Some properties of CTL

Gertrud Bauer

May 13, 2003

Contents

theory *CTL* = *Main*:

1 CTL formulae

We formalize basic concepts of Computational Tree Logic (CTL) [2, 1] within the simply-typed set theory of HOL.

By using the common technique of "shallow embedding", a CTL formula is identified with the corresponding set of states where it holds. Consequently, CTL operations such as negation, conjunction, disjunction simply become complement, intersection, union of sets. We only require a separate operation for implication, as point-wise inclusion is usually not encountered in plain set-theory.

lemmas [*intro*!] = *Int-greatest Un-upper2 Un-upper1 Int-lower1 Int-lower2*

types *'a ctl* = *'a set*
constdefs
imp :: *'a ctl* ⇒ *'a ctl* ⇒ *'a ctl* (**infixr** → *75*)
p → *q* ≡ − *p* ∪ *q*

lemma [*intro*!]: *p* ∩ *p* → *q* ⊆ *q* **by** (*unfold imp-def*) *auto*
lemma [*intro*!]: *p* ⊆ (*q* → *p*) **by** (*unfold imp-def*) *rule*

The CTL path operators are more interesting; they are based on an arbitrary, but fixed model $\mathcal{M}$, which is simply a transition relation over states $'a$.

consts *model* :: $('a \times 'a)$ *set* $\quad (\mathcal{M})$

The operators EX, EF, EG are taken as primitives, while AX, AF, AG are defined as derived ones. The formula EX p holds in a state s, iff there is a successor state s' (with respect to the model $\mathcal{M}$), such that p holds in s'. The formula EF p holds in a state s, iff there is a path in $\mathcal{M}$, starting from s, such that there exists a state s' on the path, such that p holds in s'. The formula EG p holds in a state s, iff there is a path, starting from s, such that for all states s' on the path, p holds in s'. It is easy to see that EF p and EG p may be expressed using least and greatest fixed points [2].

constdefs
EX :: $'a$ *ctl* $\Rightarrow$ $'a$ *ctl* (EX - [80] 90) EX $p \equiv \{s.\ \exists s'.\ (s, s') \in \mathcal{M} \wedge s' \in p\}$
EF :: $'a$ *ctl* $\Rightarrow$ $'a$ *ctl* (EF - [80] 90) EF $p \equiv$ *lfp* $(\lambda s.\ p \cup \text{EX}\ s)$
EG :: $'a$ *ctl* $\Rightarrow$ $'a$ *ctl* (EG - [80] 90) EG $p \equiv$ *gfp* $(\lambda s.\ p \cap \text{EX}\ s)$

AX, AF and AG are now defined dually in terms of EX, EF and EG.

constdefs
AX :: $'a$ *ctl* $\Rightarrow$ $'a$ *ctl* (AX - [80] 90) AX $p \equiv -\ \text{EX} -p$
AF :: $'a$ *ctl* $\Rightarrow$ $'a$ *ctl* (AF - [80] 90) AF $p \equiv -\ \text{EG} -p$
AG :: $'a$ *ctl* $\Rightarrow$ $'a$ *ctl* (AG - [80] 90) AG $p \equiv -\ \text{EF} -p$

lemmas [*simp*] = *EX-def EG-def AX-def EF-def AF-def AG-def*

2 Basic fixed point properties

First of all, we use the de-Morgan property of fixed points

lemma *lfp-gfp*: *lfp* $f = -$ *gfp* $(\lambda s\ .\ -\ (f\ (-\ s)))$
proof
show *lfp* $f \subseteq -$ *gfp* $(\lambda s.\ -\ f\ (-\ s))$
proof
fix x **assume** *l*: $x \in$ *lfp* f
show $x \in -$ *gfp* $(\lambda s.\ -\ f\ (-\ s))$
proof
assume $x \in$ *gfp* $(\lambda s.\ -\ f\ (-\ s))$
then obtain u **where** $x \in u$ **and** $u \subseteq -\ f\ (-\ u)$ **by** (*unfold gfp-def*) *auto*
then have $f\ (-\ u) \subseteq -\ u$ **by** *auto*
then have *lfp* $f \subseteq -\ u$ **by** (*rule lfp-lowerbound*)
from *l* **and** *this* **have** $x \notin u$ **by** *auto*
then show *False* **by** *contradiction*
qed
qed
show $-$ *gfp* $(\lambda s.\ -\ f\ (-\ s)) \subseteq$ *lfp* f
proof (*rule lfp-greatest*)

```
    fix u assume f u ⊆ u
    then have − u ⊆ − f u by auto
    then have − u ⊆ − f (− (− u)) by simp
    then have − u ⊆ gfp (λs. − f (− s)) by (rule gfp-upperbound)
    then show − gfp (λs. − f (− s)) ⊆ u by auto
  qed
qed

lemma lfp-gfp': − lfp f = gfp (λs. − (f (− s)))
  by (simp add: lfp-gfp)

lemma gfp-lfp': − gfp f = lfp (λs. − (f (− s)))
  by (simp add: lfp-gfp)
```

in order to give dual fixed point representations of AF p and AG p:

```
lemma AF-lfp: AF p = lfp (λs. p ∪ AX s) by (simp add: lfp-gfp)
lemma AG-gfp: AG p = gfp (λs. p ∩ AX s) by (simp add: lfp-gfp)

lemma EF-fp: EF p = p ∪ EX EF p
proof −
  have mono (λs. p ∪ EX s) by rule (auto simp add: EX-def)
  then show ?thesis by (simp only: EF-def) (rule lfp-unfold)
qed

lemma AF-fp: AF p = p ∪ AX AF p
proof −
  have mono (λs. p ∪ AX s) by rule (auto simp add: AX-def EX-def)
  then show ?thesis by (simp only: AF-lfp) (rule lfp-unfold)
qed

lemma EG-fp: EG p = p ∩ EX EG p
proof −
  have mono (λs. p ∩ EX s) by rule (auto simp add: EX-def)
  then show ?thesis by (simp only: EG-def) (rule gfp-unfold)
qed
```

From the greatest fixed point definition of AG p, we derive as a consequence of the Knaster-Tarski theorem on the one hand that AG p is a fixed point of the monotonic function $\lambda s.\ p \cap \text{AX}\ s$.

```
lemma AG-fp: AG p = p ∩ AX AG p
proof −
  have mono (λs. p ∩ AX s) by rule (auto simp add: AX-def EX-def)
  then show ?thesis by (simp only: AG-gfp) (rule gfp-unfold)
qed
```

This fact may be split up into two inequalities (merely using transitivity of $\subseteq$, which is an instance of the overloaded $\leq$ in Isabelle/HOL).

```
lemma AG-fp-1: AG p ⊆ p
```

proof −
note *AG-fp* **also have** $p \cap \text{AX AG } p \subseteq p$ **by** *auto*
finally show *?thesis* **.**
qed

lemma *AG-fp-2*: $\text{AG } p \subseteq \text{AX AG } p$
proof −
note *AG-fp* **also have** $p \cap \text{AX AG } p \subseteq \text{AX AG } p$ **by** *auto*
finally show *?thesis* **.**
qed

On the other hand, we have from the Knaster-Tarski fixed point theorem that any other post-fixed point of $\lambda s.\ p \cap \text{AX } s$ is smaller than $\text{AG } p$. A post-fixed point is a set of states q such that $q \subseteq p \cap \text{AX } q$. This leads to the following co-induction principle for $\text{AG } p$.

lemma *AG-I*: $q \subseteq p \cap \text{AX } q \Longrightarrow q \subseteq \text{AG } p$
by (*simp only*: *AG-gfp*) (*rule gfp-upperbound*)

3 The tree induction principle

With the most basic facts available, we are now able to establish a few more interesting results, leading to the *tree induction* principle for AG (see below). We will use some elementary monotonicity and distributivity rules.

lemma *AX-int*: $\text{AX } (p \cap q) = \text{AX } p \cap \text{AX } q$ **by** *auto*
lemma *AX-mono*: $p \subseteq q \Longrightarrow \text{AX } p \subseteq \text{AX } q$ **by** *auto*
lemma *AG-mono*: $p \subseteq q \Longrightarrow \text{AG } p \subseteq \text{AG } q$
by (*simp only*: *AG-gfp*, *rule gfp-mono*) *auto*

The formula $\text{AG } p$ implies $\text{AX } p$ (we use substitution of $\subseteq$ with monotonicity).

lemma *AG-AX*: $\text{AG } p \subseteq \text{AX } p$
proof −
have $\text{AG } p \subseteq \text{AX AG } p$ **by** (*rule AG-fp-2*)
also have $\text{AG } p \subseteq p$ **by** (*rule AG-fp-1*) **moreover note** *AX-mono*
finally show *?thesis* **.**
qed

Furthermore we show idempotency of the AG operator. The proof is a good example of how accumulated facts may get used to feed a single rule step.

lemma *AG-AG*: $\text{AG AG } p = \text{AG } p$
proof
show $\text{AG AG } p \subseteq \text{AG } p$ **by** (*rule AG-fp-1*)
next
show $\text{AG } p \subseteq \text{AG AG } p$
proof (*rule AG-I*)
have $\text{AG } p \subseteq \text{AG } p$ **..**

```
    moreover have AG p ⊆ AX AG p by (rule AG-fp-2)
    ultimately show AG p ⊆ AG p ∩ AX AG p ..
  qed
qed
```

We now give an alternative characterization of the AG operator, which describes the AG operator in an "operational" way by tree induction: In a state holds AG p iff in that state holds p, and in all reachable states s follows from the fact that p holds in s, that p also holds in all successor states of s. We use the co-induction principle *AG-I* to establish this in a purely algebraic manner.

```
theorem AG-induct: p ∩ AG (p → AX p) = AG p
proof
  show p ∩ AG (p → AX p) ⊆ AG p  (is ?lhs ⊆ -)
  proof (rule AG-I)
    show ?lhs ⊆ p ∩ AX ?lhs
    proof
      show ?lhs ⊆ p ..
      show ?lhs ⊆ AX ?lhs
      proof −
        {
          have AG (p → AX p) ⊆ p → AX p by (rule AG-fp-1)
          also have p ∩ p → AX p ⊆ AX p ..
          finally have ?lhs ⊆ AX p by auto
        }
        moreover
        {
          have p ∩ AG (p → AX p) ⊆ AG (p → AX p) ..
          also have ... ⊆ AX ... by (rule AG-fp-2)
          finally have ?lhs ⊆ AX AG (p → AX p) .
        }
        ultimately have ?lhs ⊆ AX p ∩ AX AG (p → AX p) ..
        also have ... = AX ?lhs by (simp only: AX-int)
        finally show ?thesis .
      qed
    qed
  qed
next
  show AG p ⊆ p ∩ AG (p → AX p)
  proof
    show AG p ⊆ p by (rule AG-fp-1)
    show AG p ⊆ AG (p → AX p)
    proof −
      have AG p = AG AG p by (simp only: AG-AG)
      also have AG p ⊆ AX p by (rule AG-AX) moreover note AG-mono
      also have AX p ⊆ (p → AX p) .. moreover note AG-mono
      finally show ?thesis .
    qed
```

```
  qed
qed
```

4 An application of tree induction

Further interesting properties of CTL expressions may be demonstrated with the help of tree induction; here we show that AX and AG commute.

```
theorem AG-AX-commute: AG AX p = AX AG p
proof −
  have AG AX p = AX p ∩ AX AG AX p by (rule AG-fp)
  also have ... = AX (p ∩ AG AX p) by (simp only: AX-int)
  also have p ∩ AG AX p = AG p  (is ?lhs = -)
  proof
    have AX p ⊆ p → AX p ..
    also have p ∩ AG (p → AX p) = AG p by (rule AG-induct)
    also note Int-mono AG-mono
    ultimately show ?lhs ⊆ AG p by fast
  next
    have AG p ⊆ p by (rule AG-fp-1)
    moreover
    {
      have AG p = AG AG p by (simp only: AG-AG)
      also have AG p ⊆ AX p by (rule AG-AX)
      also note AG-mono
      ultimately have AG p ⊆ AG AX p .
    }
    ultimately show AG p ⊆ ?lhs ..
  qed
  finally show ?thesis .
qed

end
```

References

[1] K. McMillan. Lecture notes on verification of digital and hybrid systems. NATO summer school, http://www-cad.eecs.berkeley.edu/~kenmcmil/tutorial/toc.html.

[2] K. McMillan. *Symbolic Model Checking: an approach to the state explosion problem.* PhD thesis, Carnegie Mellon University, 1992.