Model Checking

Felix Chang, Greg Dennis, Derek Rayside, Rob Seater
MIT CSAIL 6.883, Prof Ernst
{fschang, gdennis, drayside, rseater}@mit.edu
November 23, 2005

Contents

1 Scribbles

approaches { *explicit*, *symbolic*

technologies { *explicit*, *BDD*, *SAT*

checkers { *Spin* explicit; *SMV* symbolic, BDD-based

etymology { *model* model-theoretic sense; *model* modeling sense

properties { *safety* some specific bad thing never happens; *liveness* some good thing eventually happens infinitely often

2 Temporal Logics

There are three temporal logics used in model checking:

temporal logics { *LTL* linear temporal logic; *CTL* computation tree logic; *CTL** subsumes LTL and CTL

2.1 CTL* (extended CTL)

CTL* comprises

- atomic propositions (a variable from some state)
- logical connectives ($\Rightarrow$, $\neg$, $\vee$, $\wedge$)
- temporal operators (X, F, G, U, W) – LTL (linear temporal logic), CTL (conditional tree logic), and CTL* (extended CTL). We begin with CTL*, which subsumes CTL and LTL.
- path quantifiers (A, E)

Grammar:

```
state formulae (initial token)
s ::= Atomic(x)   atomic fromula
    | !s          negation
    | s V s       disjunction
    | s ^ s       conjunction
    | A p         all paths starting here
    | E p         some paths starting here

path formulae
p ::= s           every state formula is a path formula*
    | !p          negation
    | p V p       disjunction
    | p ^ p       conjunction
    | X p         next (on the second state in this path)
    | F p         finally (on some state in this path)
    | G p         generally (on all states in this path)
    | p1 U p2     strong until (p1 until eventually p2,
                    p2 must occur)
    | p1 W p2     weak until (p1 until possibly p2,
                    p2 needn't occur)

*A state formula used as a path formula only looks at the
first state in a path.
```

While path formulae may be composed, path formulae must always be followed by a state formulae, never by another path formula. `AXX` is well formed but `AAX` is not. Figure 2.1 shows examples of how path quantifiers work. Figure 2.1 shows examples of how state quantifiers combine with path quantifiers.

Figure 1 Path quantifiers in CTL*

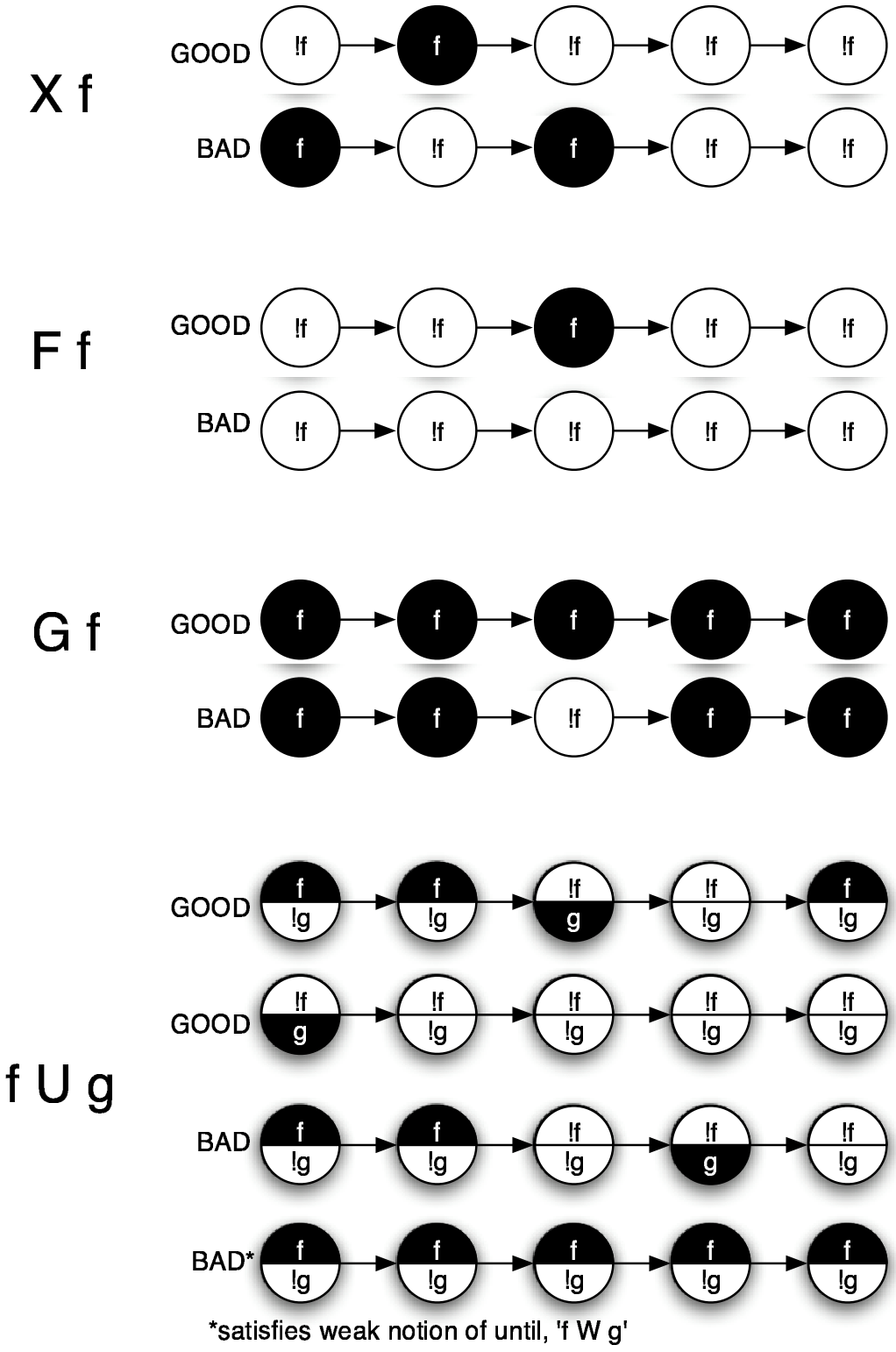
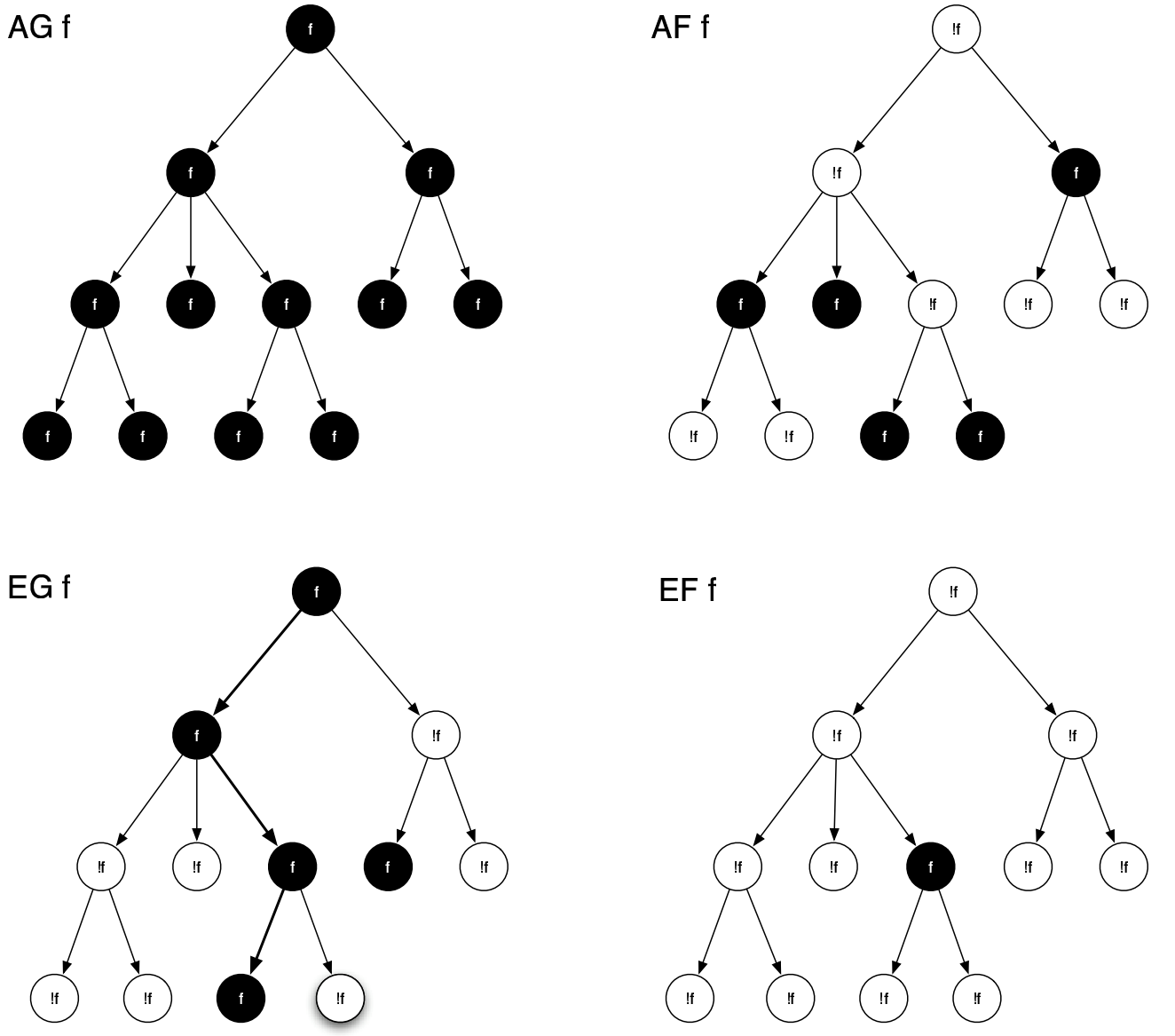


Figure 2 Interaction of state and path quantifiers in CTL*



Sample Specifications:

```
AG(req => F ack)
    each request will eventually be acknowledged

AG(!(!req U ack))
    each acknowledgement follows some request

AG !(def(x) & AX !EF use(x))
    x is never definied if not used (dead code)

AG(!(def(x) & AX (A (!use(x) W def(x)) )))
    x is defined but not used before re-defined

AGF req
    request occurs infinitely often
```

Identities:

```
!A f = E !f
!G f = F !f
!X f = X !f
F f = true U !f
```

The following operators are sufficient to express any CTL* expression: `!, V, X, U, E`.

2.2 CTL (Computation Tree Logic)

CTL is CTL* with the restriction that path quantifiers (X, F, G, U, W) must be preceeded by state quantifiers (A, E) and never by another path quantifier. Another way of looking at CTL is that there are 10 quantifiers.

```
AX, EX    for all/some successor states
AF        inevitably
EF        potentially
AG        invariably
EG        potentially always
AU, EU    for all some paths, until eventually
AW, EW    for all some paths, until possibly
```

A minimal subset is `!, V, EX, EG, EU`.

Examples in CTL:

```
EF AG f
AG EF f
```

Examples not in CTL:

```
AFG f
AXX f
```

The first is not expressible in CTL. The second is expressible as AX AX f.

2.3 LTL (Linear Temporal Logic)

LTL is CTL* with the restriction that the first quantifier must be A, and the only state formulae permitted are atomic propositions. That is,

```
s ::=   atomic(p)
    |   A p
p ::=   ! p
    |   p V p
    |   p ^ p
    |   X p
    |   F p
    |   G p
    |   p U p
    |   p W p
```

When writing LTL in isolation from CTL or CTL*, the A is generally left off the front of each formula.

Examples in LTL:

```
A FG f
A (!GFp V Fg)
```

Examples not in LTL:

```
AF AG f
AG EF f
```

2.4 Relative Expressive Power

In CTL and LTL:

```
AF f
AG f
```

In CTL but not LTL:

`AG EF f`	from any state, it is possible to get to a state for which f holds
`AF EG f`	it is possible to get to a state from which f holds no matter where you go

In LTL but not CTL:

```
A GF f    infinitely often
A FG f    almost everywhere
```

In CTL* but neither CTL nor LTL:

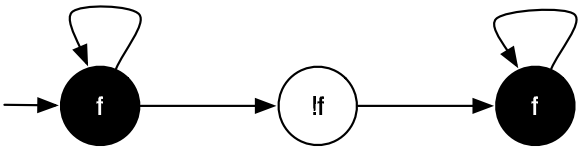
```
(AF EF f) V (A FG f)
```

In none of the three:

```
AA f
```

Most properties that get written are in $LTL \cap CTL$, except for fairness which is in $CTL* - (LTL \cap CTL)$

Figure 3 `FG f` is expressible in LTL but not in CTL.



2.5 LTL vs. CTL: `AFGf`

The most unintuitive result of those above is that LTL is not subsumed by CTL. We will thus take a more detailed look at `AFGf` and why it is in not in CTL.

`FG f` holds if there is a point after which `f` is always true. For example, it holds for the case shown in Figure 2.5. `AFGf` is not valid CTL since it has a `G` directly following a `F`. We might consider `AFAGf`, but that is not equivalent to `AFGf` – it is too strong and does not hold for the example shown in Figures 2.5. Consider the path that follows the leftmost self-arc infinitely. It need never reach a state where f is false, but it always *could* reach such a state.

3 Optimizations

3.1 Explicit State

nested DFS

partial order reductions

coarsening

state indexing

bit-state hashing

abstraction

3.2 Symbolic

Macro Steps helpfull, reasonable

References

[1] Ph Schnoebelen. The complexity of temporal logic model checking. In *Advances in Modal Logic*, pages 1–44. World Scientific Publishing, 2002.