Inferring Formal Software Specifications From Episodic Descriptions

Van E. Kelly
Uwe Nonnenmann

AT & T Bell Laboratories
600 Mountain Ave. 3D-418
Murray Hill, New Jersey 07974

ABSTRACT

The WATSON automatic programming system computes formal behavior specifications for process-control software from informal "scenarios": traces of typical system operation. It first generalizes scenarios into stimulus-response rules, then modifies and augments these rules to repair inconsistency and incompleteness. It finally produces a formal specification for the class of computations which implement that scenario and which are also compatible with a set of "domain axioms". A particular automaton from that class is constructed as an executable prototype for the specification.

WATSON's inference engine combines theorem proving in a very weak temporal logic with faster and stronger, but approximate, model-based reasoning. The use of models and of closed-world reasoning over "snapshots" of an evolving knowledge base leads to an interesting special case of non-monotonic reasoning.

I. INTRODUCTION

The WATSON[1] system addresses an important issue in applying AI to the early stages of software synthesis: converting informal, incomplete requirements into formal specifications that are consistent, complete (for a given level of abstraction), and executable. For ten years AI research has attempted this task [Balzer 77], but the problem's difficulty is exacerbated by the many possible types of imprecision in natural language specifications, as well as by the lack of a suitable corpus of formalized background knowledge for most application domains.

We have restricted ourselves to a single common style of informal specification: the "scenario". Scenarios are abstracted traces of system behavior for particular stimuli, requiring very modest natural-language technology to interpret. We have also selected a domain -- the design of telephone services -- in which most "common-sense" notions of proper system behavior can be axiomatized in a logic formalism with very weak temporal inference capabilities [DeTreville 84]. The resulting domain axioms, together with the small signaling "vocabulary" of telephones, makes exhaustive reasoning more tractable. These restrictions were justified, allowing us to concentrate on two specific issues:

a. gaining leverage from domain knowledge in generalizing scenarios, and

b. engineering fast hybrid reasoning systems for interactive environments.

In the next section, we summarize WATSON. We then demonstrate how it exploits domain knowledge using a simple case study, examine its hybrid inference engine, compare it to other research, and, finally, outline our future plans for the system.

II. WATSON IN A NUTSHELL

Figure 1 summarizes WATSON. Scenarios are parsed and generalized into logic rules. These scenarios and rules describe **agents** (finite-state process abstractions), sequences of **stimuli**, and **assertions** about the world whose truth values change dynamically. Omissions and informality in the original scenario may lead to over-generalized, under-generalized, or missing rules.

After constructing partial, underconstrained models for each type of agent mentioned in the scenario, WATSON interactively refines the rules and models. It repairs rule contradictions, eliminates unreachable or dead-end model states, infers missing rules, and ascertains that all stimuli are handled in every state of the world. To complete information missing from its domain knowledge, WATSON asks the user true-false questions about short hypothetical scenarios. The user is never asked to envision multiple situations at once.

Finally, WATSON performs an exhaustive consistency test on its refined set of rules. This proves

1. named for Alexander G. Bell's laboratory assistant, not the fictitious M.D.

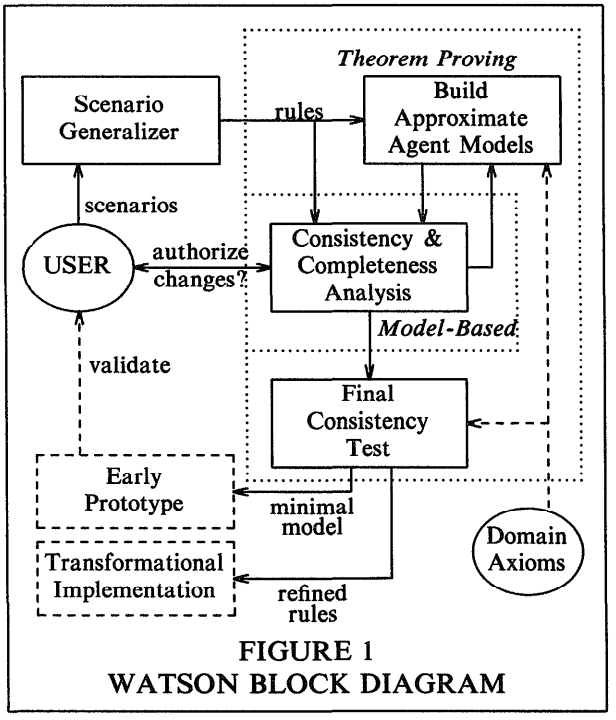

FIGURE 1
WATSON BLOCK DIAGRAM

that the refined rules prohibit transitions from consistent to inconsistent world-states for all input stimuli. If these rules are later embedded in a larger set of rules (*i.e.*, one describing more telephone services), and the larger set of rules passes the consistency test, then the original scenarios will still work. There are many possible computations (*i.e.*, agent models) that could implement a given set of scenarios, but the refined rules form a model-independent verification condition for all such computations.[2]

Different finite-state models for the same rules may require different numbers of state transitions to implement any given rule subset. WATSON computes a minimal (fewest states and transitions) model for the specification, which is useful for early acceptance testing, either by software simulation or by use of a special test-bed with actual telephone equipment.

III. SCENARIO CASE STUDY

To highlight how WATSON converts scenarios into a coherent system definition, we begin by analyzing the structure of a scenario, noting its ambiguities and omissions. Next we examine the telephony domain axioms, which provide the backdrop for all our interpretations. Finally we consider one particular anomaly fixed by WATSON.

2. For a more complete discussion of the final consistency test, the generalizer, detailed models used in the hybrid reasoner, and more case studies see [Kelly 87].

A. Scenarios, Episode Structures, and Rules

A single scenario defining a simplified version of **"plain old telephone service"**, or **POTS**[3], is given below:

First, Joe is on-hook,
and Joe goes off-hook,
then Joe starts getting dial-tone.
Next, Joe dials Bob;
then Bob starts ringing
and Joe starts calling Bob
and Joe starts getting ringback.
Next, Bob goes off-hook,
then Joe gets connected to Bob.
Next, Joe goes on-hook,
then Joe stops being connected to Bob.

Joe and Bob are agents of the same type, whose implementations must be isomorphic. The stimuli that drive the scenario are "going-on-hook", "going-off-hook", and "dialing"; in our simplified telephone domain, these are the only ways in which a telephone user can possibly affect a telephone instrument.[4] Seven different predicates (*e.g.*, "on-hook", "get-dial-tone", "connected") are used in the preceding scenario to describe the changing state of the world. Assertions constructed using these predicates appear in two different *tenses* in the scenario: a present tense ("Joe **is** on-hook") and an immediate-future tense denoting immanent change in the truth-value of the assertion ("Bob **starts** ringing"), presumably occurring before the next stimulus. A *2-tense logic* is thus a natural formalism for capturing this scenario.

1. Episodic Structure and Rules

The scenario is understood as a sequence of four sequential **episodes**, each consisting of three parts:

- **antecedents**, assertions known to be true of the agents before a stimulus,
- a **stimulus**, which may have implicit persistent side-effects[5], and
- explicit **consequents**, or changes in the truth-values of selected assertions after the stimulus. The consequents of one episode implicitly determine the antecedents of the next.

3. "On-hook" refers to when the telephone handset is resting in its cradle, *i.e.*, "hung-up". "Off-hook" is the opposite state. When "on-hook", the phone is disconnected, except for its ringing circuit. "Ringback" is the sound that a caller hears when a called party rings.

4. We omit the stimulus of momentarily "flashing" the phone switch-hook, which is usually recognized by switching systems as different from hanging up and then going off-hook again.

5. For example after "Joe goes off-hook", we should know implicitly that Joe is no longer on-hook. But the momentary stimulus "dials" has no persistent side-effects on the state of the world.

Each episode represents a single stimulus-response (S-R) cycle for the system, and is mapped into a set of logic rules, one rule per each consequent. The antecedents of the episode appear in the present tense and the consequents appear in the immediate-future tense, using the modal operators BEGINS and ENDS. The following six rules, numbered by episode, implement the scenario:

$R1.1$: $\forall x\ [on\_hook(x) \wedge EVENT(goes\_off\_hook(x)) \supset BEGINS(dial\text{-}tone(x))]$

$R2.1$: $\forall x,y\ [dial\text{-}tone(x) \wedge EVENT(dials(x,y)) \supset BEGINS(calling(x,y))]$

$R2.2$: $\forall x\ [\exists y[dial\text{-}tone(x) \wedge EVENT(dials(x,y))] \supset BEGINS(ringback(x))]$

$R2.3$: $\forall x,y\ [dial\text{-}tone(x) \wedge EVENT(dials(x,y)) \supset BEGINS(ringing(y))]$

$R3.1$: $\forall x,y\ [calling(y,x) \wedge ringback(y) \wedge ringing(x) \wedge EVENT(goes\_off\_hook(x)) \supset BEGINS(connected(y,x))]$

$R4.1$: $\forall x,y\ [connected(x,y) \wedge EVENT(goes\_on\_hook(x))] \supset ENDS(connected(x,y))$

2. Scenario Anomalies

One must distinguish between the level of abstraction of the scenario (*e.g.*, dialing is considered an atomic operation) and its informality. WATSON does not change a scenario's abstraction level, but rather corrects some of the following anomalies due to informality, *e.g.*:

- Antecedents may omitted from episodes, causing over-generalized rules, such as R2.3.
- Consequents may be missing from episodes, leading to missing rules; for instance, it is not stated that Bob stops ringing when he goes off-hook.
- Irrelevant antecedents may be included in an episode, leading to under-generalized rules.
- Causal links among antecedents and consequents are not always made explicit. Coincidence and causality may be confused.
- Specifications for a particular agent type are split over several agents (Joe and Bob), and are not harmonized. For example, both R1.1 and R3.1 have to do with telephones going off-hook: Joe in R1.1 and Bob in R3.1.

B. Domain Axioms

The "common-sense" domain axioms describing telephone services combine several different sorts of information:

1. Axioms and axiom schemas that embed temporal reasoning with 2-tense logic into standard first-order logic (FOL) resolution, for example:
   - $\forall A \in ASSERTIONS\ [[A \supset \neg BEGINS(A)] \wedge [\neg A \supset \neg ENDS(A)]]$,
   - serialization axioms for stimuli.
2. Declarations for telephone terminology:
   - types of agents,
   - stimuli and their side-effects,
   - predicates and argument types for constructing assertions.
3. Hardware constraints on telephone devices:
   - telephones must be on-hook or off-hook, but not both.
   - telephones can't ring when off-hook, and can't dial or accept audio services (like dial-tone, busy signal, or ringback) while on-hook.
   - "X dials Y" is a null operation unless X has dial-tone.
4. Etiquette rules for telephone services:
   - telephones receive at most one audio service at a time.
   - telephones should not start to ring unless someone is calling.

While this body of knowledge appears complex, it can be written down compactly (about a dozen axiom schemas, excluding declarations) and in a scenario-independent form. This means not writing FOL axioms directly, but inferring default terminology declarations from the scenario itself (as in [Goldman 77]), and using second-order schema notation extensively.

C. Consistency and Completeness Analysis

1. Types of Correction Attempted

WATSON applies four main corrections to sets of rules:

a. **Repairing inconsistent rules.** Rules contradict if they have the same stimulus, their antecedents are compatible, but their consequents are incompatible. The correction usually involves strengthening the antecedents of one or more of the contradictory rules.

b. **Finishing incomplete episodes by adding new rules.** For instance, in the second episode of the POTS scenario, no rule ends Joe's dial-tone, yet it clearly must end.

c. **Eliminating unreachable "states".** Some combinations of assertions may be consistent with the domain axioms but never happen in any scenario. For instance, Joe may call Bob and not get a ringback (*i.e.*, if he gets a busy-signal instead), but our scenario does not show this. WATSON will solicit a new scenario that accesses such states, or modify existing rules.

d. **Ensuring all stimuli are handled in all states.** Suppose Joe hangs up during dial-tone. The domain axioms completely determine which "state" Joe would then be in (on-hook but not ringing). So, WATSON can build this entire episode from first principles. In more complex cases, WATSON cannot determine the exact outcome, and must get help from the user, either by proposing several alternatives or asking for a new scenario.

The procedure is similar for each of these four cases. First the rules and agent models are used to detect potential problems. Then a heuristic search is performed for a "simplest" workable fix. Next, the fix is explained to the user, who is asked for approval. Finally, the rules and models are updated.

2. Example: Fixing the Inconsistency of R1.1 and R3.1

Consider the contradiction between R1.1 and R3.1. After detecting the inconsistency, WATSON notes that the antecedents of R1.1 are strictly more general than those of R3.1. WATSON attempts to strengthen the antecedents of R1.1 until the antecedents of the two rules become incompatible. The most obvious way, by finding all antecedents of R3.1 not in R1.1, negating them, and conjoining them to R1.1, produces something correct but verbose:

$$R1.1a\colon \forall x\ [on\text{-}hook(x) \wedge [\neg ringing(x) \vee [\forall y\ [\neg calling(y,x) \vee \neg ringback(y)]]] \wedge EVENT(goes\_off\_hook(x))] \supset BEGINS(dial\text{-}tone(x))].$$

WATSON searches for simpler versions that are "negligibly" stronger then R1.1a. Considering rules involving only a single negated antecedent from R3.1, it finds the following candidates which are simpler than R1.1a, but stronger by varying amounts:

$$R1.1b\colon \forall x\ [on\text{-}hook(x) \wedge \forall y\ [\neg ringback(y)] \wedge EVENT(goes\_off\_hook(x)) \supset BEGINS(dial\text{-}tone(x))]$$

$$R1.1c\colon \forall x\ [on\text{-}hook(x) \wedge \forall y\ [\neg calling(y,x)] \wedge EVENT(goes\_off\_hook(x)) \supset BEGINS(dial\text{-}tone(x))]$$

$$R1.1d\colon \forall x\ [on\text{-}hook(x) \wedge \neg ringing(x) \wedge EVENT(goes\_off\_hook(x)) \supset BEGINS(dial\text{-}tone(x))]$$

It heuristically ranks R1.1d as the simplest, since it doesn't introduce new variables to R1.1. It then tries to show that R1.1d is "virtually" as general as R1.1a by establishing

$$CWR \models \forall x\ [ringing(x) \supset \exists y\ [calling(y,x) \wedge ringback(y)]],$$

where CWR is an expanded set of "closed world rules" constructed on the fly from the scenario rules to support reasoning both forward and backward in time over a single S-R cycle. These contain an implicit assumption that the presently known rules are the only rules. Since *according to the known rules* there is no way for a phone to ring without another phone initiating a call and getting ringback, this proof succeeds, leading to a conclusion that R1.1d is safe to use.

At this point, WATSON knows it has a "maximally simple" acceptable solution. It now asks the user for approval before replacing R1.1 with R1.1d. It paraphrases R1.1 and R3.1 back into "scenario-ese", describes the contradiction, and asks for permission to replace R1.1 with R1.1d. If WATSON had found two equally desirable solutions, the user would be asked to make the choice.

D. Summary of the POTS Case Study

The simplified POTS scenario requires fourteen rules to pass the final consistency check. WATSON can obtain these by rewriting three of the original six and generating the remaining eight from first principles. Four of these eight are required to complete unfinished episodes, and the other four handle telephones that hang up midway through the scenario (unanticipated stimuli). This requires 5 minutes of machine time on a Symbolics 3600 workstation, 96% of which is spent proving theorems. Our home-brew theorem prover runs faster than similar resolution engines (*i.e.*, full FOL, breadth-first or depth-first clause selection) that we have used in the past, but the response time still taxes user patience.

IV. DISCUSSION

A. Hybrid Reasoning

Not only is theorem proving slow, but WATSON's temporal reasoning is confined to a single stimulus-response cycle of the system. This weak logic was necessitated by exponential blowups when using more powerful, episode-chaining logics (*e.g.*, situation calculi) with the initially under-constrained scenario rules. Our solution integrates model-based reasoning into WATSON's inference methods.

WATSON's most important models are the minimal (fewest states) automatons required to implement each type of agent. Each state corresponds to a particular assignment of truth-values to every known assertion about the agent. State transitions are governed by the logical rules generalized from the scenario. These automatons are initially fragmentary and underconstrained, but evolve into connected, deterministic state transition graphs as WATSON edits the rules. The models are stored in a form that facilitates their use even when incomplete. They consist of a list of states telling what assertions hold in each state, a list of constraints on the possible states of agents both before and after each episode, and groupings of rules according to which state transitions they influence.

Model-based reasoning increases both the strength and speed of WATSON's reasoning. Of the queries posed during the correction/completion stage, about 20% require reasoning over multiple episodes, which cannot be done by theorem proving in 2-tense logic, but can be answered in the model. Another 65% are fully instantiated queries asking whether some property P holds in some model; for these, querying the model is typically 50 times faster using the theorem prover. The major *caveat* is incompleteness: a query might fail in WATSON's minimal model, and still hold in some other model. Fortunately, many important properties, such as graph connectivity, hold in minimal models or not at all. In other cases, models can at least filter the set of theorems to be proved. Still, care is needed when interpreting the results of model-based reasoning. In WATSON, such circumspection is presently hard-coded, but explicit automated meta-reasoning about model limitations is on our future research agenda.

B. Non-Monotonicity

There are well-known relationships between reasoning with a minimum Herbrand model and theorem proving with a closed-world assumption in ordinary first-order logic. Either one of these techniques, when applied to an evolving knowledge base, opens the door to non-monotonicity. Similar problems arise in WATSON's 2-tense logic and extended models. For example, if a new scenario provided a rule that would allow a telephone to ring without some other phone calling it and receiving a ringback (say, a line-test feature), that would invalidate the assumption by which we chose R1.1d as the best replacement for R1.1.

WATSON's non-monotonicity is nevertheless simpler than the general case. Most treatments of non-monotonicity assume retractions are forced asynchronously by evidence external to the system (new data). WATSON's problems all arise from internal decisions to apply closed-world reasoning. Thus, non-monotonicity can be minimized (but not eliminated) by careful static ordering of process steps, and by exploiting meta-knowledge about which model properties are stable. For instance, once WATSON has eliminated all unreachable states, it is safe to assume that no presently known state will ever become unreachable. The burden of this meta-reasoning should also be transferred eventually from WATSON's designers to WATSON.

For those relatively few determinations vulnerable to later retraction, WATSON applies brute force -- re-validating them all after every rule and model update. Fortunately, the speed of model-based reasoning reduces this overhead. Furthermore, WATSON batches unrelated rule updates together to minimize the frequency of update cycles. The POTS scenario, for example, can be processed in only four update-revalidate cycles.

V. RELATED RESEARCH

A. Programming From Informal Specifications

The general goals of WATSON recall the SAFE project at ISI ([Balzer 77], [Goldman 77]). SAFE attempted to understand an English prose description of a process, inferring the relevant entities mentioned in the process and their key relationships. It then generated an algorithm to implement the process. Of the six types of specification ambiguity corrected by SAFE, four of them, accounting for 88% of those corrections, were artifacts of using fairly unconstrained natural language input. Conversely, the scenario ambiguities corrected by WATSON did not arise in the SAFE case studies, because SAFE's initial specifications were more expansive (100-200 words).

B. Acquiring Programs from Examples

Several approaches have been used to "learn" programs based on sample traces: the pattern-matching approaches of Biermann & Krishnawamy [Biermann 76] and Bauer [Bauer 79], the language recognizer generators

of Angluin [Angluin 78] and Berwick [Berwick 86], and Andreae's NODDY system [Andreae 85]. Of the three, the work of Andreae is much the closest to the spirit of WATSON by its use of explicit domain knowledge. The other approaches attempt domain-independence, entailing that their input examples must be either numerous or meticulously annotated. NODDY, like WATSON, uses its domain knowledge to constrain the generality of the programs it writes, much like the function of negative examples in an example-based learning system. NODDY writes robot programs, and its domain knowledge is solid geometry.

One distinctive feature of WATSON, compared with other systems, is its handling of multi-agent interactions; the others are restricted to single-agent worlds. Another difference is that WATSON produces a specification for a class of computations, not just a single program. The final implementation, adapted to a particular machine architecture, may have much more internal complexity than WATSON's minimal model. The rules form an model-independent *post-hoc* verification criterion, and guidance for further transformational development.

VI. STATUS AND PLANS

WATSON evolved from PHOAN [DeTreville 84], which provided its axiomatization style, exhaustive consistency test, and FSA synthesis procedure. PHOAN successfully programmed POTS service for an experimental ethernet-based telephone switch [DeTreville 83]. By argument from parentage, we conclude that the same hardware should execute WATSON's code, but this has not yet been verified.

Our WATSON prototype can handle the POTS scenario and several extensions, such as busy-signals, billing, and multi-call contention. We are extending it to cover the "toy telephone" suite of telephone services defined in [IEEE 83]. An important feature of WATSON is its ability to detect unforeseen interactions among different services. The "toy telephone" domain contains several examples of such interactions.

We have previously noted ongoing research issues in meta-reasoning about model errors and non-monotonicity. We would also like to generalize the style of scenario WATSON can accept; *i.e.*, closer to idiomatic English. Therefore, a very flexible user-customizable English parser used to generate FOL database queries [Ballard 86] will soon be adapted for WATSON.

References

[Andreae 85] Peter Andreae, "Justified Generalization: Acquiring Procedures From Examples", MIT AI Lab Technical Report 834, 1985.

[Angluin 78] Dana Angluin, "Inductive Inference of Formal Languages from Positive Data", *Information and Control,* 1978, Vol 45, pp. 117-135.

[Ballard 86] Bruce Ballard and Douglas Stumberger, "Semantic Acquisition in TELI: A Transportable, User-Customizable Natural Language Processor", in *ACL-24 Proceedings,* Association For Computer Linguistics, 1986, pp. 20-29.

[Balzer 77] Robert Balzer, Neil Goldman, and David Wile, "Informality in Program Specifications", in *Proceedings of IJCAI-5,* 1977, pp. 389-397.

[Bauer 79] Michael A. Bauer, "Programming by Examples," *Artificial Intelligence,* May 1979, vol. 12, no. 1, pp. 1-21.

[Berwick 86] Robert C. Berwick, "Learning from Positive-Only Examples: The Subset Principle and Three Case Studies," in *Machine Learning: An Artificial Intelligence Approach, Volume II* Michalski, Carbonell, and Mitchell, eds.; Morgan Kaufmann, 1986

[Biermann 76] Alan W. Biermann and Ramachandran Krishnawamy, "Constructing Programs From Example Computations," *IEEE Transactions on Software Engineering,* Sept. 1976, Vol. SE-2, no. 3, pp. 141-153

[DeTreville 83] John DeTreville and W. David Sincoskie, "A Distributed Experimental Communications System", *IEEE Transactions on Communication,* Dec. 1983, Vol. COM-31, no. 12.

[DeTreville 84] John DeTreville, "Phoan: An Intelligent System For Distributed Control Synthesis", *ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments,* P. Henderson, ed.; 1984, pp. 96-103.

[Goldman 77] Neil Goldman, Robert Balzer, and David Wile, "The Inference of Domain Structure from Informal Process Descriptions", USC-ISI Research Report 77-64, October 1977.

[IEEE 83] *JSP & JSD: The Jackson Approach To Software Development,* IEEE Computer Society, 1983

[Kelly 87] Van E. Kelly and Uwe Nonnenmann, "From Scenarios To Formal Specifications: the WATSON System" Computer Technology Research Laboratory Technical Report (forthcoming).