

Formal verification with natural language specifications

December 1999

Alexander Holt

(Claire Grover, Ewan Klein, Marc Moens)

Language Technology Group

Division of Informatics

University of Edinburgh

alexander.holt@ed.ac.uk

Why an NL interface for formal verification?

- ▶ Formal verification: mathematical guarantees about correctness of program/protocol/chip.
- ▶ Increasingly successful, especially for hardware—but industrial take-up limited
- ▶ One reason: “only experts can make the tools work”

Proposal:

- ▶ Some techniques require less expertise: *model checking*
- ▶ But must know specification language: temporal logic
- ▶ Many hardware engineers not at ease with temporal logic, despite precise intuitions about intended behaviour
- ▶ Appears to be a role for natural language: *translate* English specifications to temporal logic

2

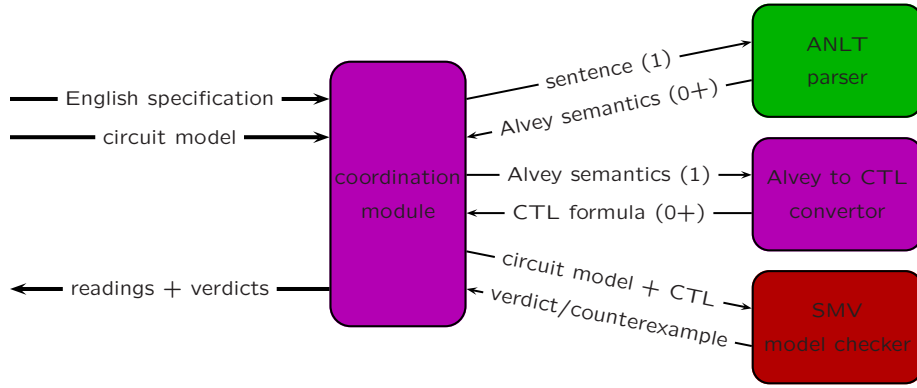
What this talk is about

- ▶ Why an NL interface for formal verification?
 - ▷ Motivation: expert user bottleneck
 - ▷ Plausible application: model checking
- ▶ A prototype: temporal model checking with English specs
 - ▷ How the interface works
 - ▷ Example: from English to temporal logic
- ▶ Observations
 - ▷ Restricting the language
 - ▷ Ambiguity
 - ▷ Compositionality & expressibility
 - ▷ Justifying conversions
 - ▷ Timing diagrams & waveforms
- ▶ Results & prospects

Prototype: English specifications for SMV

- ▶ SMV: freely available model checker, uses computation tree logic (CTL)
- ▶ Alvey Natural Language Toolkit (ANLT): efficient chart parser + reasonably large-scale English grammar
- ▶ What we did:
 - ▷ Gathered some data
 - ▷ Customized ANLT grammar for hardware specification domain
 - ▷ Wrote convertor from ANLT semantic representations to formulas of CTL
 - ▷ Packaged it up
 - ▷ Web interface

System structure



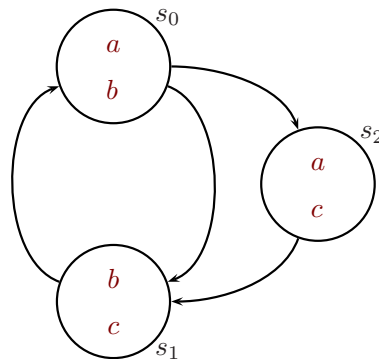
Example

- (1) After sig1 is active, sig2 is active for three cycles.
- (2) (DECL
 (AFTER (uqe (some (e1) e1))
 (BE (uqe (some (e2) (PRES e2)))
 (ACTIVE (name (the (x1) (and (sg x1) (named x1 sig1))))
 (degree unknown)))
 (and
 (BE #1=(uqe (some (e3) (PRES e3)))
 (ACTIVE (name (the (x2) (and (sg x2) (named x2 sig2))))
 (degree unknown)))
 (FOR #1# (uq ((NN \3) (x3) (and (pl x3) (CYCLE x3))))))
 (timespan unknown)))
- (3) (AFTER
 (ACTIVE sig1)
 (FOR
 (CYCLE 3)
 (ACTIVE sig2))
 (timespan unknown))

Computation Tree Logic

AGp	for all paths, globally p is true
AFp	for all paths, p is true at some future point
AXp	for all paths, at the next state p is true
$A[p U q]$	for all paths, q is eventually true, and p till then

	at s_0
AXc	true
AGb	false
$AF(AX(a \wedge b))$	true



Example (cont.)

- (1) After sig1 is active, sig2 is active for three cycles.
- (3) (AFTER
 (ACTIVE sig1)
 (FOR
 (CYCLE 3)
 (ACTIVE sig2))
 (timespan unknown))
- (4) $AG(\text{sig1} \rightarrow AX^1 \text{FOR}^3 \text{sig2})$
- (5) $AG(\text{sig1} \rightarrow AX(\text{sig2} \wedge AX\text{sig2} \wedge AXAX\text{sig2}))$

Observations 1: Restricting the language

- ▶ How to guarantee that English input *has* valid CTL translation?
- ▶ —by restricting the input (*controlled language*)
- ▶ Started by back-translating from CTL to English
- ▶ Developed hierarchy of subsets of English, according to
 - ▷ nature of semantic interpretation (compositional?)
 - ▷ expressiveness of ‘natural’ target formalism (CTL—CTL with extended events—FOL—?)
- ▶ Middle path between point in hierarchy and actual data

Observations 3: Compositionality & expressibility

- ▶ Strict compositional translation inadequate:

$$\left. \begin{array}{l} \text{if sig1 is high then} \\ \text{sig1 is high until} \end{array} \right\} \text{sig2 is high for 3 cycles}$$
- No single CTL fragment for *sig2 is high for 3 cycles*:

$$\text{sig1} \rightarrow \text{sig2} \wedge \text{AX}(\text{sig2} \wedge \text{AXsig2})$$

$$\text{A}[\text{sig1} \text{ U sig2} \wedge \text{sig1} \wedge \text{AX}(\text{sig2} \wedge \text{sig1} \wedge \text{AXsig2})]$$

- ▶ Backwards temporal reference common in NL specs but CTL/LTL normally omit past operators
- ▶ CTL is harder than LTL

Observations 2: Ambiguity

From our corpus of specification discourses:

- (6) *After sig1 becomes active sig2 should not become active until sig3 becomes active.*

System assigns two readings (English syntactic ambiguity):

$$(7) \quad \text{A}[(\text{sig1} \rightarrow \text{AX}(\neg \text{sig2})) \text{ U sig3}]$$

$$(7') \quad \text{AG}(\text{sig1} \rightarrow \text{AXA}[\neg \text{sig2} \text{ U sig3}])$$

How to explain different interpretations to user?

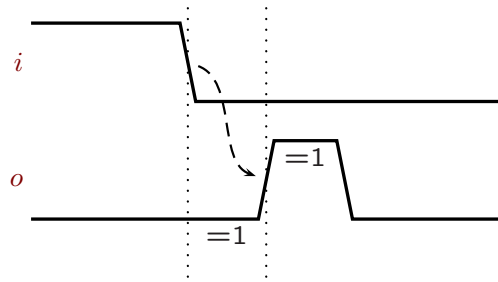
Currently can order *false* readings by length of counter-example trace when presenting results. Fairly arbitrary metric (but easy to implement ...).

Observations 4: Justifying conversions

Conversion of ANLT semantic representations to temporal logic:

- ▶ Scoped or unscoped ANLT output?
- ▶ Scoped ANLT semantic representations not quite a logic
- ▶ Our intermediate representation definitely not a logic
- ▶ First-order relations over Davidsonian event variables (e_1, e_2, e_3) must become modal operators
- ▶ *Formal* verification demands that we justify our representations and conversions between them

Observations 5: Timing diagrams & waveforms



Timing diagrams: signal waveforms annotated for causality, abstraction, etc.

Very common, especially in conjunction with linguistic descriptions of circuit behaviour.

Can be formalized (e.g., Fislér). Perhaps best seen as adjunct to linguistic specification, though commercial input tools exist.

Intend to at least back-generate waveforms from model checker counter-traces.

Results & prospects

- ▶ Instantiates interesting class of NL understanding systems whose target is a specialised reasoning engine
- ▶ Increased access to a significant industrial technique
- ▶ Demo: <http://www.ltg.ed.ac.uk/prosper/>

Prospects:

- ▶ Better coverage, better conversions, more robust
- ▶ Integration into hardware verification workbench: user trials
- ▶ Higher-order logic? New inferences; more natural conversions; framework for *justifying* conversions (HOL)
- ▶ Model checkers that reveal more: game-theoretic?
- ▶ Mixing diagrams and language