Efficient Model Checking for *Circus* Using FDR

26th May 2015

Introduction

Aims

- Introduce Model Checking
- Talk about efficient Model Checking for *Circus*...

Topics

1. Model Checking Overview
2. FDR Introduction
3. *Circus* Introduction
4. Model Checking Problems

Thanks

- Alvaro Miyazawa
- Tom Gibson-Robinson (Oxford)

Model Checking

What is Model Checking?

- Technique for verifying concurrent systems
- Determines if M is a model for a formula f
  - M is a transition system
  - f is a temporal logic specification
- That is, M exhibits whatever property f captures
- In traditional Model Checking, M and f are written in different languages

Model Checking

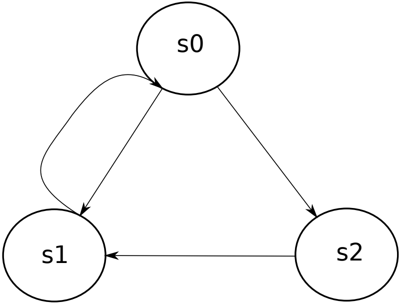
Temporal Logic

- In temporal logic a formula has dynamic truth
  - A formula can be true for some of a system's states...
  - But false for others
- Usually has operators to specify...
  - What happens next
  - What happens sometime in the future
  - Something that happens for all future states
  - And logic operators like negation, conjunction and disjunction

Model Checking

Transition Systems

- Model a system using. . .
    - States (static structure)
    - Transitions (dynamic structure)
- Finite transition systems can be expressed as directed graphs



Model Checking

Advantages

- No need to write proofs. . .
    - Although the model and the specification are needed
- Automatic. . .
    - After the model and specification are written
- Concurrency errors. . .
    - Difficult to reproduce with testing
- Fast. . .
    - Compared to other rigorous methods, like proof checking
- Counterexamples. . .
    - Invaluable for debugging

Model Checking

Disadvantages

- Finite state. . .
    - Automation generally requires finite state systems
- State Explosion. . .
    - Can be a big problem, most of the problems we address are due to this

Failures-Divergences Refinement

FDR

- Model Checking tool...
    - Although it's really a refinement checker
- Model and the specification are the same language...
    - `CSPm`, the machine-readable version of CSP
- Performing a check...
    - **Compile** both CSP processes into Labelled Transition Systems
    - **Check** if one is a refinement of the other

Failures-Divergences Refinement

Refinement

- Is a model an implementation of a specification?
- $P \sqsubseteq Q$ if every behaviour of Q is also a behaviour of P
    - Q exhibits the property captured by P
    - Q implements P

Semantic Models

- Traces refinement $\sqsubseteq_{\mathrm{T}}\ldots$
    - $P \sqsubseteq_{\mathrm{T}} Q$ if every trace of Q is a trace of P
- Failures Refinement $\sqsubseteq_{\mathrm{F}}\ldots$
    - Failure: a trace leading to events that may be refused
- Failures-Divergences Refinement $\sqsubseteq_{\mathrm{FD}}\ldots$
    - Divergence: a trace leading to undefined behaviour

Circus and `CSPm` Introduction

Circus

- Combines. . .
    - CSP to capture Behaviour
    - Z to capture Data
- Variants. . .
    - Object Orientation – Oh*Circus*
    - Time – *Circus*Time
- Model Checking in FDR. . .
    - Requires translation to `CSPm`
    - Different capabilities. . .

Circus and CSPm Introduction

CSPm

- Events. . .
    - Model important points in the history of the system
    - May communicate parameters (`c!x` or `c?x` or simply `c.x`)
    - Parameters can be restricted to certain values (`c?x:set`)
    - Sequenced using prefix (`c -> P`)
    - May be guarded by a predicate (`(guard)& c -> P`)
- Processes in. . .
    - External choice (`P [] Q`)
    - Interleaving (`P ||| Q`)
    - Parallel (`P [|X|] Q`)
    - Sequence (`P ; Q`)
- Replication (e.g. `||| s : set @ P(s)`)

Model Checking Problems

Problem Classes

- Data...
    - FDR struggles with...
        - Large data-types
        - Processes holding several unrelated parameters of large data-types
        - Infinite types
- Structure...
    - FDR struggles with...
        - Large numbers of states
        - Large orderings of processes from parallelisms

Model Checking Problems

Data Problems

- *Circus* has variables but CSP does not...
    - To represent variables in CSP we use processes to control variables
    - e.g. `P(num)` to control the variable `num`
- FDR evaluates and compiles all possible combinations of states
- Strive to reduce the state space in the model...

Data Problems

Reducing State Space

- Bound large types
  - For example `{0..50}`
- Use direct function calls to pass large parameters, if possible. . .
  - Uses the functional language elements of `CSPm`
  - e.g instead of accumulating the set with

    ```
    P(set) = add?x -> P(union(set, x))...
    ```
  - Call `P({1..100})` to directly pass the set
- Refactor processes that control large state variables into many smaller processes in interleaving. . .

Data Example One

Use Case

- Model contains a set of data of type `value`
- We can put values in and take values out
- Attempting to put the same value in twice is not possible

Data Example One

Process Controlling a Set: Bad

```
P1(set) =
    in?x:set -> P1(set)
    []
    in?x:diff(value,set) -> P1(union(set, x))
    []
    out?x:set -> P1(diff(set, x))
```

Data Example One

Process Controlling a Set: Good

```
P₂ = ||| x :  value @ NotMember(x)

NotMember(x) =
    in.x -> Member(x)

Member(x) =
    in.x -> Member(x)
    []
    out.x -> NotMember(x)
```

Data Example One

Comparison

- Since both processes can only add the same number to the set once, they are equivalent...
    - $P_1 \sqsubseteq_{\mathrm{T}} P_2$ and $P_2 \sqsubseteq_{\mathrm{T}} P_1$
    - $P_1 \sqsubseteq_{\mathrm{F}} P_2$ and $P_2 \sqsubseteq_{\mathrm{F}} P_1$
    - $P_1 \sqsubseteq_{\mathrm{FD}} P_2$ and $P_2 \sqsubseteq_{\mathrm{FD}} P_1$

Data Example One

Deadlock Freedom Check

		`value=`			
		`{0..5}`	`{0..10}`	`{0..15}`	`{0..20}`
P_1	Compiled	0.02s	0.69s	39.57s	7438.52s[1]
	Checked	0.04	0.02s	0.12s	3.84s
P_2	Compiled	0.01s	0.01s	0.01s	0.01s
	Checked	0.06s	0.08s	0.23s	4.41s

---

[1] $\sim$ 2 hours

Data Example One

Explanation

- P_1 takes longer to compile than P_2 but P_2 takes longer to check
- There is a trade-off between the compilation time of a set and the checking time of many interleaved actions
  - Checking phase can be more easily parallelised

Data Example Two

Use Case

- Model contains a sequence representing a stack
- We can push and pop a value, and check the top element
- We cannot push more than a max number of values
- We cannot pop a value or check the top value of an empty stack

Data Example Two

Process Controlling a Sequence: Bad

```
Q₁(sequence) =
    (#sequence < maxStackId) &
        push?x ->
        Q₁(<x>^sequence)
    []
    (not null(sequence))  &
    (
        pop ->
        Q₁(tail(sequence))
        []
        top!head(sequence) ->
        Q₁(sequence)
    )
```

Data Example Two

Process Controlling a Sequence: Good

```
Q₂ =
    (|| i :  value @ [ AlphaFree(i) ]
          Free(i)
    ) \{| resume|}
```

Data Example Two

Process Controlling a Sequence: Good

```
Free(id) = push.id?v -> Full(id, v)

Full(id, v) =
     pop.id-> (if id > minStackId then
             resume.id-1 -> Free(id)
         else
             Free(id))
     []
     (id < maxStackId) &
         push.id+1?_ -> resume.id -> Full(id, v)
     []
     top.id!v -> Full(id, v)
```

Data Example Two

Comparison

- Since this is only a change of structure (once we apply renaming to Q_2) these processes are equivalent...
    - $Q_1 \sqsubseteq_{\mathrm{T}} Q_2$ and $Q_2 \sqsubseteq_{\mathrm{T}} Q_1$
    - $Q_1 \sqsubseteq_{\mathrm{F}} Q_2$ and $Q_2 \sqsubseteq_{\mathrm{F}} Q_1$
    - $Q_1 \sqsubseteq_{\mathrm{FD}} Q_2$ and $Q_2 \sqsubseteq_{\mathrm{FD}} Q_1$

Data Example Two

Deadlock Freedom check

		`value=`			
		`{0..5}`	`{0..6}`	`{0..7}`	`{0..8}`
Q_1	Compiled	0.82s	13.92s	417.83s	21462.05[2]
	Checked	0.02s	0.07s	0.99s	N/A[3]
Q_2	Compiled	0.02s	0.02s	0.03s	0.03s
	Checked	0.09s	0.14s	0.89s	18.57s

---

[2] ∼ 5.9 hours

[3] FDR crashed. . .

Data Example Two

Explanation

- Q_1 compiles slower due to checking the length of the sequence
- Q_2 compiles faster because each process only controls one variable
- Trade-off in terms of checking. . .
    - Q_1 controls a sequence, but is sequential
    - Q_2's sub-processes only control one variable each but being are parallel

Process Structure Problems

Large Possible Orderings of Behaviour

- FDR explores all the possible states
  - Before any hiding or parallel restrictions can occur
- This can cause FDR to use a lot of memory

Structure Example One

Use Case

- Waiting for many subordinate processes to signal readiness. . .
- Which may happen in any order

Structure Example One

For a process. . .

```
R(x) = ready.x -> SKIP
```

Simple Replicated Interleave. . .

```
Interleave = ||| x :  value @ R(x)
```

. . . Becomes Replicated Sequential Composition

```
Sequential = ; x :  seq(value) @ R(x)
```

Structure Example One

Comparison

- $Interleave \sqsubseteq_{\mathrm{T}} Sequential$ but not the other way around...
  - Because $Interleave$ can perform `ready.x` events in any order, whereas in $Sequential$ the order is fixed
- Refinement does not hold in any other semantic models
- $Sequential$ can be said to implement $Interleave$

Structure Example One

Deadlock Freedom check

		`value=`			
		`{0..15}`	`{0..20}`	`{0..25}`	`{0..30}`
Intr	Compiled	0.01s	0.01s	0.01s	0.01s
	Checked	0.17s	1.99s	77.34s	3455.14s[4]
Seq	Compiled	0.00s	0.00s	0.01s	0.01s
	Checked	0.05s	0.05s	0.04s	0.05s

---

[4] ∼ 57.5 minutes

Structure Example One

Explanation

- $R(x)$ is a simple process so both versions are fast
- $Interleave$ is slower to check because the events may happen in any order
- $Sequential$ is much quicker because FDR chooses an arbitrary order for the events to happen in
- However, these processes are not equivalent

Summary

Model Checking

- Verification technique useful for concurrent systems
- Usually automatic and quick
- However, has some general limitations
    - State explosion being the most common

Summary

Efficient Model Checking in FDR

- Problems can occur when using data or parallelism yield a large number of state
- Solutions...
  - Use the functional elements of `CSPm`
  - Refactor processes controlling variables
  - Possibly refactor interleavings
- Mileage may vary...