Reinforcing a Claim in Commonsense Reasoning

Jonathan Campbell and Vladimir Lifschitz
Department of Computer Sciences
University of Texas at Austin
{campbell,vl}@cs.utexas.edu

Abstract

Formalizations of commonsense knowledge rely on highly simplified representations of the world. After a conclusion is justified on the basis of one such representation in a nonmonotonic logic, it may not remain valid when additional relevant facts are taken into account to bring the formalization to a closer approximation of reality. However, if the conclusion continues to hold in the enriched theory, we can consider it to be reinforced. An argument in support of a claim can be represented by a sequence of nonmonotonic theories, each containing and enhancing the previous theory in the sequence and each entailing the claim. This idea is illustrated here by a sequence of formalizations of "Sam's Calculus"—an example of commonsense reasoning due to Ernie Davis—in the language of the Causal Calculator.

Introduction

Formalizations of commonsense knowledge rely on highly simplified representations of the world. When a commonsense argument in support of a certain claim is represented as a proof in an axiomatic theory (Davis 1990), we are often tempted to ask what will happen to the proof if the axiomatization is made a little more realistic. If the theory is based on a nonmonotonic logic (Gabbay, Hogger, & Robinson 1994), then the conclusion may be invalidated simply by postulating an additional fact—some piece of commonsense knowledge not taken into account previously. If the conclusion does continue to hold in the extended theory, we can consider it to be reinforced, increasing our confidence that the conclusion is valid in reality and not simply an artifact of the simplified representation.

Consider, for instance, the following problem, contributed by Ernie Davis to the Common Sense Problem Page:[1]

> Sam got straight C's in high school math and has not thought for a moment about math in the 20 years since. Infer that Sam is not the person to ask about a calculus problem.

Even if we were not told that Sam has not thought about math since high school, just on the basis of the information about Sam's grades we would have been pessimistic about his ability to help us with calculus, because it would be natural to assume by default that Sam's knowledge of mathematics has not significantly improved over the years. This reasoning provides the first layer of evidence in support of the claim that Sam would not be able to solve a calculus problem.

On the other hand, we can take into account the fact that the default used in the first-level proof has exceptions. Isn't it possible that Sam learned some calculus in college? The answer to this question is no—we are told that after high school Sam has not thought about math. This is the second step in the process of accumulating arguments supporting the same conclusion.[2]

But isn't it possible that Sam knew calculus when he was a high-school student? Maybe it just happened that, for some reason, Sam's grades didn't reflect his good knowledge of math, including some calculus. Maybe his test-taking skills were poor, or maybe his math teacher didn't like him and didn't give him the good grades that he deserved. To this we can reply that Sam has not thought about math for a very long time, 20 years. Even if it somehow happened that, in spite of his poor grades, Sam knew some calculus in high school, he surely forgot it by now.[3] This is a third argument in support of the same assertion.

In this paper we describe three formalizations of Sam's story along the lines of this proposal. Each formalization is a nonmonotonic theory in the language of the Causal Calculator (CCALC)[4], and we have used CCALC to verify that Sam's inability to solve a calculus problem is entailed by each theory. The second theory is an extension of the first, and the third is an extension of the second, in the spirit of the discussion of elaboration tolerance in (McCarthy 1999) and (Lifschitz 2000).

We begin with a brief introduction to the input language of CCALC.

[1] `http://www-formal.Stanford.edu/leora/cs/` .

[2] We are indebted to John McCarthy for this observation. He noted, "The fact that Sam is weak in math can be expected to persist. The fact that he didn't think about math reinforces this persistence" (personal communication, May 8, 2002).

[3] This enhancement was suggested by Michael Gelfond (personal communication, November 18, 2002).

[4] `http://www.cs.utexas.edu/users/tag/cc/` .

The Language of CCALC

The main part of the input language of CCALC is known as action language $\mathcal{C}+$ and is discussed in (Giunchiglia *et al.* 2003), which presents a more detailed and formal description of the syntax and semantics of the language. $\mathcal{C}+$ serves for describing actions and their effects, and it can be used to formalize domains with a finite number of states and discrete time.

The expression

```
study(P,A) causes knows(P,A)
```

is a typical CCALC proposition. It is used in one of the formalizations of Sam's story below to express the effect of the action `study(P,A)` on the fluent `knows(P,A)`. The symbol `P` here is a variable for persons, while `A` is a variable for areas of knowledge; they are declared as follows:

```
:- variables
   P :: person;
   A :: area.
```

The symbols `person` and `area` are two of the sorts (or types) in our formalization:

```
:- sorts
   person; area.
```

Sam is a person; calculus and high school math are areas of knowledge:

```
:- objects
   sam                           :: person;
   calculus, highSchoolMath :: area.
```

The declarations

```
:- constants
 study(person,area)  :: exogenousAction;
 knows(person,area)  :: inertialFluent.
```

express that `study(P,A)` is an action, and `knows(P,A)` is a fluent. Studying is "exogenous" in the sense that the cause for `P` to study `A` is not part of our description of the domain. Knowledge is "inertial" in the sense that it is governed by the commonsense law of inertia: in the absence of information to the contrary, it is assumed not to change.

Formally, every proposition of $\mathcal{C}+$ is a "causal law"—an assertion about causation (McCain & Turner 1997; Giunchiglia & Lifschitz 1998). For instance, the postulate

```
good(a)
```

(A is a good grade) has the same meaning as

```
caused good(a)
```

(there is a cause for A to be a good grade). Noncausal assumptions are expressed by "negative" causal laws. For instance,

```
constraint knows(P,A) & dependsOn(A,A2)
             ->> knows(P,A2)
```

(if `P` knows `A`, then `P` must know every area on which `A` depends) is an abbreviation for

```
caused false
     if -(knows(P,A) & dependsOn(A,A2)
         ->> knows(P,A2))
```

where `-` is negation, and `->>` is material implication.

"States" are snapshots of the world at discrete instants of time, as in the situation calculus (McCarthy & Hayes 1969), and actions are assumed to be executed in the intervals between successive states. The formula

```
       (1:study(sam,calculus))
   ->> (2:know(sam,calculus))
```

expresses that Sam knows calculus at time step 2 if he studied it during the time interval that begins at time step 1.

In the formalizations that follow, `/\` and `\/` represent the universal and existential quantifiers over finite domains, and `<->` represents equivalence.

Basic Formalization

In the first formalization in the sequence, we naïvely assume that a person's grades in an area always correspond to his knowledge of that area, and there is no way to gain or lose knowledge. It demonstrates the following commonsense argument:

> Sam got straight C's in high school math, so he doesn't know high school math. Since this is a prerequisite for calculus, he doesn't know calculus either. Therefore he cannot solve a calculus problem.

CCALC can determine whether a formalization is satisfiable, and in particular, whether a given sequence of events is consistent with the rest of the propositions. Our goal will be to construct a theory of knowledge and earning grades, and then use CCALC to demonstrate that a sequence of events in which Sam earns all bad grades in high school math but then solves a calculus problem is inconsistent.

The formalization is shown in Figures 1 and 2. Figure 1 includes the portion which formalizes general common sense, rather than facts specific to the question of Sam and his calculus problem. Here, two sorts are declared in addition to the two described above; `grade` is for letter grades which can be earned by a person in an area, and `problem` is for problems a person may attempt to solve.

Several more constants are declared as well. The first three constants are declared as neither fluents nor actions; they are "rigid," in the sense that their values do not change over time. Symbols `dependsOn` and `good` are binary relations, while the value of `problemArea` belongs to sort `area`. Two actions are also declared, `solve` and `earnGrade`. Executing `earnGrade(P,G,A)` in a time interval intuitively means that `G` is one of the grades earned by `P` in area `A` during that time interval.

The three statements following the variable declarations express that A and B are good grades, and the other grades are not good. Similarly, the next two propositions define `dependsOn`; these state that calculus depends on high school math, but otherwise, the areas of knowledge under consideration are independent.

The first two propositions beginning with `constraint` are noncausal postulates which relate earning grades to knowledge. The first of these propositions states that it is impossible for a person to know an area at the end of a time

```
:- sorts
   area; grade; person; problem.

:- objects
   calculus, highSchoolMath :: area;
   a, b, c, d, f            :: grade.

:- constants
   dependsOn(area,area),
   good(grade)
      :: boolean;
   problemArea(problem)
      :: area;
   knows(person,area)
      :: inertialFluent;
   earnGrade(person,grade,area),
   solve(person,problem)
      :: exogenousAction.

:- variables
   A, A2 :: area;
   G     :: grade;
   P     :: person;
   Pr    :: problem.

good(a).
good(b).
default -good(G).

dependsOn(calculus,highSchoolMath).
default -dependsOn(A,A2).

constraint -knows(P,A)
     after [\/G | earnGrade(P,G,A)]
           & [/\G | earnGrade(P,G,A)
                    ->> -good(G)]
    unless abInaccurateGrade(P,A).

constraint knows(P,A)
     after [\/G | earnGrade(P,G,A)]
           & [/\G | earnGrade(P,G,A)
                    ->> good(G)]
    unless abInaccurateGrade(P,A).

constraint knows(P,A) & dependsOn(A,A2)
           ->> knows(P,A2).

nonexecutable solve(P,Pr)
   if -knows(P,problemArea(Pr)).
```

Figure 1: File `sams-1`, Part 1.

interval during which he earned bad grades and only bad grades in that area. The second is the dual of this proposition, stating that a person must know an area if he earns only good grades in it. We make no assertion about a person's knowledge if he earns both good and bad grades in an area.

```
:- objects
   sam              :: person;
   calculusProblem :: problem.

problemArea(calculusProblem)=calculus.

:- macros
   startOfHighSchool  -> 0;
   afterLastMathExam  -> 1;
   whenAskedProblem   -> 2;
   afterSolvedProblem -> 3.

:- macros
   earnStraightGrade(#1,#2,#3) ->
      [/\G | earnGrade(#1,G,#3)
             <-> G = #2].

:- query
 label :: 1;
 maxstep :: afterSolvedProblem;
 startOfHighSchool: earnStraightGrade(
    sam,c,highSchoolMath);
 whenAskedProblem: solve(sam,
    calculusProblem).
```

Figure 2: File `sams-1`, Part 2.

Note that both of these propositions end with

`unless abInaccurateGrade(P,A)`

This makes these rules defeasible; when

`abInaccurateGrade(P,A)`

becomes true, it disables the instances of the rules with those values of `P` and `A`. For now, this has no effect, but will be used in the third formalization in the sequence to retract a simplifying assumption made here.[5]

The last two propositions in Figure 1 are also noncausal. The first of these was discussed in Section ; the second states that a person cannot solve a problem if he does not know the area to which it belongs.

The latter portion of the first formalization, shown in Figure 2, includes the facts which are specific to Sam's story. Following the object declarations is a proposition stating that the calculus problem just declared belongs to the area of calculus.

The two declarations following this begin with `:- macros`. Macros are abbreviations in CCALC; throughout the formalization, the term on the left of the arrow stands for the term on the right. The first set of macro definitions gives "friendly" names to the four steps in the

[5]For general commonsense reasoning, it may be appropriate to make every proposition defeasible in this manner. However, the two propositions discussed here are the only ones which are defeated in the formalizations later in the sequence, so for brevity, we have made only these defeasible.

potential history of Sam solving the calculus problem. Steps 0 and 1 refer to the beginning and end, respectively, of the period during which Sam was taking high school math classes. Step 2 is the time at which we ask him to solve a calculus problem, and step 3 is the time at which he has solved it, if he is able to do so.

The second macro definition uses formal parameters `#1`, `#2`, `#3`. It defines what it means for a person to earn a "straight" grade in a subject, as in "straight A's," in terms of the action `earnGrade`. According to this definition,

```
earnStraightGrade(sam,c,highSchoolMath)
```

(Sam earned straight C's in high school math) has the same meaning as

```
[/\G | earnGrade(sam,G,
       highSchoolMath <-> G = c ]
```

(Sam earned a grade in high school math iff that grade was a C).

Finally, we define a query. This posits a sequence of events in which Sam earns straight C's in high school math, but later solves a calculus problem. To answer a query, CCALC reduces it to an instance of the satisfiability problem for propositional logic, as in satisfiability planning (Kautz & Selman 1992; McCain & Turner 1998). In this case, CCALC determines that this query is unsatisfiable; thus, Sam's grades entail his inability to solve a calculus problem. If we remove from the query either the assumption about Sam's grades or the assumption that he solves the problem then the query becomes satisfiable. Histories in which Sam earns *good* grades and then solves the problem and histories in which Sam earns bad grades and then does *not* solve the problem are both consistent with the theory.

Enhancement: Sam Didn't Study Math

The second formalization in the sequence adds to the theory the possibility of a person studying and thereby gaining knowledge in an area, but uses the additional given fact that Sam has not thought about math since he earned C's in high school to conclude that he nevertheless is unable to solve a calculus problem.

Figure 3 shows the additional facts supplied in this enhancement. The first statement,

```
:- include 'sams-1'
```

indicates that this formalization is an extension of the first one in the sequence; all of the declarations and propositions from the latter are included in the former. Then, several new objects and constants are declared. Since the problem statement indicates that Sam has not thought about *math* since high school, without referring to a specific branch, we declare `math` to be an area, but then declare a binary relation `subarea` on areas to relate it to the areas discussed in the first formalization. Also, two new actions are introduced, `think` and `study`, which a person can perform on an area.

The definition of the `subarea` relation just declared follows. By default, two areas are assumed not to be related.

```
:- include 'sams-1'.

:- objects
   math                  :: area.

:- constants
   subarea(area,area)
      :: boolean;
   think(person,area),
   study(person,area)
      :: exogenousAction.

%%%%% Additional facts about %%%%%
%%%%% areas of knowledge     %%%%%

subarea(highSchoolMath,math).
subarea(calculus,math).
default -subarea(A,A2).

%%%%% Studying %%%%%

study(P,A) causes knows(P,A).

study(P,A) causes think(P,A).

think(P,A) causes think(P,A2)
   if subarea(A,A2).

:- query
 label :: 2;
 maxstep :: afterSolvedProblem;
 startOfHighSchool: earnStraightGrade(
    sam,c,highSchoolMath);
 afterLastMathExam: -think(sam,math);
 whenAskedProblem: solve(sam,
    calculusProblem).
```

Figure 3: File `sams-2`.

High school math and calculus are both exceptions, however, and both are declared to be subareas of math.

The remaining propositions define the effects of the new actions. The first states that studying an area causes a person to know that area. The second states that studying an area causes a person to concurrently *think* about that area; here, one action causes another. The final rule indicates that thinking about an area causes a person to think about the superareas of that area; for example, if a person thinks about calculus, it can also be said more generally that he is thinking about math.

With these additions to the theory, the query from the first formalization, which was previously unsatisfiable, becomes satisfiable. CCALC finds histories in which Sam does not know calculus during high school, but studies it afterwards and then solves the problem posed to him. However, the new query adds the additional fact that Sam has not thought about math since he earned the C's in high school. This query is unsatisfiable, since to study calculus, Sam would have had to think about math. Thus, the conclusion that Sam cannot answer a calculus problem is reinforced.

Enhancement: Sam Forgot Whatever He Knew

In the third and final formalization in the sequence, we relax the assumption that Sam's grades in high school necessarily correspond to his knowledge. Perhaps Sam had poor test-taking skills or antagonistic teachers, and he got bad grades even though he learned the material. Thus, we no longer assume that Sam's grades imply that he did not know high school math, or calculus for that matter, during high school. However, we also include a formalization of the passage of time and the tendency of people to forget things over long periods of disuse. This further reinforces our conclusion that Sam is unable to solve a calculus problem.

Figure 4 shows the extension to the formalization. As before, we include the previous formalization, `sams-2`. Also, after some macro definitions, we include the file `time` (Figure 5), discussed below. These macro definitions are parameters to `time`. For example, `maxTime` denotes the maximum number of years covered by our formalization.

The first of the two propositions in Figure 4 states that if a person doesn't think about an area for a long time, he will forget it (won't know it) at the end of that time. The property `longTime` is declared in the file `time` in terms of the macros `largeTime` and `smallTime` defined above. A time interval is long if its length is greater than or equal to `largeTime`, and it is not long if its length is less than or equal to `smallTime`; otherwise we make no such assumptions.

The second proposition declares the fluent

```
abInaccurateGrade(sam,highSchoolMath)
```

to be true at time 0. This fluent appeared in the first formalization, after the keyword `unless` in two propositions (see Figure 1). When the fluent becomes true, it defeats (disables) the instances of these two propositions corresponding to `sam` and `highSchoolMath`. Thus, by asserting `abInaccurateGrade` in the third formalization, we relax the assumption from the first formalization that Sam earned the grades he deserved in high school math.

The formalization of time in Figure 5 declares the integers from 0 to `maxTime` to be of sort `time`, and names a time-valued fluent `currentTime`. The sort `simpleFluent` refers to fluents which are not assumed to be inertial; time does not tend to remain the same. The proposition

```
exogenous currentTime
```

indicates that `currentTime` may arbitrarily change value each step; but the following statement is a constraint which prohibits `currentTime` from decreasing from one step to the next. Thus, the value of `currentTime` may only stay the same or increase every step. The intuitive meaning of `longTime` is discussed above.

Together, the third formalization in the sequence and the formalization of time permit us to further reinforce

```
:- include 'sams-2'.

:- macros
   maxTime   -> 30;
   smallTime ->  2;
   largeTime -> 10.

:- include 'time'.

caused -knows(P,A)
   if currentTime=T
   after -think(P,A) & currentTime=T2 &
      longTime(T-T2)
   where (T-T2 >= 0) &
      (T-T2 =< maxTime).

startOfHighSchool:
   abInaccurateGrade(sam,
   highSchoolMath).

:- query
 label :: 3;
 maxstep :: afterSolvedProblem;
 startOfHighSchool: earnStraightGrade(
    sam,c,highSchoolMath);
 afterLastMathExam: -think(sam,math);
 whenAskedProblem:  solve(sam,
    calculusProblem);
 (whenAskedProblem:currentTime) -
    (afterLastMathExam:currentTime)
    = 20.
```

Figure 4: File `sams-3`.

our conclusion. Since the rules relating knowledge and grades were disabled via the abnormality predicate `abInaccurateGrade`, the query from the second formalization, which was previously unsatisfiable, becomes satisfiable. It is possible that Sam knew calculus in high school despite his bad math grades, and that he hasn't had enough time to forget calculus since then. However, the new query at the end of Figure 4 adds the information that 20 years elapsed between the time Sam earned C's in math and the time he is asked to solve the problem. Since 20 years is a long time by our definition, he is certain to forget calculus in that time even if he knew it in high school. Thus the query is unsatisfiable, indicating that our assumptions yet again entail that Sam cannot solve a calculus problem.

Conclusion

In commonsense reasoning, the conclusions we can draw from a simplified formalization of the world can often be invalidated by representing formerly simplified elements of the formalization in greater detail. However, a sequence of theories, each including and extending the previous one, can demonstrate that a conclusion continues to hold even when additional information is introduced. This sequence of the-

```
:- sorts
   time.

:- objects
   0..maxTime  :: time.

:- variables
   T, T2       :: time.

:- constants
   currentTime :: simpleFluent(time).

exogenous currentTime.
constraint -(currentTime<T)
   after currentTime=T.

:- constants
   longTime(time) :: boolean.

exogenous longTime(T).
constraint -longTime(smallTime).
constraint longTime(largeTime).
constraint longTime(T) ->> longTime(T2)
   where T2 > T.
```

Figure 5: File `time`.

ories represents a *reinforced* solution that persists as the formalization is enhanced to bring it to a closer approximation of reality.

We conjecture that this may be useful as a general methodology for commonsense knowledge representation. Consider the "Opening a Safe," another example from Ernie Davis on the Common Sense Problem Page:

> The combination of a safe consists of 3 numbers between 1 and 50, with a tolerance of plus/minus two. No one knows what the comination is. Infer (a) that it will not be possible to open the safe using the combination within 5 minutes (unless you are very lucky); (b) that it will be possible in a couple of days work.

A reasonable first approximation to the theory of opening safes is that a person cannot open a safe unless he knows the combination. This simplification is sufficient for most day-to-day reasoning and still allows us to reach conclusion (a). However, if we enhance this theory by considering the number of possible combinations, and the rate at which combinations can be tested, we will have a theory that better describes reality, yet still permits us to conclude that the safe cannot be opened in a short time. This gives us increased confidence that our original conclusion was valid in the "real world" and not simply an artifact of our simplified representation.

Acknowledgements

The Sam's Calculus problem was extensively discussed in meetings of the Texas Action Group at Austin in the Spring of 2002. Esra Erdem, Selim Erdoğan, Greg Kuhlmann, and Joohyung Lee proposed alternative solutions, and we are grateful to them and to the other participants for their contributions and criticisms. We have also benefited from conversations with Michael Gelfond, G. Neelakantan Kartha, and John McCarthy on this subject, and from the comments of the anonymous referees on this paper. This work was partially supported by the National Science Foundation under Grant IIS-9732744 and by the Texas Higher Education Coordinating Board under Grant 003658-0322-2001.

References

Davis, E. 1990. *Representations of Commonsense Knowledge*. Morgan Kaufmann.

Gabbay, D.; Hogger, C.; and Robinson, A., eds. 1994. *The Handbook of Logic in AI and Logic Programming*, volume 3. Oxford University Press.

Giunchiglia, E., and Lifschitz, V. 1998. An action language based on causal explanation: Preliminary report. In *Proc. AAAI-98*, 623–630. AAAI Press.

Giunchiglia, E.; Lee, J.; Lifschitz, V.; McCain, N.; and Turner, H. 2003. Nonmonotonic causal theories.[6] *Artificial Intelligence*. To appear.

Kautz, H., and Selman, B. 1992. Planning as satisfiability. In *Proc. ECAI-92*, 359–363.

Lifschitz, V. 2000. Missionaries and cannibals in the Causal Calculator. In *Principles of Knowledge Representation and Reasoning: Proc. Seventh Int'l Conf.*, 85–96.

McCain, N., and Turner, H. 1997. Causal theories of action and change. In *Proc. AAAI-97*, 460–465.

McCain, N., and Turner, H. 1998. Satisfiability planning with causal theories. In Cohn, A.; Schubert, L.; and Shapiro, S., eds., *Proc. Sixth Int'l Conf. on Principles of Knowledge Representation and Reasoning*, 212–223.

McCarthy, J., and Hayes, P. 1969. Some philosophical problems from the standpoint of artificial intelligence. In Meltzer, B., and Michie, D., eds., *Machine Intelligence*, volume 4. Edinburgh: Edinburgh University Press. 463–502.

McCarthy, J. 1999. Elaboration tolerance.[7] In progress.

[6] `http://www.cs.utexas.edu/users/vl/papers/nmct.ps` .

[7] `http://www-formal.stanford.edu/jmc/elaboration.html` .