Model Checking from a Type Theoretic Perspective

Karim Kanso
Anton Setzer

Swansea University, Wales, UK

April 15^{th}, 2010

This work is funded by: Invensys Rail, Chippenham, UK

Motivation

Compose Interactive and Automatic theorem proving techniques

Motivation

Compose Interactive and Automatic theorem proving techniques

Theorem proving can be a complicated task.

- Interactive provers guide and check proofs,
  - Good for proving abstract/generic theorems.
- Automatic provers solve problems,
  - Typically, simple but large problem sets.
    - i.e. Industrial verification
  - Good for verifying finite concrete theorems.

This project is concerned with not only verification but also producing correct software; for this, Agda is used.

Talk Outline

- About Agda,
- Embedding Automated Theorem Provers,
- Model Checking,
  - CTL

Agda and its Dependents

Agda2[1] is a:

- dependently typed functional programming language, and
- proof assistant.

Based on intuitionistic type theory developed by the Swedish logician Martin-Löf.

Belongs to a family of tools the first of which, Alf (1992), followed by: Half, CHalf, Agda and Alfa.

Ulf Norell at Chalmers started Agda2 in 2007.

Agda has many similarities with other proof assistants based on dependent types, such as Coq, Epigram, Matita and NuPRL.

[1] See: http://wiki.portal.chalmers.se/agda/

Dependent Type Examples

Natural Numbers

```
data ℕ : Set where
  zero : ℕ
  suc : ℕ → ℕ
```

Vectors of type A of length n

```
data Vec (A : Set) : ℕ → Set where
  [] : Vec A zero
  _::_ : {n : ℕ} → A → Vec A n → Vec A (suc n)
```

Existential quantifier

```
data ∃ (A : Set) (P : A → Set) : Set where
  _,_ : (x : A) (y : P x) → ∃ A P
```

Embedding Automated Theorem Provers in Agda

A generic approach is applied to embedding theorem provers:

1. Define (in Agda)
   - What it means for a formula to hold,

     $$\mathcal{M}, E \models \varphi$$

   - Simple decision procedure

     $$D_{\mathcal{M},E} : Formula \rightarrow Boolean$$

2. Prove (in Agda)
   - Correctness

     $$\forall \mathcal{M}\ \forall E\ \forall \varphi \quad D_{\mathcal{M},E}(\varphi) \Leftrightarrow \mathcal{M}, E \models \varphi$$

3. Replace D by actual call to automated theorem prover.

Where M is a model, E is an environment and φ is a formula.

Approach: Reflection

Evaluation of $D_{\mathcal{M},E}(\varphi)$ proceeds in one of two ways:

1. $D_{\mathcal{M},E}(\varphi)$ is a closed term,
   - Theorem prover will be executed efficiently, and
   - Should the prover return *true*, Agda gets a proof of

$$\mathcal{M}, E \models \varphi$$

2. $D_{\mathcal{M},E}(\varphi)$ has holes,
   - Agda attempts partial evaluation of $\mathcal{M}, E \models \varphi$
   - using the inbuilt inefficient decision procedure $D_{\mathcal{M},E}$.

This method gives Agda a proof of tautologies.

Approach: Reflection

$D_{\mathcal{M},E}$ is defined naïvely, thus simplifying correctness proofs.

Already implemented an embedding of SAT into type theory, [AVoCS'09]. The interface to Agda was by an ad hock plug-in.

For a case study, our sponsor provided industrial verification problems from the railway industry.

This architecture will be used to implement CTL model checking.

Model Checking

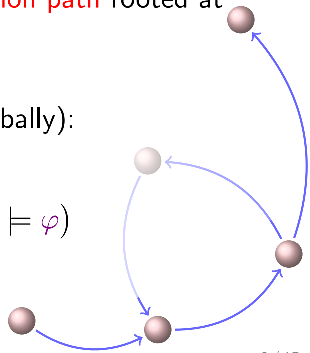
This project is concerned with CTL model checking FSM,

- using combined operators, i.e. EX and EG.
- As defined by Huth and Ryan: Logic in Computer Science.

CTL model checking is essentially determining whether some property φ holds for all/some infinite computation path rooted at some state s.

Consider the proof obligation for EG (exists globally):

$$\mathcal{M}, s_0 \models \mathrm{EG}\,\varphi \Leftrightarrow \exists\langle s_0 \to s_1 \to \ldots\rangle \quad \forall i\big(\mathcal{M}, s_i \models \varphi\big)$$



CTL: Infinite Paths

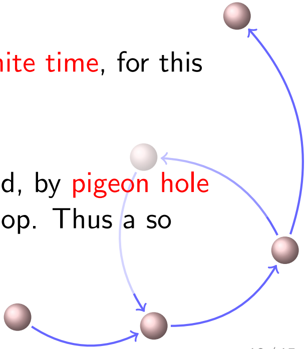
Consider the proof obligation for EG (exists globally):

$$\mathcal{M}, s_0 \models \mathrm{EG}\,\varphi \Leftrightarrow$$
$$\exists \langle s_0 \rightarrow s_1 \rightarrow \ldots \rangle \quad \forall i \left(\mathcal{M}, s_i \models \varphi \right)$$

There exists an infinite path rooted at state s_0 such that property φ always holds.

Checking all infinite paths cannot be done in finite time, for this reason $D_{\mathcal{M},s}$ relies upon checking finite paths.

Only state machines with n states are considered, by pigeon hole principle any path longer than n must have a loop. Thus a so called lasso can be constructed.



Pigeon Hole Principle

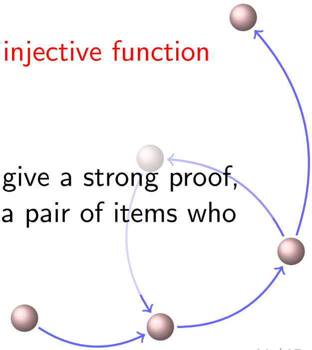
Putting n items into m holes, with $n > m$.

$$\Downarrow$$

At least one hole contains more than one item.

Proving the above amounts to proving that an injective function $f : n \rightarrow m$ does not exist, w.r.t. finite sets.

In the case of splitting a path, it is required to give a strong proof, such that a counter example is computed. I.e. a pair of items who share a hole.



EG: Checking $4 \Rightarrow 1$

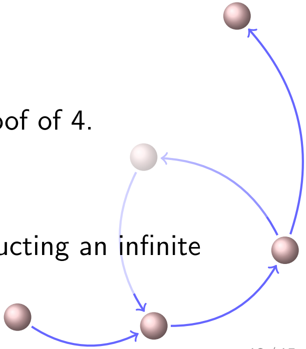
In the case of EG the following are equivalent:

1. $\mathcal{M}, s_0 \models \mathrm{EG}\,\varphi$
2. $\exists \langle s_0 \to s_1 \to \ldots \rangle \quad \forall i \big(\mathcal{M}, s_i \models \varphi\big)$
3. $\exists \langle s_0 \to s_1 \to \ldots s_k \rangle$
   $\exists \langle s_k \to s_{k+1} \to \ldots \to s_{k+m} \to s_k \rangle$
   $\forall i \leq k + m \quad \mathcal{M}, s_i \models \varphi$
4. $\exists \langle s_0 \to s_1 \to \ldots s_n \rangle \quad \forall i \leq n \quad \mathcal{M}, s_i \models \varphi$

The inbuilt decision procedure $D^{\mathrm{EG}}_{\mathcal{M},s}$ gives a proof of 4.

$4 \Rightarrow 3$ by php, a lasso can be constructed.

$3 \Rightarrow 2$, by means of canonical unfolding, constructing an infinite path represented by an element of a co-algebra.



EG: Infinite Path $\Rightarrow$ Lasso $\quad 2 \Rightarrow 3$

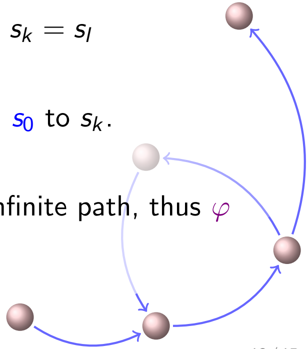
2. $\exists\langle s_0 \rightarrow s_1 \rightarrow \ldots\rangle \quad \forall i\big(\mathcal{M}, s_i \models \varphi\big)$

3. $\exists\langle s_0 \rightarrow s_1 \rightarrow \ldots s_k\rangle$
   $\exists\langle s_k \rightarrow s_{k+1} \rightarrow \ldots \rightarrow s_{k+m} \rightarrow s_k\rangle$
   $\forall i \leq k+m \quad \mathcal{M}, s_i \models \varphi$

Only n states in $\mathcal{M}$, thus $\exists k < l \leq n$ such that $s_k = s_l$

Therefore, a loop exists on s_k and a prefix from s_0 to s_k.

Both the loop and prefix are sub paths of the infinite path, thus φ holds along both of these paths.



Current Progress

- SAT has been formalised and correctness proven in Agda, and
- CTL has been formalised in Agda, and
- Correctness has been proven for all but the EU case, and
- Much work has been done modelling the case study.

Next Step

Implement generic plug-in mechanism for Agda.

Conclusion

Our technique has the following advantages:

Theorem provers integrated into development environment allows assigning to programs a type, which guarantees that every element of this type is a correct program w.r.t. some property.

Abstract and concrete properties can be verified. I.e.

$\forall x\ \varphi(x)$ holds and

For a fixed y $\varphi(y)$ holds

Potentially, allow for model of software to be compiled and simulated. "Virtual sand boxing" / "Rapid prototyping"