Verified Synthesis of Knowledge-Based Programs in Finite Synchronous Environments

P. Gammie

School of Computer Science
Australian National University

National ICT Australia (summer job)

Interactive Theorem Proving
Second International Conference
2011

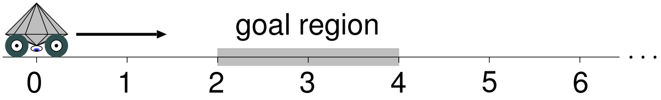
Motivation: Knowledge in Distributed Systems

Protocols in distributed systems are often presented informally by talking about who knows what when:

> *... When the writer knows (via the acknowledgements and its knowledge of its own local behaviour) that a majority of the processes have their tag values equal to t, it returns ack. ...*
>
> — Nancy Lynch (1996)

Idea: write specifications using formal knowledge conditionals.

Stock example: the autonomous robot



(Kai Engelhardt, tgif on Linux, 2001.)

A robot travels along the natural number line.

- It starts at 0 and aims to halt in the goal region $\{2, 3, 4\}$.
- The environment tries to move it to the right 0 or 1 positions at each time step.
- The robot has a sensor that reads the current position $\pm$ 1.
- It can pull the brake instantaneously.

What program should the robot execute?

The robot: knowledge-based program (KBP)

```
do
      [] $\mathbf{K}_{\text{robot}}$ goal        $\rightarrow$ Halt
      [] $\neg\mathbf{K}_{\text{robot}}$ goal       $\rightarrow$ Nothing
od
```

... and a suitable definition of the environment ...

Outline

1. Semantics
2. Finite-state implementations of KBPs
3. Formalisation in Isabelle/HOL

The Logic of Knowledge

Standard multi-modal account of knowledge using Kripke structures:

- propositional
- modality $\mathbf{K}_a$ for each agent a.
- Standard $S5_n$ models: one equivalence relation per agent.

Guard syntax

For some sets of propositions $\mathcal{P}$ and agents $\mathcal{A}$:

$$\phi ::= \mathcal{P} \mid \phi \wedge \phi \mid \neg \phi \mid \mathbf{K}_{\mathcal{A}} \phi$$

Semantics

Given a Kripke structure M consisting of:

- a set of worlds $\mathcal{W}$
- one equivalence relation $\sim_a \subseteq \mathcal{W} \times \mathcal{W}$ per agent *a*
- a valuation function $\mathcal{V} : \mathcal{P} \Rightarrow 2^{\mathcal{W}}$

define satisfaction at a world $w \in \mathcal{W}$:

$$\begin{array}{ll}
\mathrm{M}, w \models p & \text{iff } w \in \mathcal{V}p \\
\mathrm{M}, w \models \neg\phi & \text{iff } \mathrm{M}, w \not\models \phi \\
\mathrm{M}, w \models \phi \wedge \psi & \text{iff } \mathrm{M}, w \models \phi \text{ and } \mathrm{M}, w \models \psi \\
\mathrm{M}, w \models \mathbf{K}_a\phi & \text{iff } \forall w' \in \mathcal{W}. w \sim_a w' \longrightarrow \mathrm{M}, w' \models \phi
\end{array}$$

Interpreted systems

The following is a streamlined version of the story told by Fagin, Halpern, Moses, and Vardi (1995).

A KBP is a set of guarded commands, a subset of $\phi \times aAct$.

A *finite-state environment* consists of:

- a finite set of (global, instantaneous) states $\mathcal{S}$
- some way of evaluating propositions $\mathcal{V} : \mathcal{P} \Rightarrow 2^{\mathcal{S}}$
- some set of initial states init : $2^{\mathcal{S}}$
- the environment's action envAct : $\mathcal{S} \Rightarrow 2^{eAct}$
- a global transition function:

$$s \xrightarrow[\text{aActs}]{\text{eAct}} s' : eAct \Rightarrow (\mathcal{A} \Rightarrow aAct) \Rightarrow \mathcal{S} \Rightarrow \mathcal{S}$$

Equivalence classes over traces

Assume there is an observation function $\mathrm{O}_a : \mathcal{S} \Rightarrow \mathcal{O}$ for each agent a.

We hoist these to traces $t, t' \in \mathcal{S}$ list:

The synchronous perfect-recall (SPR) semantics for knowledge

$$t \sim_a t' \equiv \text{map } \mathrm{O}_a\ t = \text{map } \mathrm{O}_a\ t'$$

Therefore all agents can tell the time:

Synchrony

$$t \sim_a t' \Longrightarrow \text{length } t = \text{length } t'$$

Synchrony supports an inductive construction

The set of traces of the system of length *n* are:

$$\begin{aligned}
\mathrm{jkbpC}_0 &= \mathrm{init} \\
\mathrm{jkbpC}_{n+1} &= \{t \xrightarrow[\mathrm{aActs}]{\mathrm{eAct}} s \mid t \in \mathrm{jkbpC}_n \\
&\qquad \wedge \mathrm{eAct} \in \mathrm{envAct}\ (\mathrm{last}\ t) \\
&\qquad \wedge \forall a \in \mathcal{A}.\ \mathrm{aActs}\ a \in \{act \mid (\phi, act) \in KBP_a \wedge \mathrm{jkbpC}_n, t \models \phi\}\}
\end{aligned}$$

This yields a canonical set:

$$\mathrm{jkbpC} = \bigcup_n \mathrm{jkbpC}_n$$

Finite-state implementations of KBPs

We really want automata that *implement* the KBPs, i.e. enable the same actions in the same knowledge states.

$\rightarrow$ As we will see, infinite-state implementations always exist.

It is natural to ask when there are finite-state implementations.

$\rightarrow$ van der Meyden (1996) showed that this is only sometimes.

Therefore we only treat special cases, such as:

- when there is only a single agent
- when agents communicate only by broadcast

$\rightarrow$ Details in the paper / AFP.

I refined the existence theorems into an algorithmic story.

Implementations

A *Moore automaton* with states of type $\mathcal{X}$ consists of:

- **Initial states:** $\mathcal{O} \to \mathcal{X}$;
- **Transitions:** $\mathcal{O} \to \mathcal{X} \to \mathcal{X}$
- **Actions:** $\mathcal{X} \to aAct$ list

Canonical infinite-state implementations always exist; for agent a:

- **States:** The partition of jkbpC under $\sim_a$.
- **Transitions:** For $X \in \mathcal{X}$, transition on $o \in \mathcal{O}$ to state

$$\{t \rightsquigarrow s \in \mathrm{jkbpC} \mid t \in X \wedge \mathrm{O}_a(s) = o\}$$

- **Actions:** Choose $t \in \mathcal{X}$ and calculate

$$\{act \mid (\phi, act) \in KBP_a \wedge \mathrm{jkbpC}, t \models \phi\}$$

Key technical machinery: simulations

van der Meyden showed that if we can *simulate* the Kripke structure corresponding to jkbpC by a finite Kripke structure, then we have our finite-state implementations.

Simulations

A simulation is a map f from the worlds of M to the worlds of M' s.t.:

- For all propositions p: $u \in \mathcal{V}_M\ p$ iff $f\ u \in \mathcal{V}_{M'}\ p$
- If $u \sim_a v$ in M then $f\ u \sim_a f\ v$ in M'
- If $f\ u \sim_a v'$ in M' then there is a v s.t. $f\ v = v'$ and $u \sim_a v$ in M

It follows that $M, w \models \phi$ iff $M', f\ w \models \phi$.

There are simulations for the two special cases mentioned earlier.

$\rightarrow$ Details in the paper / AFP.

I showed how an automaton for agent *a* can be constructed using only the simulated equivalence classes for *a*.

$\rightarrow$ Details in the paper / AFP.

Key Problem: the action function looks like this:

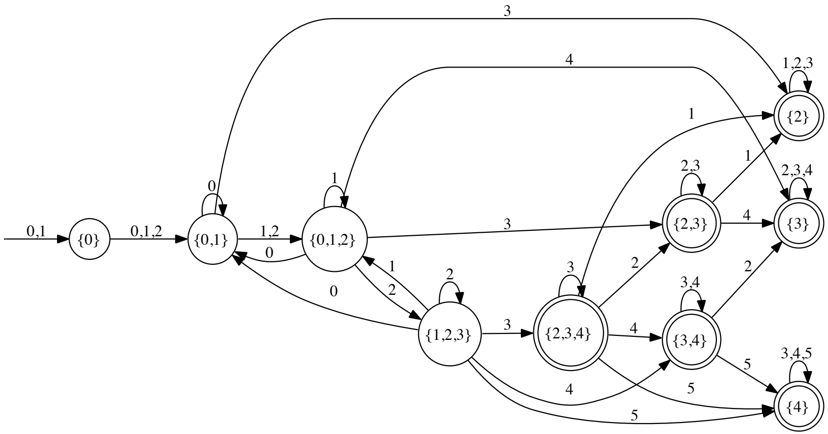
$$\{act \mid (\phi, act) \in KBP_a \wedge \text{jkbpC}_{(\text{length } t)}, t \models \phi\}$$

How do we compute this? $\text{jkbpC}_{(\text{length } t)}$ might be huge.

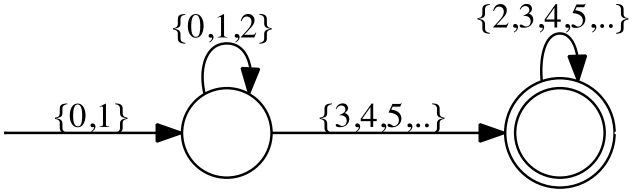
For the simulations we have, the simulated traces contain enough information already. This is a bit surprising.

The algorithm itself is a depth-first search and is similar to the subset construction for determinising NDFAs.

An implementation of the robot



... but this is what a human would have designed:



The bigger picture

"Broadcast environments" also go by the name of (the logic of) public announcements.

- However the framework presented here is more expressive: it can deal with a changing world.

A variant of this semantics has been used to treat KBPs for "snoopy cache protocols" manually.

- This is a multi-agent scenario.

$\rightarrow$ So don't be fooled by the triviality of the robot!

Formalisation in Isabelle/HOL

The mechanisation effort proceeded much as I presented it here.

We show once-and-for-all that:

- Canonical interpretations (jkbpC) exist
- Canonical automata exist

and assuming simulated operations exist:

- Simulated automata exist
- The generic algorithm produces simulated automata.

Then for each special case we show that the corresponding simulation and simulated operations do the trick.

Finally we instantiate the entire framework on two examples, and Isabelle's code generator produces automata for us.

Locales for context

Ultimately the algorithm is parametrised by the environment, the KBPs and the simulated operations.

Isabelle's locales are a perfect fit for this. For example:

```
locale Environment =
 fixes jkbp :: "'a ⇒ ('a, 'p, 'aAct) KBP"
  and envInit :: "('s :: finite) list"
  and envAction :: "'s ⇒ 'eAct list"
  and envTrans :: "'eAct ⇒ ('a ⇒ 'aAct) ⇒ 's ⇒ 's"
  and envVal :: "'s ⇒ 'p ⇒ bool"
  and envObs :: "'a ⇒ 's ⇒ 'obs"
 assumes subj: "∀ a gc. gc ∈ set (jkbp a) ⟶ subjective a (guard gc)"
```

This worked quite well, apart from the repetition due to instantiating types and the code generator.

Top-down derivations

Showing that an algorithm works “once and for-all” (with all these parameters) in a classical top-down manner seems to necessitate data refinement.

What I refined was a matter of necessity; the end product is good enough for today but maybe not tomorrow.

It also involved some cutting-and-pasting of existing theories such as the Isabelle Collections Framework of Lammich and Lochbihler (2010).

Isabelle’s simplifier is much less useful than usual as it struggles with “abstract” quotients. In contrast the recent work of Kaliszyk and Urban (2011) deals with “concrete” quotients.

Thanks!