Abstract Interpretations of Games

Perdita Stevens

Laboratory for Foundations of Computer Science
Division of Informatics
University of Edinburgh

Motivation

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.

Plan

- Previous approaches and shortcomings
- Games and set games
- Remaining shortcomings
- Abstract interpretation as a solution?
- Work in progress and remaining problems

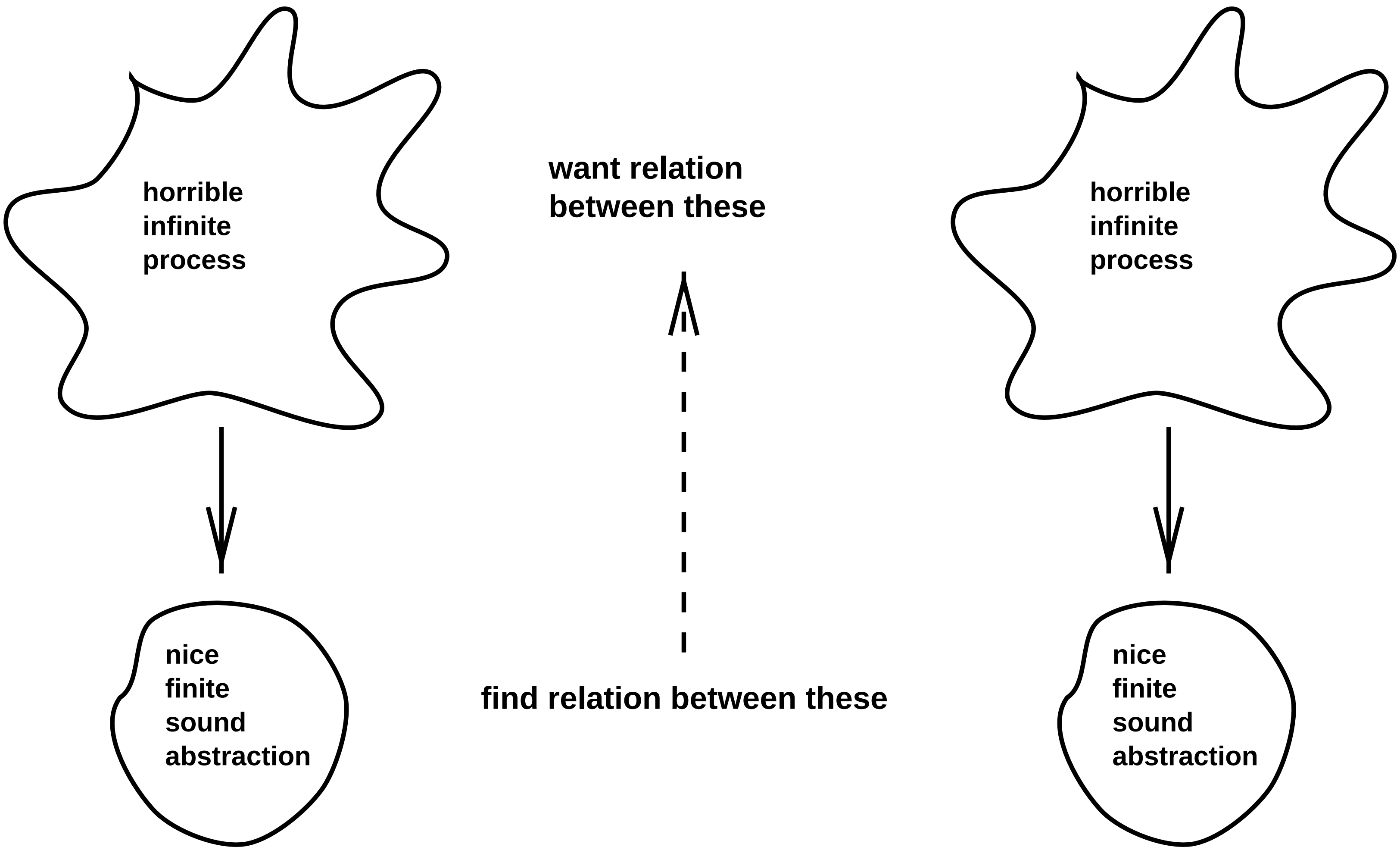
Starting points

Lots of work on such systems. Usual approach:

- Take a system and a class of questions of interest
- Define an abstraction of the system
- Prove that the abstraction gives the same answer as the original system on the class of questions
- Work with the abstraction

The most interesting general approach was that by Hennessy and others on symbolic transition graphs.

Typical approach to equivalence of infinite processes



Wires as plays of game

All the problems we're interested in can be seen as two-player games: one player ($\exists$loise) wants the answer to be Yes, the other ($\forall$belard) No.

A winning strategy for the game is a proof object demonstrating the answer.

Game is defined by set Pos of positions, starting position I , who moves from each position ($\lambda : \mathrm{Pos} \rightarrow \{\forall, \exists\}$), rules for legal moves ($\rightarrow$), and rules about who wins ($W_\forall$, $W_\exists$...). A play is a legal sequence of positions.

For example:

- bisimulation game: Abelard positions are pairs of processes: Abelard chooses a transition. Eloise must match from the other process. Eloise wins all infinite plays, and a winning strategy for her is a bisimulation.

- model-checking game: positions are (process, formula) pairs, winner of an infinite play is player who owns the outermost fixpoint unwound infinitely often, and a winning strategy is a tableau.

Bisimulation game

$\mathrm{B} = \mathrm{in}(x).\overline{\mathrm{out}}(x).\mathrm{B} \qquad ? \sim ? \qquad \mathrm{C} = \mathrm{in}(x).\overline{\mathrm{out}}(5).\mathrm{C}$

Abelard has a winning strategy: e.g. "choose $\mathrm{B} \xrightarrow{\mathrm{in}(2)} \overline{\mathrm{out}}(2).\mathrm{B}$, then follow your nose".

$$(\mathrm{B},\quad \mathrm{C})$$

$$\forall\downarrow$$

$$(\overline{\mathrm{out}}(2).\mathrm{B},\quad \mathrm{C},\quad \mathrm{in}(2),\quad 2)$$

$$\exists\downarrow$$

$$(\overline{\mathrm{out}}(2).\mathrm{B},\quad \overline{\mathrm{out}}(5).\mathrm{C})$$

$$\forall\downarrow$$

$$(\mathrm{B},\quad \overline{\mathrm{out}}(5).\mathrm{C},\quad \overline{\mathrm{out}}(2),\quad 2)$$

Now it's Eloise's turn, but she can't go, so Abelard wins.

Eloise has no better choices: no bisimulation can exist.

Bisimulation set game

Informally, you can play with sets of positions. For example:

$\{(\mathrm{B}, \quad \mathrm{C}) \quad : \quad \mathrm{true}\}$ in this case a singleton set

$\forall \downarrow$

$\{(\overline{\mathrm{out}}(\mathrm{p}).\mathrm{B}, \quad \mathrm{C}, \quad \mathrm{in}(\mathrm{p}), \quad 2) \quad : \quad \mathrm{p} \neq 5\}$ the crucial restriction

$\exists \downarrow$

$\{(\overline{\mathrm{out}}(\mathrm{q}).\mathrm{B}, \quad \overline{\mathrm{out}}(5).\mathrm{C}) \quad : \quad \mathrm{q} \text{ even}\}$ a pointless but legal restriction

$\forall \downarrow$

$\{(\mathrm{B}, \quad \overline{\mathrm{out}}(5).\mathrm{C}, \quad \overline{\mathrm{out}}(\mathrm{r}), \quad 2) \quad : \quad \mathrm{r} \text{ even}\}$

Now it’s Eloise’s turn, but again she can’t go, so Abelard wins.

Details in a minute!

Use of the set game

In the CONCUR paper “Abstract games for infinite state processes” I showed

- (the intuitively clear fact) that the same player has a winning strategy for the set game as for the original concrete game
- that strategies translate nicely (useful for debugging)
- that a certain algorithm finds winning strategies if it terminates
- that it does terminate under certain conditions

This enabled me to recapture some known decidability results, and to show how the CWB could implement a single strategy-finding algorithm that could easily be instantiated to solve a very wide variety of problems.

So far, so good...

Remaining shortcomings

But two things showed this wasn't the whole story:

- People asked what the relationship was with abstract interpretation
- The CONCUR referees rightly commented that it was hard to follow and that there must be a better way of presenting it.

Intuitively it seemed that the algorithm was constructing an equivalence relation on the set of concrete positions, and hence building

a finite, but detailed enough, abstract game

but this wasn't clearly developed.

So next I tried to make the connection with abstract interpretation explicitly, in the hope that this would lay bare what was going on.

Some progress, and some possibly interesting observations – but help needed!

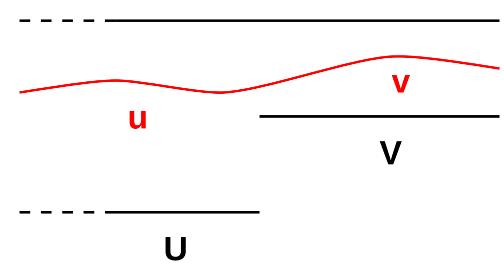
Abstract interpretations of games

Start with a concrete game $G^{\mathcal{C}}$ characterising the problem.

Suppose we have (as deus ex machina) a p.o. $(\mathrm{Pos}^{\mathcal{A}}, \leq)$ and abstraction map $h : \mathrm{Pos}^{\mathcal{C}} \longrightarrow \mathrm{Pos}^{\mathcal{A}}$. Write $u \sim v$ for $hu = hv$. For sense, require h to be such that pointwise equivalent infinite plays are won by the same player.

Wlo(useful?)g, $\mathrm{Pos}^{\mathcal{A}}$ consists of non-empty subsets of $\mathrm{Pos}^{\mathcal{C}}$ from which the same player is to move, and $\leq$ is $\subseteq$.

Extend to abstract game $G^{\mathcal{A}}$:



- $I^{\mathcal{A}} = h(I^{\mathcal{C}})$
- $\lambda^{\mathcal{A}}(U) = \lambda^{\mathcal{C}}(u)$, any u with $hu \leq U$.
- $U \rightarrow_{\mathcal{A}} V$ iff $\forall v. hv \leq V\ \exists u. hu \leq U$ and $u \rightarrow_{\mathcal{C}} v$ (and $u \rightarrow_{\mathcal{C}}$ *something* if $hu \leq U$).

  For sense, require h to be s.t. $u \rightarrow_{\mathcal{C}} v \Rightarrow hu \rightarrow_{\mathcal{A}} hv$. Extend h and $\leq$ pointwise to plays. Abstract play P subsumes p if $hp \leq P$.
- You win an infinite abstract play P iff P subsumes some concrete play won by you and no concrete play won by the other player.

For what h is $G^{\mathcal{A}}$ detailed enough?

The following conditions on h are enough for a strategy transfer theorem: i.e. to ensure that we may soundly play $G^{\mathcal{A}}$ instead of $G^{\mathcal{C}}$:

1. $u \to_{\mathcal{C}} v \sim v' \Rightarrow \exists u' \sim u \,.\, u' \to v'$
2. $u' \sim u \to_{\mathcal{C}} v \Rightarrow \exists v' \sim v \,.\, u' \to v'$

$$\begin{array}{ccc} u & \longrightarrow & v \\ \wr & & \wr \\ u' & \longrightarrow & v' \end{array}$$

The original set game, the $G^{\mathcal{A}}$ got from

$$h : u \mapsto \{u\}$$

is certainly detailed enough: but it's even more complex than $G^{\mathcal{C}}$!

The algorithm starts with an h_0 ("shape") which is finite, but not detailed enough. It *tries* to construct some refinement which is both finite and detailed enough.

Little Red Workbench and the Undecidability Wolf

Observation from CWB: decidability isn't very interesting for tool builders.

Giving the answer never is in no way worse than giving it in 7 years' time or after using 7TB of memory.

(And many users regard undecidability as no excuse!)

Picture courtesy of *The Little Red Riding Hood Project* editor Michael N. Salda: *The de Grummond Children's Literature Research Collection, University of Southern Mississippi* `http://www-dept.usm.edu/ engdept/lrrh/lrrhhome.htm`

Abstract interpretation of *what*?

a.i. normally seems to talk about a.i. of a *program*.

But here we have no distinguished program.

We may have two systems to compare – neither is THE program.

For model-checking, a.i. approaches have normally made the system the program and talked about a.i. of that, leaving the formula alone. But this doesn't fit well with powerful logics, where formulae may also need to be abstracted.

It looks to me as though a.i. really abstracts a *problem*, and it's just a historical accident that the problems considered have normally concerned a program.

But I'm a stranger here. Does that make sense?

Work in progress and outstanding problems

In progress:

- Implementation and experimentation
- Applications: for example, observational mu-calculus as “assembly language” logic for timed/value-passing logics, model-checked by abstract games. (Joint with with Julian Bradfield: preliminary paper in Fixed Points in Computer Science 98.)

Areas where I’d really like help from a.i. experts:

- What exactly is the relationship with ideas of widening and narrowing?
- Can a better understanding of the relationship with a.i. lead to better algorithms, and/or better proofs of correctness?
- Is there any useful notion of approximation or partial answer?