Projector colour check

Structural elements

Example text

School of Informatics
University of Edinburgh

A practical introduction to games, infinity and the Edinburgh Concurrency Workbench

Perdita Stevens

June 29th, 2005

Outline

- The Edinburgh Concurrency Workbench
- Abstraction
- Games
- Implementation
- Conclusions

- The Edinburgh Concurrency Workbench
- Abstraction
- Games
- Implementation
- Conclusions

What can you do with it?

- Explore and simulate processes in (variants on) CCS
- Define and model check properties in the (full) modal μ-calculus
- Check lots of equivalences and preorders
- etc. (50+ commands)

Example session

```
Command: agent Cell = a.'b.Cell;
Command: agent C0 =  Cell[c/b];
Command: agent C1 =  Cell[c/a,d/b];
Command: agent C2 =  Cell[d/a];
Command: agent Buff3 = (C0 | C1 | C2)\{c,d};
Command: agent Spec =  a.Spec';
Command: agent Spec' = 'b.Spec + a.Spec'';
Command: agent Spec'' = 'b.Spec' + a.'b.Spec'';
Command: eq (Buff3, Spec);
true
```

Kinds of users

1. Students (widely used)
2. Research (experimental additions)
3. Industrial case studies (not so much now: performance outclassed)

Where is more information?

- CWB's on-line help
- CWB documentation at http://homepages.inf.ed.ac.uk/perdita/cwb
- Papers at http://homepages.inf.ed.ac.uk/perdita/
- Me (Perdita.Stevens@ed.ac.uk)

History

- Originally developed in 1980s
- Many accretions over following years
- Major redesign in mid 1990s
- Preliminary parameterisation and early games experiments
- Left alone since 2000 until last month...
- Steady stream of downloads continues

Good things and limitations

Good:

- Versatile program – many functions
- Reliable – no known semantic bugs
- Now reasonably modular and extensible
- Fundamental decision: work directly with PA terms

Limitations:

- Written in legacy language (SML-NJ)
- Never designed for maximising performance

Looking forward

CWB will go open source as soon as I get round to proposing a particular licence to the LFCS directorate.

However, I don't wish to continue development in SML-NJ, and migration prohibitive.

So: the value passing release, which I'll talk about today, will probably be the last major release of the CWB.

CWB will remain as prototyping environment.

For serious application, ideas will migrate to other tools...

Beyond finite state processes

CWB began by concentrating on single finite-state processes.

However, Edinburgh's research focus was infinite state processes. So...

1. CWB did most things "on the fly" - did not begin by creating an automaton. Often inefficient, but retained a chance of handling infinite state processes.
2. Glenn Bruns' Value Passing Front End...
3. Parameterisation capabilities (but not true value passing).

Rethink: what do we really want?

Users of the Edinburgh Concurrency Workbench wanted to be able to work with non-finite-state systems:

- value passing systems
- families of systems with unspecified numbers of components
- real-time systems?
- early and late variants of relations
- corresponding logics
- ...

We wanted a powerful, general way of understanding how to work with such systems, which on a practical level would also save effort in CWB development.

Value-passing CCS

a value may pass when two agents synchronise

Will use CWB syntax throughout this talk.

e.g.

VP processes are often naturally infinite state.

Value-passing CCS

a value may pass when two agents synchronise

Will use CWB syntax throughout this talk.

e.g.

a(x:int).'b(x).0|'a(3).0

VP processes are often naturally infinite state.

Value-passing CCS

a value may pass when two agents synchronise

Will use CWB syntax throughout this talk.

e.g.

a(x:int).'b(x).0|'a(3).0

- tau $\longrightarrow$

- a(VX0 : int) $\longrightarrow$

- 'a(VX0 : int) $\longrightarrow$

VP processes are often naturally infinite state.

Value-passing CCS

a value may pass when two agents synchronise

Will use CWB syntax throughout this talk.

e.g.

a(x:int).'b(x).0|'a(3).0

- tau → 'b(VNew1 : int).0|0
where (VNew1 : int=3)

- a(VX0 : int) → 'b(VNew1 : int).0 | 'a(3).0
where (VX0 : int=VNew1 : int)

- 'a(VX0 : int) → a(x : int).'b(x : unknown).0|0
where (VX0 : int=3)

VP processes are often naturally infinite state.

Testing values

Can see synchronisation as a kind of implicit test on a value.

Also have explicit testing, e.g.

agent A = in(x:int).<if (x=0) then ('zero.0) else ('nonzero.0)>;

Remark on approach

We fundamentally think in terms of *families of values*, not symbols.

E.g.

```
a(x:int).'b(x).0|'a(3).0
- a(VX0 : int) → 'b(VNew1 : int).0 | 'a(3).0
where (VX0 : int=VNew1 : int)
```

is an infinite family of transitions, parameterised on VX0 and VNew1.

Remark on approach

We fundamentally think in terms of *families of values*, not symbols.

E.g.

```
a(x:int).'b(x).0|'a(3).0
- a(VX0 : int) → 'b(VNew1 : int).0 | 'a(3).0
where (VX0 : int=VNew1 : int)
```

is an infinite family of transitions, parameterised on VX0 and VNew1.

(Presentation to the user, here and throughout, leaves a lot of scope for improvement!)

Implication for CWB

Can easily plug new process algebras into the CWB

- but sadly, this is outside the scope of that flexibility.

Where transition function returns (infinite) sets of transitions, all CWB functions need to be set-aware.

Currently, can only use a few commands (transitions, sim, strongeq, modelcheck,...) with value-passing processes.

Parameterised processes

Parameterised processes are closely related to VP processes.

They arise naturally in the course of computation:

```
'b(VNew1 : int).0|'a(3).0
```

That is, even to work with VP processes, we have to be able to reason about *families* of related processes.

- The Edinburgh Concurrency Workbench
- Abstraction
- Games
- Implementation
- Conclusions

Observation

Observation

Any interesting question (bisimulation, model-checking,...)

Observation

Any interesting question (bisimulation, model-checking,...)

on any reasonable language of value passing processes (e.g., including integers that can be incremented and a test for zero)

Observation

Any interesting question (bisimulation, model-checking,...)

on any reasonable language of value passing processes (e.g., including integers that can be incremented and a test for zero)

is undecidable.

Observation

Any interesting question (bisimulation, model-checking,...)

on any reasonable language of value passing processes (e.g., including integers that can be incremented and a test for zero)

is undecidable.

:-(

Need for abstraction

Often it's clear that although a process is technically infinite state, that doesn't actually matter.

E.g., *data independent* processes, like

agent Buff = in(x:int).'out(x).Buff;

Or cases where it's obvious that a finite quotient will do: e.g., an integer is used, but only odd/even matters.

Remarkably hard to formalise this, though – partly because it's often questions, not processes, which are "easy".

Little Red Workbench and the Undecidability Wolf

Little Red Workbench and the Undecidability Wolf

Who cares about decidability, anyway?

To summarise

- we need abstractions
- often, it's intuitively clear which abstraction
- no hope of defining languages that express only decidable problems
- but that's OK

So where can we get good abstractions?

Abstraction: manual or automatic

Until recently user had to define an abstraction manually.

Lately a lot of work has been done on automatic abstraction techniques.

CWB uses one such, generating an abstraction based on the question being answered (not just one process).

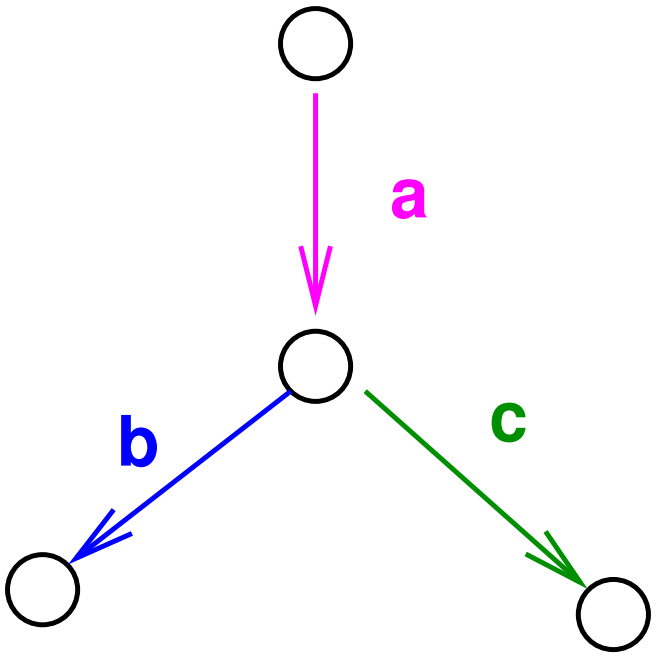
This is one of the roles of games.

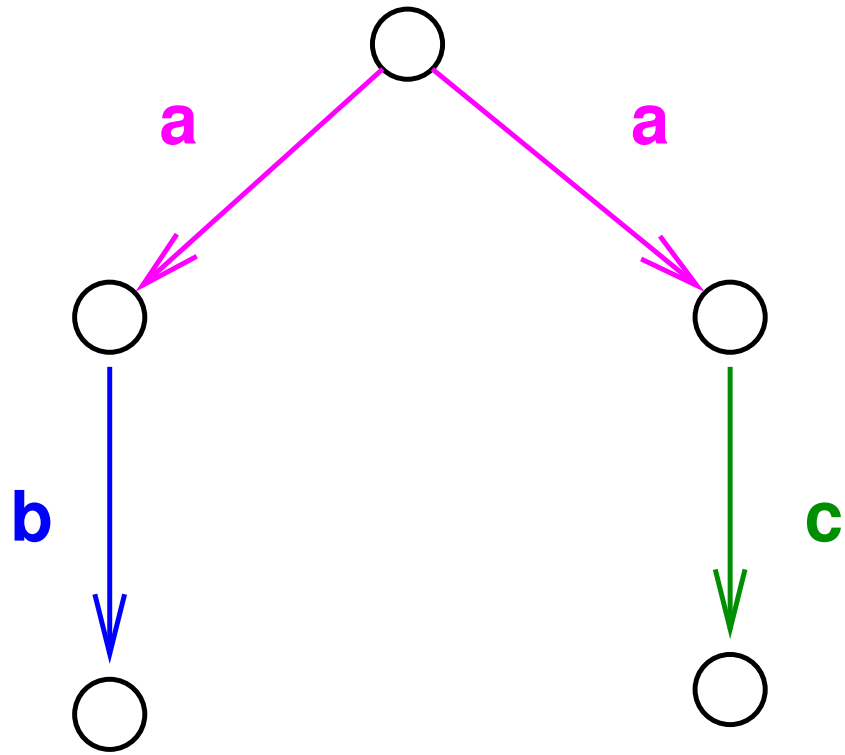
- The Edinburgh Concurrency Workbench
- Abstraction
- Games
- Implementation
- Conclusions

Games in verification

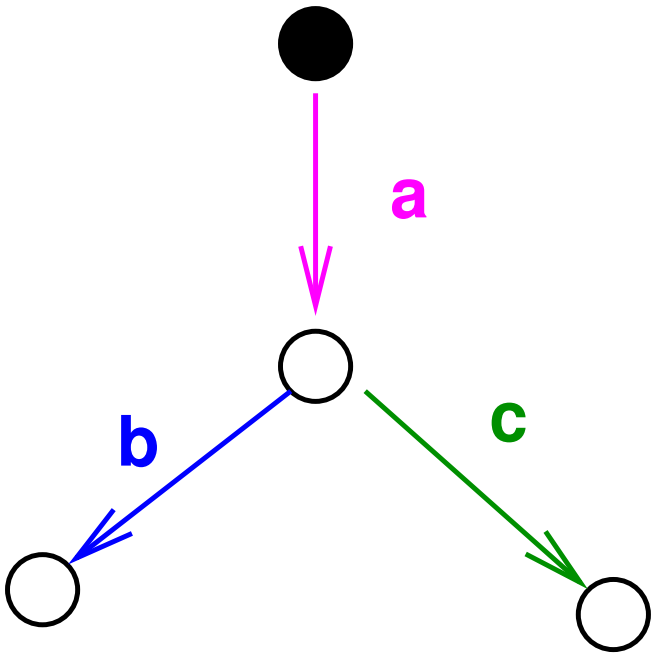
One player (Refuter, Abelard...) tries to demonstrate that the answer to the question is No, the other (Verifier, Eloise...) that it is Yes.

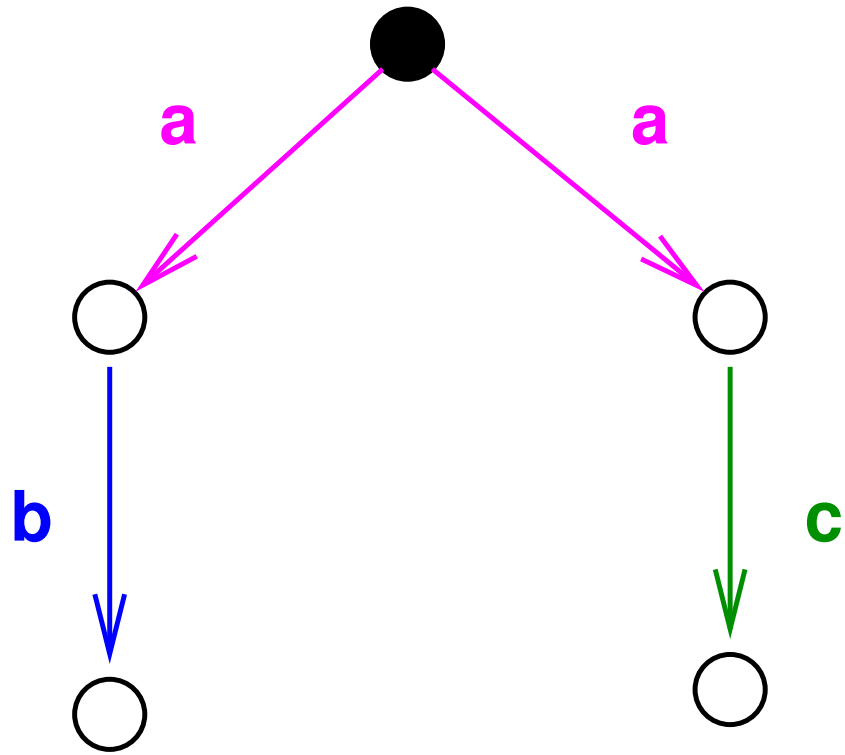
Playing a game



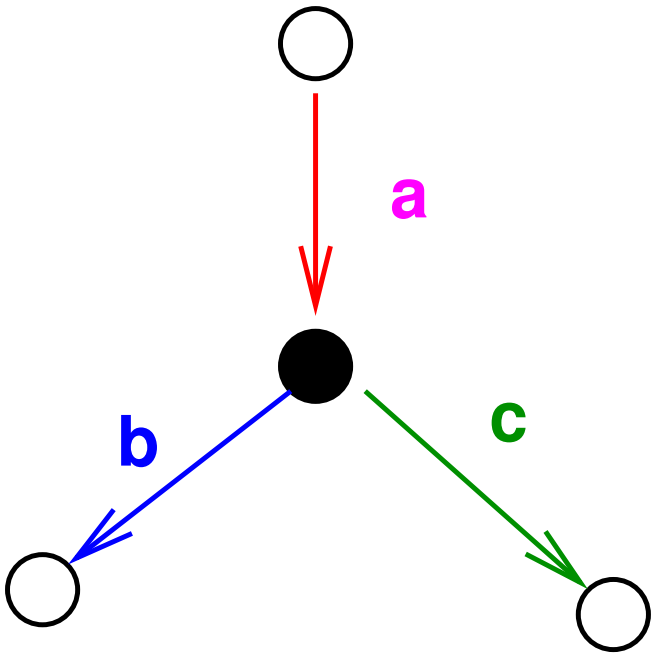


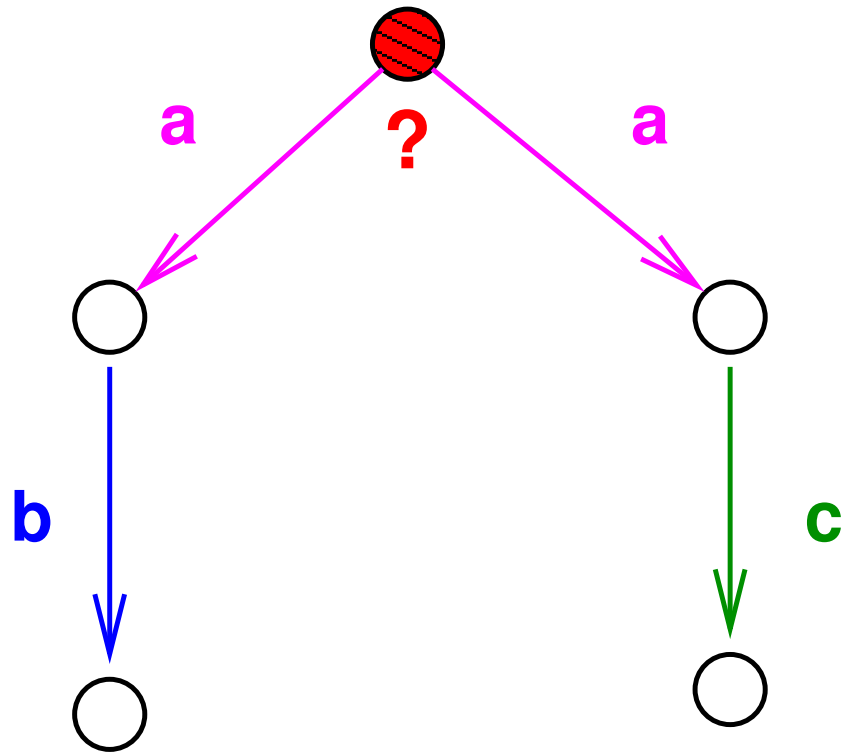
Playing a game



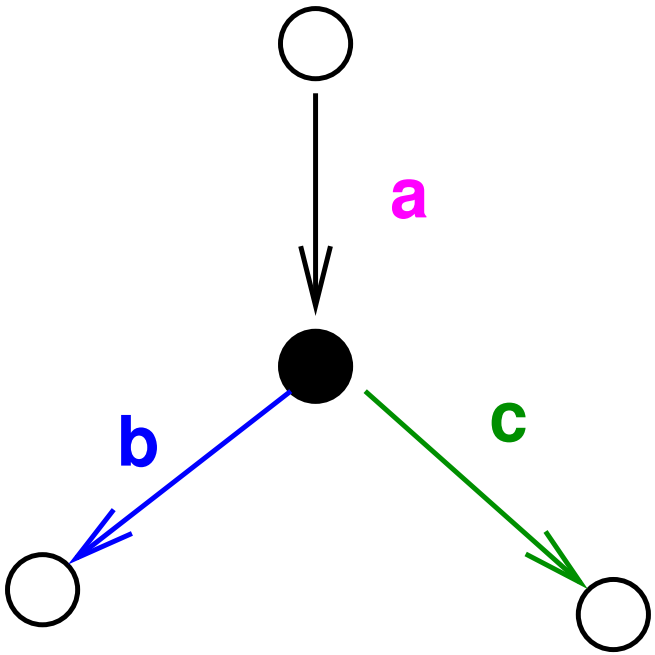


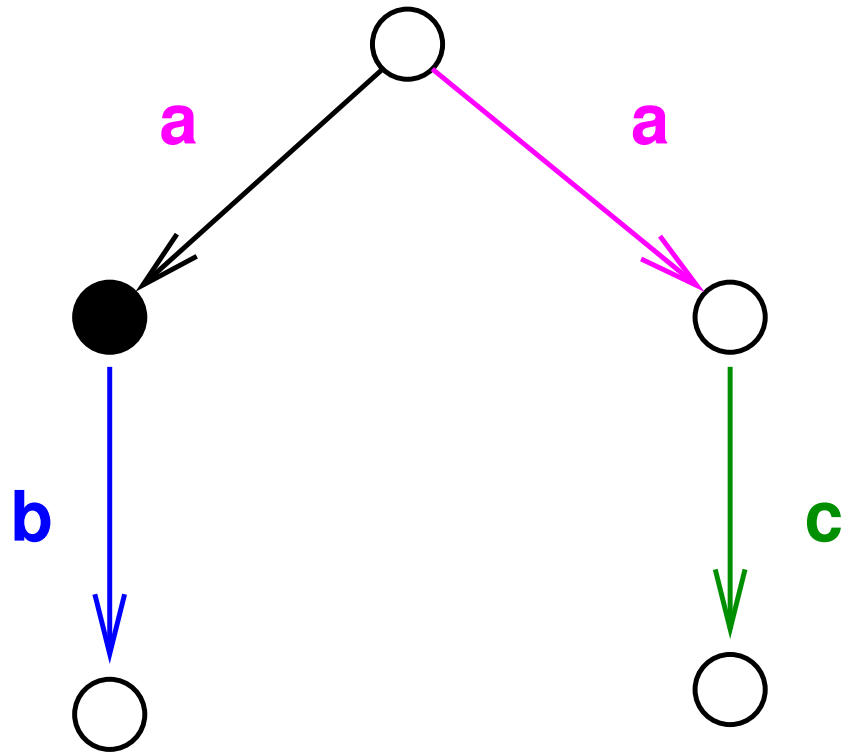
Playing a game



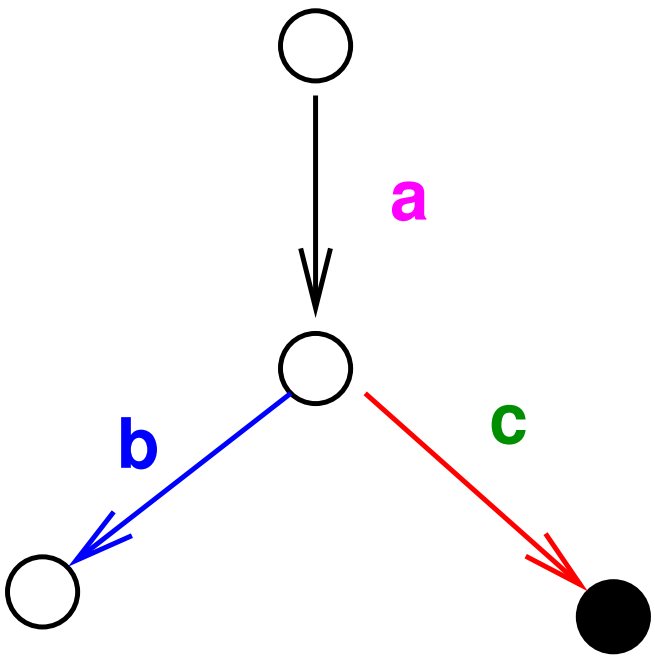


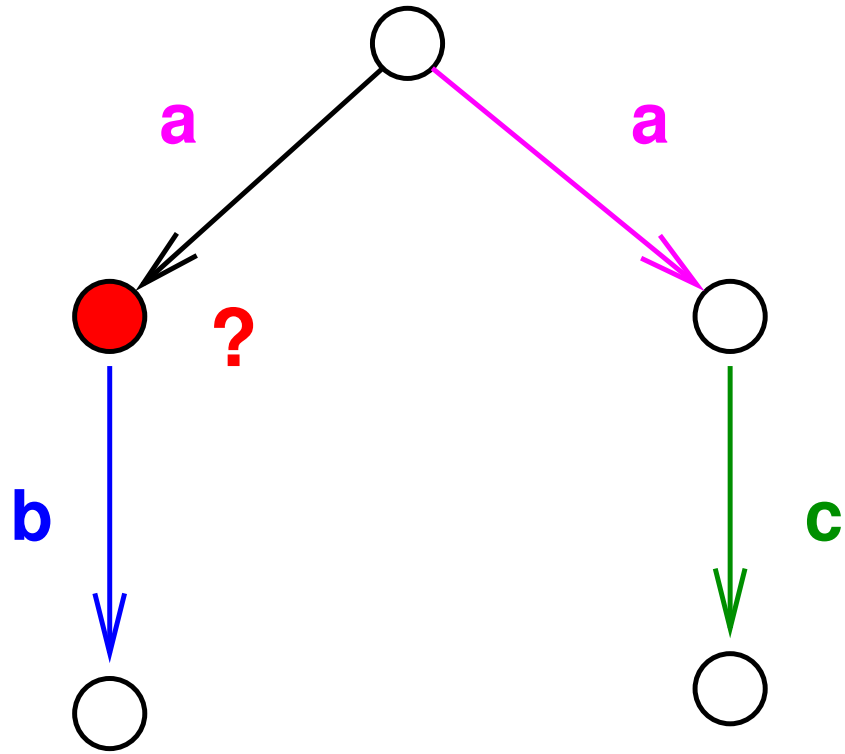
Playing a game





Playing a game





Examples

Different games can be defined to capture different verification questions. E.g.

bisimulation game:

- Refuter positions are pairs of processes
- Play alternates
- Moves: Refuter may choose any transition from either process; Verifier must choose a matching transition.
- Winning: Either player wins if the other can't go; Verifier wins all infinite plays.
- Interpretation: winning strategy for Verifier is a bisimulation.

Examples

Different games can be defined to capture different verification questions. E.g.

model-checking game:

- Positions: (process, formula) pairs.
- Play: Top combinator of formula determines whose turn it is - Verifier moves from $|$, $< >$, Refuter from &, [ ] etc.
- Moves: depend on combinator
- Winning: Either player wins if the other can't go; Verifier wins if formula True is reached, Refuter if False; winner of an infinite play is player who owns the outermost fixpoint unwound infinitely often
- Interpretation: a winning strategy for Verifier is a tableau.

Advantages of games

For users:

- Possibility of playing the game interactively for debugging

Advantages of games

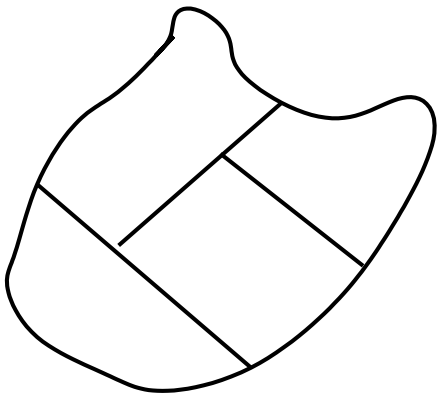
For users:

- Possibility of playing the game interactively for debugging

For developers/theoreticians:

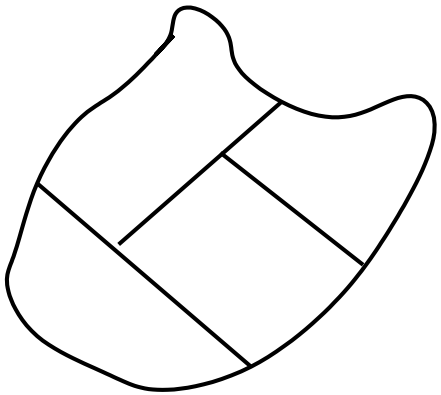
- Convenient way of thinking: first class object (the game) that corresponds to the problem being addressed
- Develop generic methods for finding winning strategies
- Allows refining only as needed, on the fly

Equivalence relations on positions

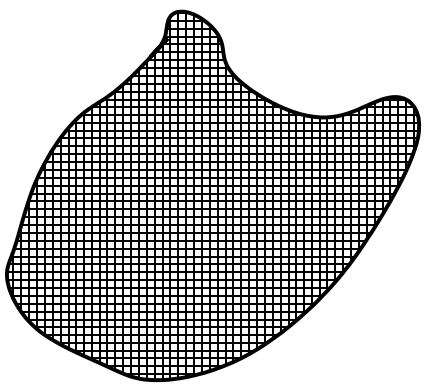


Shape
– too coarse, unsound

Equivalence relations on positions

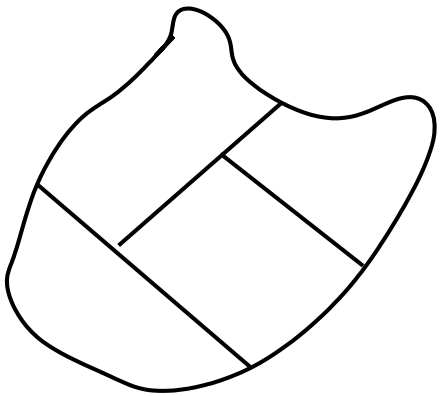


Shape
– too coarse, unsound

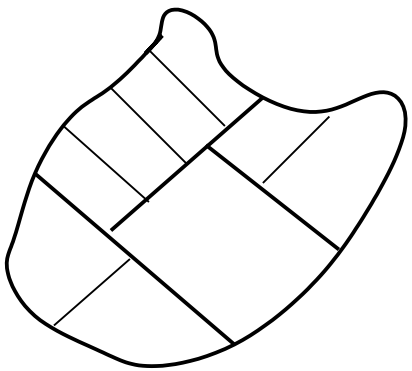


Shape + constraint that specifies values exactly
– too fine, intractable

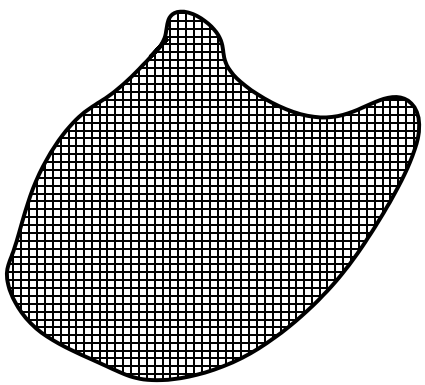
Equivalence relations on positions



Shape
– too coarse, unsound



sound + finite
– if this exists, we'll find it



Shape + constraint that specifies values exactly
– too fine, intractable

Relating to the CWB aims

I said:

> *We wanted a powerful, general way of understanding how to work with such systems, which on a practical level would also save effort in CWB development.*

We have a rather simple, general notion of abstraction of games, and results about when a winning strategy for one game induces one for another.

Based on this, we have an algorithm to search for winning strategies of games, and a theorem about when it is guaranteed to terminate.

Algorithm implemented in CWB.

Two examples: strong bisimulation, mu calculus model checking.

- The Edinburgh Concurrency Workbench
- Abstraction
- Games
- Implementation
- Conclusions

Separation of concerns

CWB provides:

- a generic strategy-finding functor which takes a description of a particular game as argument.
- a (primitive) language of values and constraints

Game description specifies:

- positions – appropriate notions of *shape* and *constraint*
- moves – functions for forward and backward exploration
- winning conditions – functions on shape lists saying who wins etc.

Types and manipulation of values

For now we provide:

- int
- bool
- user-defined ML style (recursive) datatypes, e.g.
  `datatype IntOpt = some of int | none;`

with a few pre-defined operations.

NB strategy-finding algorithm is cleanly separate. Would be nice to plug in a more expressive language!

First bisimulation example

First bisimulation example

Command: agent C = in(x:bool).'out(x).C;
Command: agent D = in(x:bool).'out(false).D;

First bisimulation example

Command: agent C = in(x:bool).'out(x).C;
Command: agent D = in(x:bool).'out(false).D;
Command: strongeq(C,D);

First bisimulation example

```
Command: agent C = in(x:bool).'out(x).C;
Command: agent D = in(x:bool).'out(false).D;
Command: strongeq(C,D);
false
```

First bisimulation example

```
Command: agent C = in(x:bool).'out(x).C;
Command: agent D = in(x:bool).'out(false).D;
Command: strongeq(C,D);
false
Command: strategybisim;
```

First bisimulation example: strategy

Player: Refuter (Abelard)
Position: AbelardToGo (C,D)
true
Move to: EloiseToGo ('out(VNew1 : bool).C,D,in(VX0 : bool),false)
&((VX0 : bool=LNew1 : bool),(LNew1 : bool=true))

Player: Refuter (Abelard)
Position: AbelardToGo ('out(VNew1 : bool).C,'out(false).D)
(LNew1 : bool=true)
Move to: EloiseToGo (C,'out(false).D,'out(VX0 : bool),false)
(VX0 : bool=true)

First bisimulation example: playing

First bisimulation example: playing

Command: playbisim;

First bisimulation example: playing

Command: playbisim;
Player Refuter (Abelard) has a winning strategy, so CWB will take that side!

First bisimulation example: playing

Command: playbisim;
Player Refuter (Abelard) has a winning strategy, so CWB will take that side!
CWB moves to EloiseToGo ('out(VNew1 : bool).C, D,in(VX0 : bool),false)
&((VX0 : bool=LNew1 : bool),(LNew1 : bool=true))

Your turn to move from [...]

1: AbelardToGo ('out(VNew1 : bool).C,'out(false).D)
(LNew1 : bool=true)

Pick a shape:

First bisimulation example: playing

Command: playbisim;
Player Refuter (Abelard) has a winning strategy, so CWB will take that side!
CWB moves to EloiseToGo ('out(VNew1 : bool).C, D,in(VX0 : bool),false)
&((VX0 : bool=LNew1 : bool),(LNew1 : bool=true))

Your turn to move from [...]

1: AbelardToGo ('out(VNew1 : bool).C,'out(false).D)
(LNew1 : bool=true)

Pick a shape:

1

First bisimulation example: playing

Command: playbisim;
Player Refuter (Abelard) has a winning strategy, so CWB will take that side!
CWB moves to EloiseToGo ('out(VNew1 : bool).C, D,in(VX0 : bool),false)
&((VX0 : bool=LNew1 : bool),(LNew1 : bool=true))

Your turn to move from [...]

1: AbelardToGo ('out(VNew1 : bool).C,'out(false).D)
(LNew1 : bool=true)

Pick a shape:

1

Now pick values for parameters satisfying constraint

LNew1 : bool must be true

CWB moves to EloiseToGo (C,'out(false).D,'out(VX0 : bool),false)
(VX0 : bool=true)

Your turn to move from [...]

But you have no legal moves from here, so YOU LOSE!

Comment on parameters vs specific values

Printout of strategy must involve parameters (e.g. show how to move from any position in infinite set, not just from one of them).

Comment on parameters vs specific values

Printout of strategy must involve parameters (e.g. show how to move from any position in infinite set, not just from one of them).

But when we play, we have choices.

Do players have to pick specific values (e.g. "input 23")?

or may they show off by picking constrained sets (e.g. "input anything except 17"?)

Comment on parameters vs specific values

Printout of strategy must involve parameters (e.g. show how to move from any position in infinite set, not just from one of them).

But when we play, we have choices.

Do players have to pick specific values (e.g. "input 23")?

or may they show off by picking constrained sets (e.g. "input anything except 17"?)

For now, user must always pick specific value, but CWB will only make the choices that matter. Maybe should offer both options?

Second bisimulation example

Second bisimulation example

Command: agent A = in(x:int).'out(x).A;
Command: agent B = in(y:int).('out(y).B + 'out(17).B);
Command: strongeq(A,B);

Second bisimulation example

```
Command: agent A = in(x:int).'out(x).A;
Command: agent B = in(y:int).('out(y).B + 'out(17).B);
Command: strongeq(A,B);
false
Command: strategybisim;
```

Second bisimulation example: strategy

Player: Refuter (Abelard)
Position: AbelardToGo (A,B)
true
Move to: EloiseToGo ('out(VNew1 : int).A,B,in(VX0 : int),false)
&(∼((VX0 : int=17)),(VX0 : int=LNew1 : int))

Player: Refuter (Abelard)
Position: AbelardToGo ('out(VNew1 : int).A,'out(VNew1 :
int).B+'out(17).B)
&(∼((LNew1 : int=17)),(RNew1 : int=LNew1 : int))
Move to: EloiseToGo ('out(VNew1 : int).A,B,'out(VX0 : int),true)
&(∼((VX0 : int=LNew1 : int)),(VX0 : int=17))

Modelchecking example

Modelchecking example

```
Command: agent B = in(y:int).('out(y).B + 'out(17).B);
Command: modelcheck (B,[in(3)]min(X0.[-]X0 | <'out(17)>T));
```

Modelchecking example

```
Command: agent B = in(y:int).('out(y).B + 'out(17).B);
Command: modelcheck (B,[in(3)]min(X0.[-]X0 | <'out(17)>T));

true
```

Modelchecking example

```
Command: agent B = in(y:int).('out(y).B + 'out(17).B);
Command: modelcheck (B,[in(3)]min(X0.[-]X0 | <'out(17)>T));

true
Command: playmc;
```

Modelchecking example

Command: agent B = in(y:int).('out(y).B + 'out(17).B);
Command: modelcheck (B,[in(3)]min(X0.[-]X0 | <'out(17)>T));

true
Command: playmc;

Player Verifier (Eloise) has a winning strategy, so CWB will take that side!

Modelchecking example

Your turn to move from (B,[in(3)]min(X0.[-]X0 | <'out(17)>T))
true

[but again you only have one choice, so moving swiftly on...]

Modelchecking example

Your turn to move from (B,[in(3)]min(X0.[-]X0 | <'out(17)>T))
true

[but again you only have one choice, so moving swiftly on...]

CWB moves to ('out(VNew1 : int).B+'out(17).B,[-]X0 | <'out(17)>T)
(VNew1 : int=3)

Modelchecking example

Your turn to move from (B,[in(3)]min(X0.[-]X0 | <'out(17)>T))
true

[but again you only have one choice, so moving swiftly on...]

CWB moves to ('out(VNew1 : int).B+'out(17).B,[-]X0 | <'out(17)>T)
(VNew1 : int=3)

CWB moves to ('out(VNew1 : int).B+'out(17).B,<'out(17)>T)
(VNew1 : int=3)

Modelchecking example

Your turn to move from (B,[in(3)]min(X0.[-]X0 | <'out(17)>T))
true

[but again you only have one choice, so moving swiftly on...]

CWB moves to ('out(VNew1 : int).B+'out(17).B,[-]X0 | <'out(17)>T)
(VNew1 : int=3)

CWB moves to ('out(VNew1 : int).B+'out(17).B,<'out(17)>T)
(VNew1 : int=3)

CWB moves to (B,T)
true

Modelchecking example

Your turn to move from (B,[in(3)]min(X0.[-]X0 | <'out(17)>T))
true

[but again you only have one choice, so moving swiftly on...]

CWB moves to ('out(VNew1 : int).B+'out(17).B,[-]X0 | <'out(17)>T)
(VNew1 : int=3)

CWB moves to ('out(VNew1 : int).B+'out(17).B,<'out(17)>T)
(VNew1 : int=3)

CWB moves to (B,T)
true

Your turn to move from (B,T)
true

Modelchecking example

Your turn to move from (B,[in(3)]min(X0.[-]X0 | <'out(17)>T))
true

[but again you only have one choice, so moving swiftly on...]

CWB moves to ('out(VNew1 : int).B+'out(17).B,[-]X0 | <'out(17)>T)
(VNew1 : int=3)

CWB moves to ('out(VNew1 : int).B+'out(17).B,<'out(17)>T)
(VNew1 : int=3)

CWB moves to (B,T)
true

Your turn to move from (B,T)
true

But you have no legal moves from here, so YOU LOSE!

Comment on infinite plays

Both the bisimulation and the modelchecking games may have infinite plays.

The strategy-finding algorithm of course uses careful inspection of repeats to terminate.

What should the playing function do?

Decision: we simply let play continue indefinitely.

Lessons from the implementation

Lessons from the implementation

- Syntax is hard :-([Yacc?!]

Lessons from the implementation

- Syntax is hard :-([Yacc?!]
- It really pays to think values and sets of values, not symbols

Lessons from the implementation

- Syntax is hard :-([Yacc?!]
- It really pays to think values and sets of values, not symbols
- Much as I hate ML :-), it's pretty good for this kind of thing.

Lessons from the implementation

- Syntax is hard :-([Yacc?!]
- It really pays to think values and sets of values, not symbols
- Much as I hate ML :-), it's pretty good for this kind of thing.
- It's ridiculous not to be using a constraint/SAT package...

- The Edinburgh Concurrency Workbench
- Abstraction
- Games
- Implementation
- Conclusions

Where next?

Two possible directions:

Where next?

Two possible directions:

1. See whether (or not) the abstract game approach can be made to stand up against state-of-the-art automatic abstraction techniques.

Where next?

Two possible directions:

1. See whether (or not) the abstract game approach can be made to stand up against state-of-the-art automatic abstraction techniques.
2. Combine verification-style games with ideas about *incrementally defined games for software design* (the talk I didn't give...!)

The talk I didn't give (1)

My student Jennifer Tenzer is about to submit a PhD thesis on *exploration games for software design with UML.*

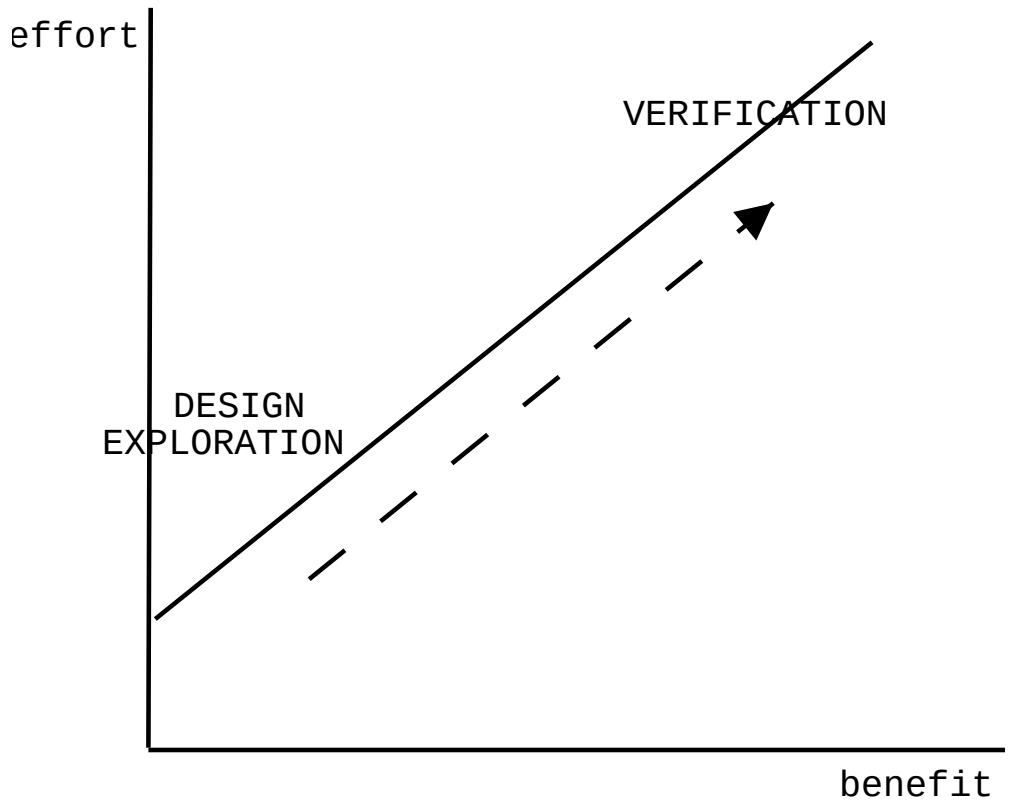
Basic idea:

1. start with a partial UML model of a system, and an incomplete, informal idea of its specification.
2. directly define a game based on the model and the specification:
   - moves are based on system's behaviour and "reasonable" environmental challenges
   - Refuter wins if the system misbehaves
3. allow user to modify the game's rules, refining the system model and the specification

Currently, tool's main role is as referee: only primitive attempt to find winning strategy.

The talk I didn't give (2)

Long term aim is to abolish the exploration/verification gap.



Connections with feature interaction

Pessimistically, maybe just an illustration of why FI is hard.

Focus of this work is to *make easy things easy*.

Maybe feature interaction is all about inherently hard things?

Would be very interested in collaborations to try to prove otherwise, though...

More info

http://homepages.inf.ed.ac.uk/perdita

More info

http://homepages.inf.ed.ac.uk/perdita

Questions? Comments? Offers of software?