangle_s \textbf{ in } \langle \Delta \rangle_s \\
\langle \textbf{some } \Gamma \rangle_s &= \textbf{some } \langle \Gamma \rangle_s \\
\langle \textbf{all } x : \Gamma \mid \phi \rangle_s &= \textbf{all } x : \langle \Gamma \rangle_s \mid \langle \phi \rangle_s \\
\langle \textbf{after } \phi \rangle_s &= \langle \phi \rangle_{s.\mathtt{next}} \\
\langle \phi \textbf{ until } \psi \rangle_s &= \textbf{some } s_0 : s.{*}\mathtt{next} \mid \langle \psi \rangle_{s_0} \textbf{ and} \\
& \textbf{all } s_1 : \mathtt{upto}[s, s_0] \mid \langle \phi \rangle_{s_1} \\
\langle \mathtt{before}\ \phi \rangle_s &= \textbf{some } s_0 : \mathtt{succ}.s \mid \langle \phi \rangle_{s_0} \\
\langle \phi \textbf{ since } \psi \rangle_s &= \textbf{some } s_0 : {*}\mathtt{succ}.s \mid \langle \psi \rangle_{s_0} \textbf{ and} \\
& \textbf{all } s_1 : \mathtt{downto}[s, s_0] \mid \langle \phi \rangle_{s_1}
\end{aligned}
$$

TRANSLATION TO TEMPORAL LOGIC



$$
\begin{aligned}
[\Gamma\ \mathbf{in}\ \Delta]_\sigma &= \bigwedge_{t \in [\![\Gamma]\!]_\sigma} ([\Gamma]_\sigma(t) \Rightarrow [\Delta]_\sigma(t)) \\
[\mathbf{some}\ \Gamma]_\sigma &= \mathtt{count}\{[\Gamma]_\sigma(t) | t \in [\![\Gamma]\!]_\sigma\} \geq 1 \\
[\mathbf{all}\ x : \Gamma \mid \phi]_\sigma &= \bigwedge_{t \in [\![\Gamma]\!]_\sigma} \left([\Gamma]_\sigma(t) \Rightarrow [\phi]_{\sigma[x \mapsto t]}\right) \\
[\mathbf{after}\ \phi]_\sigma &= \mathsf{X}[\phi]_\sigma \\
[\phi\ \mathbf{until}\ \psi]_\sigma &= [\phi]_\sigma\, \mathsf{U}\, [\psi]_\sigma \\
[\mathtt{before}\ \phi]_\sigma &= \mathsf{Y}[\phi]_\sigma \\
[\phi\ \mathbf{since}\ \psi]_\sigma &= [\phi]_\sigma\, \mathsf{S}\, [\psi]_\sigma
\end{aligned}
$$

ITERATION ON SOLUTIONS

- Technique to explore the behaviours that satisfy an arbitrary temporal logic specification
- Interactive exploration mode akin to simulation
- Unified interface for simulating the modelled system and exploring its counter-examples
- Formalised with state/event linear temporal logic

EXPLORATION DEMO

SYMMETRIES

Symmetry

A symmetry is a permutation P over **`univ`** such that for any path π,

$$\pi \text{ is a solution iff } P(\pi) \text{ is a solution,}$$

where $P(\pi)$ applies the permutation P to all bindings in π.

SYMMETRY BREAKING (NON TEMPORAL CASE)

```
univ = {A0, A1}

a :1 {} {(A0),(A1)}
r :2 {} {(A0,A0),(A0,A1),(A1,A0),(A1,A1)}
```

Symmetry

P = A0 $\mapsto$ A1

Symmetry Breaking Predicate

Idea:

- chose an ordering over the relations $\Rightarrow$ lexicographic ordering over the solutions;
- for each set of symmetrical solutions, we only keep the smallest solution.

[a(A0), r(A0,A0), r(A0,A1)] $\leqslant$ [a(A1), r(A1,A1), r(A1,A0)]

SYMMETRY BREAKING (NON TEMPORAL CASE)

```
univ = {A0, A1}

a :1 {} {(A0),(A1)}
r :2 {} {(A0,A0),(A0,A1),(A1,A0),(A1,A1)}
```

Symmetry

P = A0 $\mapsto$ A1

Symmetry Breaking Predicate

[a(A0), r(A0,A0), r(A0,A1)] $\leqslant$ [a(A1), r(A1,A1), r(A1,A0)]

Propositional encoding

a(A0) $\rightarrow$ a(A1) $\wedge$
a(A0)=a(A1) $\rightarrow$ (r(A0,A0) $\rightarrow$ r(A1,A1))$\wedge$
(a(A0)=a(A1)$\wedge$ r(A0,A0)=r(A1,A1)) $\rightarrow$ (r(A0,A1) $\rightarrow$ r(A1,A0))

SYMMETRY BREAKING (TEMPORAL CASE)

```
univ = {A0, A1}

var a :1 {} {(A0),(A1)}
var b :1 {} {(A0),(A1)}
```

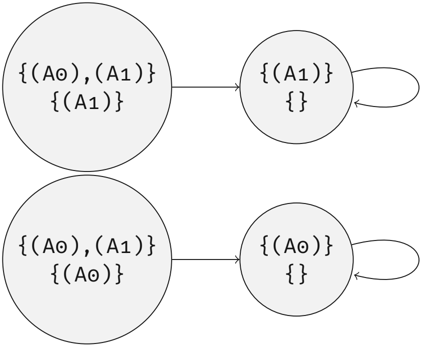
Symmetry Breaking Predicate

[a(A0), b(A0)] $\leqslant$ [a(A1), b(A1)]

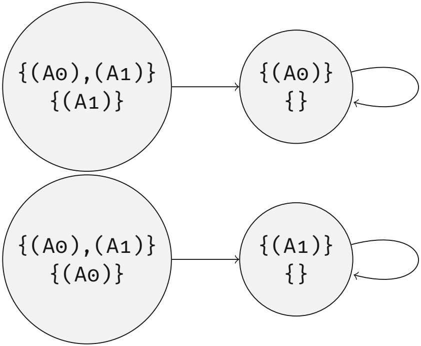
Propositional encoding (for a sequence of length 2)

a(A0,0) $\rightarrow$ a(A1,0) $\wedge$
a(A0,0)=a(A1,0) $\rightarrow$ (b(A0,0) $\rightarrow$ b(A0,0)) $\wedge$
(a(A0,0)=a(A1,0)$\wedge$ b(A0,0)=b(A1,0)) $\rightarrow$ (a(A0,1) $\rightarrow$ a(A1,1)) $\wedge$
...

EXAMPLE OF SYMMETRIC PATHS



EXAMPLE OF SYMMETRIC PATHS (2)



CONCLUSION

- Pardinus is a relational model finder used as a backend for Alloy offering
  - a novel iteration mechanism,
  - efficient solving algorithms.

[JAR 2022] N. Macedo, J. Brunel, D. Chemouil, A. Cunha. Pardinus: A Temporal Relational Model Finder

Ongoing and future works

Extensions of Alloy encoded by translation to Pardinus:

- An event layer to ease the specification of automata-like transition systems
- Adding a data structure for records