Model-Checking Memory Requirements of Resource-Bounded Reasoners

A. Albore[1] **N. Alechina**[2] **P. Bertoli**[3] **C. Ghidini**[3] **B. Logan**[2] **L. Serafini**[3]

[1] Universitat Pompeu Fabra
pg.Circumval·lació 8
08003, Barcelona
Spain
albore@gmail.com

[2] School of Computer Science
University of Nottingham
Nottingham
NG8 1BB, UK
{nza,bsl}@cs.nott.ac.uk

[3] ITC-irst
via Sommarive 18
Povo, 38050,
Trento, Italy
{bertoli,ghidini,serafini}@itc.it

Abstract

Memory bounds may limit the ability of a reasoner to make inferences and therefore affect the reasoner's usefulness. In this paper, we propose a framework to automatically verify the reasoning capabilities of propositional memory-bounded reasoners which have a sequential architecture. Our framework explicitly accounts for the use of memory both to store facts and to support backtracking in the course of deductions. We describe an implementation of our framework in which proof existence is recast as a strong planning problem, and present results of experiments using the MBP planner which indicate that memory bounds may not be trivial to infer even for simple problems, and that memory bounds and length of derivations are closely interrelated.

Introduction

The question of how much memory a reasoner needs to derive a formula is of considerable theoretical and practical interest. From a theoretical point of view, it is interesting to investigate how the deductive strength of a particular logic changes when only a fixed number of formulas are allowed to be 'active' in a derivation. This is related to research in the space complexity of proofs, see e.g. (Alekhnovich *et al.* 2002). Another interesting theoretical challenge is to design an epistemic logic where properties of bounded memory reasoners can be expressed.

From a practical point of view, the question of whether a reasoner has sufficient memory to achieve its goals is a major concern for a developer. For example, the non-trivial reasoning capabilities assumed by many agent-based web service applications (e.g., reasoning over complex ontologies or about business processes described by a set of business rules) makes it hard for a developer to determine an agent's memory requirements a priori.

In this paper, we describe the implementation of a formal framework for verifying memory requirements of reasoning agents. The main idea of our approach is to model a reasoning agent as a transition system where states correspond to the set of formulas held in the agent's memory, and transitions correspond to applications of the agent's inference rules. The agent's memory is bounded; we model this by assuming that at any time, the agent can remember at most n formulas. We say that the agent has sufficient memory to derive a formula A if there is a sequence of choices (of which inference rules to apply, which formulas to overwrite, etc.) which 'reaches A' in the transition system. For some agents, for example those which reason using resolution or similar deterministic rules, the notion of what it means to 'reach A' is simple: from the start state, there exists a path to a state where A is in memory. For agents which reason by cases, such as the ones we concentrate on in this paper, A has to be reachable on *all* branches starting from the state where the case split occurred. Backtracking to investigate the pending branch requires storing the state and the pending formula on the stack and costs memory; we will assume that the contents of the stack are included in the memory limit n.

The rest of the paper is structured as follows. We explain our approach on a simple example. Then we describe the language and models of a novel epistemic logic which allows us to formulate properties of bounded memory reasoners. We show how to use the MBP planner to verify those properties, and describe some experiments. Finally, we briefly discuss related work and conclude.

Intuitive Model and Problem Statement

Consider a reasoning agent which is attempting to derive a sentence (goal formula) ϕ from a set of formulas stored in a knowledge base K. We assume that K is too large to fit in the agent's memory, and the agent can store at most *n* formulas in memory at any given time. We model the agent's memory as a pool of n memory cells, each of which is assumed to hold a single formula (i.e., we abstract away from the size of the formulas). This idealisation is made to simplify the analysis. However, even with this idealisation, it is not possible to derive all consequences using a memory of constant size, as the inference rules we consider do not include an unrestricted conjunction introduction rule. Applying an inference rule or loading new data from Kwhen the agent's memory is full, overwrites the contents of one or more memory cells. Cells from the pool may be dynamically allocated to the *stack*. The stack stores the information about branches of the proof tree which still need to be explored, and is used to control backtracking when reasoning by cases. When the application of an inference rule to a formula creates a choice point (e.g., the application of an 'or elimination' inference rule), the cell containing the formula is pushed on the stack. Formulas on the stack are *locked* and can't be overwritten by the application of inference rules or reading a formula from K. In addition, the reasoner may optionally add to the stack, and therefore lock, any formulas which are in memory at a choice point. Locking a formula preserves it in memory until it is unlocked, and provides the reasoner with a means of ensuring that this formula will be available on all branches of the search tree.

A Simple Example

To informally illustrate the reasoning process, we show a simple example, where a reasoner with 3 memory cells intends to derive fact D, starting from a knowledge base composed of the following facts:

$$A, \quad A \to (B \vee C), \quad B \to D, \quad (C \wedge A) \to D$$

Table 1 shows a derivation of D which uses only three cells.

#	Applied rule	Memory content	Stack
1	read(A)	$\{A\}$	$\emptyset$
2	read$(A \to B \vee C)$	$\{A,\ A \to B \vee C\}$	$\emptyset$
3	MP	$\{A,\ B \vee C\}$	$\emptyset$
4	$\vee_E$, branch=right	$\{A,\ C\}$	$\{B\}$
5	$\wedge_I$	$\{A,\ C \wedge A\}$	$\{B\}$
6	read$(C \wedge A \to D)$	$\{C \wedge A \to D,\ C \wedge A\}$	$\{B\}$
7	MP	$\{D,\ C \wedge A\}$	$\{B\}$
8	read$(B \to D)$	$\{B,\ B \to D\}$	$\emptyset$
9	MP	$\{B,\ B \to D,\ D\}$	$\emptyset$

Table 1: A proof of D (right-first).

In steps 1 and 2, A and $A \to (B \wedge C)$ are loaded into memory, so that at step 3 we can infer $B \vee C$ by Modus Ponens. At step 4, $\vee$-elimination requires reasoning by cases; the reasoner decides to consider the right branch first, i.e. C, and also decides that A will not be used when backtracking to the left branch. This means that A and C are in memory as (overwriteable) facts, while B is in the "stack" part of the memory and cannot be overwritten. Steps 5,6,7 then derive D by first conjoining A and C, and then using Modus Ponens over a formula read from K. Now backtracking takes place, and the stack content (B) is popped up and put in memory; step 8 reads $B \to D$ from K, and step 9 finally applies Modus Ponens to derive D on this branch. The proof is depicted in Fig. 1, left, indicating with shaded cells the ones used as stack. Notice that the decision at step 4 not to lock A for possible use on the pending branch is crucial: if we are unable to overwrite A, then the derivation runs out of memory at step 7. Similarly, the way facts are overwritten in steps 3,4,5,6 (a choice left implicit in the table) is essential to limit memory usage. Note also that if we had decided to consider the left branch first at step 4 would have led either to a longer proof, or to one that requires more memory. Fig. 1, right, shows a derivation of D going left-first with the same number of steps, but using 4 cells of memory: A (as well as C) is saved on the stack to have it immediately available when backtracking, incrementing the needed memory on the left branch.

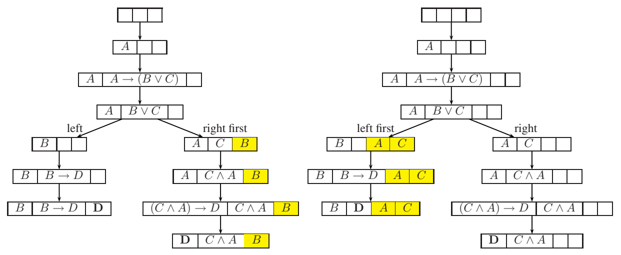

Figure 1: Proofs of D: right-first, left-first

The Formal Model

In this section, we define the language and the models of our logic for verifying bounded memory reasoners, BML (for bounded memory logic). We assume we have: an agent with an internal language L and a set of inference rules $\mathcal{R}$; the agent's knowledge base K; and the goal formula A_G. We denote by Ω the set of all subformulas of $K \cup \{A_G\}$. States can be identified with assignments to formulas from Ω and the contents of the stack. Transitions correspond to reading formulas from K, applying inference rules from $\mathcal{R}$ to the formulas in the state, and manipulating the stack.

The language of the logic BML is defined relative to the agent's internal language L. There are two kinds of well formed formulas: world formulas and state formulas, which are defined as follows:

- If A is a formula of L, then $\mathrm{B}\,A$ (the agent believes A) and $\bar{\mathrm{B}}\,A$ (the agent disbelieves A) are world formulas; in addition, a nullary modality (k), for any natural number k, is a world formula (which states that the number of formulas on the stack is k).[1]
- If ϕ is a world formula, then $\Diamond\phi$ ('it is possible that ϕ') is a state formula;
- if ϕ is a state formula, then $\neg\phi$ and $\mathrm{EF}\,\phi$ ('in some future state, ϕ') are state formulas;
- If ϕ_1 and ϕ_2 are state formulas, then $\phi_1 \wedge \phi_2$ is a state formula.

Other boolean connectives are defined in the usual way. We also define $\Box\phi$ as $\neg\Diamond\neg\phi$.

Below we define a formal model for a classical reasoner. Formal models for different kinds of reasoners can be defined similarly. A BML transition system for a classical reasoner, $M = (S, W, R, Y, T, F, c, L)$, consists of a set of states S, a set of possible worlds or epistemic alternatives W, a binary relation R on S, a function assigning to each state a set of epistemic alternatives $Y : S \longrightarrow \mathcal{P}(W)$ (intuitively, sets of formulas a reasoner by cases considers possibly true when in state s), and two assignments $T : W \longrightarrow \mathcal{P}(\Omega)$ and $F : W \longrightarrow \mathcal{P}(\Omega)$ which say which formulas of the internal language are true or false in w. The truth assignments should be consistent, i.e., $T(w) \cap F(w) = \emptyset$. We will say that a formula A is *defined in* w if $A \in T(w) \cup F(w)$. c is a function from W into natural numbers, assigning every $w \in W$ the number of pending formulas on the stack

[1]Similar modality $min(n)$ was introduced in (Ågotnes & Alechina 2006).

in that world. $L : S \longrightarrow \mathcal{P}(W)$ assigns each state the set of formulas which are locked in that state; we require that $L(w) \subseteq T(w) \cup F(w)$. For each w, the memory restriction is expressed by $|T(w)| + |F(w)| + c(w) \leq n$.

Transitions between states correspond to applying the agent's inference rules. Below we list transitions characterising a classical reasoner which uses the connectives $\neg$ and $\vee$. The transitions are of two kinds, branching and non-branching. A state s can transit to a state t by a non-branching transition **X** if the precondition of **X** holds in some $w \in Y(s)$, the postcondition of **X** holds in some $w' \in Y(t)$, and $Y(t) = (Y(s) \setminus \{w\}) \cup \{w'\}$. The postcondition of all non-branching transitions also includes a standard proviso that $L(w') = L(w)$, $c(w') = c(w)$ and that all formulas which were defined in w have the same truth value in w', apart from possibly one formula B, where B is defined in w, $B \notin L(w)$, and B is undefined in w' (i.e., some formula B in w which is not locked is overwritten in w').

The non-branching transitions for the classical reasoner are:

Read *precondition*: some formula $A \in K$ is undefined in w; *postcondition*: $A \in T(w')$.

makeNot *precondition*: some A is defined in w, and $\neg A$ is not; *postcondition*: $\neg A$ is defined in w' and has the opposite truth value to A.

elimNot *precondition*: some $\neg A$ is defined in w, and A is not; *postcondition*: A is defined in w' and has the opposite truth value to $\neg A$.

ExContradictio *precondition*: some A and $\neg A$ have the same truth value in w; *postcondition*: $A_G \in T(w')$.

makeOr *precondition*: A, B are defined in w; *postcondition*: $A \vee B$ is defined in w' and its truth value is the 'or' of the truth values of A, B.

elimOr$_\mathbf{F}$ *precondition*: $A \vee B \in F(w)$; *postcondition*: $A, B \in F(w')$; note that in this case the standard proviso is modified in that *two* formulas in w may be overwritten in w'.

A state s can transit to a state t by a branching transition **X** if the precondition of **X** holds in some $w \in Y(s)$, the postcondition of **X** holds in some $w_1, w_2 \in Y(t)$, and $Y(t) = (Y(s) \setminus \{w\}) \cup \{w_1, w_2\}$. Each branching transition has two versions: left (w_1) first or right (w_2) first. If the w_1 branch is explored first, then $c(w_1) = c(w) + 1$ (stack is incremented), $L(w_1) = L(w) \cup T(w_2) \cup F(w_2)$ (formulas which need to be remembered for the w_2 branch are locked in w_1, in addition to formulas which were locked in w); $L(w_2) = L(w)$ and $c(w_2) = c(w)$. All formulas which are defined in w are defined and have the same value in w_1, apart from possibly one non-locked formula which may be overwritten. w_2 contains some subset of formulas from w; we can think of it as allowing an arbitrary number of formulas to be overwritten or forgotten on the pending branch. The case for w_2 branch being explored first is symmetrical.

The branching conditions for the classical reasoner are:

elimOr$_\mathbf{T}$ *precondition*: $A \vee B \in T(w)$; *postcondition*: $A \in T(w_1)$, $B \in T(w_2)$.

Split *precondition*: some formula A is undefined in w; *postcondition*: $A \in T(w_1)$ and $A \in F(w_2)$ (this is essentially reasoning by cases on some formula A).

Any transition is disabled if it would lead to some A being both in $T(w)$ and $F(w)$ for some alternative w or violate the memory size constraint.

The satisfaction relation $\models$ is defined inductively; note that world formulas are evaluated relative to a world and state formulas relative to a state:

$M, w \models \mathrm{B}\, A$ iff $A \in T(w)$

$M, w \models \bar{\mathrm{B}}\, A$ iff $A \in F(w)$

$M, w \models (k)$ iff $c(w) = k$

$M, s \models \Diamond\phi$ iff there exists w in $Y(s)$, such that $M, w \models \phi$

$M, s \models \neg\phi$ iff $M, s \not\models \phi$

$M, s \models \phi \wedge \psi$ iff $M, s \models \phi$ and $M, s \models \psi$

$M, s \models \mathrm{EF}\, \phi$ iff there exists a sequence of states $t_1, \ldots, t_k$ such that for all $i \in \{1, \ldots, k-1\}$, $R(t_i, t_{i+1})$, $t_1 = s$ and $M, t_k \models \phi$

Theorem 1 *The transitions for the classical agent are deductively sound and complete: for every classical consequence A of K, there exists a memory bound n such that an agent with memory bound n can derive A, and only the logical consequences of K are derivable.*

The proof is omitted due to the lack of space.

In the language of BML, the bound n on the size of the agent's memory is expressed by the following axiom schema:

SB(n) $\Box\big((n-k) \wedge \widetilde{\mathrm{B}}\, A_1 \wedge \ldots \wedge \widetilde{\mathrm{B}}\, A_k \rightarrow \neg\widetilde{\mathrm{B}}\, A_{k+1}\big)$, where $\widetilde{\mathrm{B}}\, A_i$ stands for either $\mathrm{B}\, A_i$ or $\bar{\mathrm{B}}\, A_i$ and $A_i \neq A_j$ for all $i, j \in \{1, \ldots, k+1\}$ such that $i \neq j$.

The axiom schema says that if there are $n-k$ formulas on the stack, then at most k different formulas can be in memory.

Finally, we can formulate the verification problem as a logical problem: *A classical reasoner agent can derive a formula A_G from a knowledge base K iff in the classical reasoner transition system, the start state s_0 with $Y(s_0) = \{w_0\}$, $T(w_0) = F(w_0) = \emptyset$ and $c(w_0) = 0$ satisfies the BML formula* $\mathrm{EF}\, \Box \mathrm{B}\, A_G$.

Verifying Reasoning Capabilities Using MBP

The problem of identifying the existence (and the minimal length) of a deduction for A_G from a knowledge base K, for a reasoner with n memory cells modelled as a transition system M can be recast as a problem of *strong planning* (Cimatti *et al.* 2003): find a tree-structured plan π which, starting from any state in K, leads M to some state satisfying A_G. The plan π is indeed the proof of A_G.

In this section, we provide a high-level description of the way we recast the proof existence problem using the MBP planner (Cimatti *et al.* 2003), whose input language allows M to be described as a set of propositional constraints over finitely-valued variables.

Formally, M is encoded as a planning domain, i.e. a triple $(\Sigma, \mathcal{A}, \mathcal{T})$, where Σ are the states of the domain, $\mathcal{A}$ is a set of actions, and $\mathcal{T} \subseteq \Sigma \times \mathcal{A} \times \Sigma$ is the transition relation, describing the outcomes of action execution. Essentially, in our mapping, actions represent inference rules, and domain states represent epistemic states of the reasoner, represented by a set of *state variables*. In our case, we will have a set V of $|\Omega|$ three-valued state variables to represent the T, F assignments in the formal model. We will denote by $V(\phi)$ the value of the variable associated with $\phi \in \Omega$. $V(\phi)$ corresponds to the known value of ϕ ($\top$ or $\bot$), or indicates that its value is unknown (U). The boolean predicates $\top(\phi)$, $\bot(\phi)$, $U(\phi)$ are shortcuts for $V(\phi) = \top$, $V(\phi) = \bot$, $V(\phi) = U$ respectively. Associated with formula ϕ, we also have a bit $L(\phi) \in \{\top, \bot\}$ that represents whether ϕ is on the stack, and thus locked to writing.

Moreover, since, in our model, alternatives are generated in parallel, we use an integer $S \in [0, n]$ to indicate, in a state, the number of formulas on the stack that belong exclusively to alternative states, and will be used when backtracking. For instance, the two states produced by step 4 of Table 1 would be represented with $V(A) = \top \wedge V(C) = \top \wedge (S = 1)$ and $V(B) = \top \wedge (S = 0)$.

The actions of the domain represent the choices that the reasoner must perform at each inference step, that is:

- which, amongst the inference rules applicable in the current state of knowledge, to apply;
- whether the newly derived fact(s) should overwrite some currently known facts in memory, and if so, which fact(s) to overwrite;
- in case the rule produces more than one possible world, i.e. it branches, in which order to analyse the branches;
- in case the rule branches, which facts should be locked to writing so that they are available when backtracking to the second branch.

Correspondingly, actions feature (a) one argument in Ω representing the formula to be read, split, or composed, (b) one or two additional arguments in $\Omega' = \Omega \cup \{A_0\}$, indicating the formula(s) to be overwritten, and the fictitious formula A_0 if no rewriting must occur, (c) for "branching" actions, one boolean argument d indicating the "direction" of the search: left or right branch first, (d) for "branching" actions, $|\Omega|$ boolean arguments indicating whether the i-th fact should be saved on stack for use in backtracking, and thus locked to writing.

This defines the range of the action variable α in the planning domain. In the following, for reasons of space, we define the range of α considering just one of the non-branching rules, and one of the branching rules:

$$\alpha \in \bigcup_{\substack{A \in \Omega \\ B \in \Omega' \setminus \{A\}}} \mathbf{Read}(A, B) \cup \bigcup_{\substack{A \in \Omega \\ B \in \Omega' \setminus \{A\} \\ d, s_1, \ldots, s_{|\Omega|} \in \{\top, \bot\}}} \mathbf{Split}(A, B, d, s_1, \ldots, s_{|\Omega|})$$

Regarding their executability preconditions, in addition to the constraints specific to each rule, we specify (by a constraint $\rho(B)$) that we may decide to overwrite fact B only if its value is currently known:

$$\rho(B) := (B = A_0 \vee \neg U(B))$$

and we specify (by a constraint $\lambda(B)$) that the reasoner can lock a formula only if it is already in memory and it is not going to be overwritten:

$$\lambda(B) := \forall i \in [1, |\Omega|] : s_i \rightarrow (\neg U(A_i) \wedge \neg L(A_i) \wedge A_i \neq B)$$

Also, we prevent the application of rules whose result is known already. Thus, the precondition for $\mathbf{Read}(A, B)$ is $\rho(B) \wedge U(A)$ and the precondition for $\mathbf{Split}(A, B, d, s_1, \ldots, s_{|\Omega|})$ is $\rho(B) \wedge U(A) \wedge \lambda(B)$.

The effects of an action over the formulas in Ω are partitioned into effects on derived formulas, overwritten formulas, the rest of the state, and the stack. The way the derived formula(s) are affected is encoded as follows. Below, X is read 'in the next state'.

$$\alpha = \mathbf{Read}(A, B) \rightarrow X(\top(A))$$

$$\alpha = \mathbf{Split}(A, B, d, s_1, \ldots, s_{|\Omega|}) \rightarrow X(\top(A) \vee \bot(A))$$

The removal of overwritten fact(s) from memory is easily represented:

$$\alpha = \mathbf{Read}(A, B) \rightarrow \big(B = A_0 \vee X(U(B))\big)$$

$$\alpha = \mathbf{Split}(A, B, d, s_1, \ldots, s_{|\Omega|}) \rightarrow \big(B = A_0 \vee X(U(B))\big)$$

In the case of non-branching rules, the fact that the remaining formulas are not affected can be easily represented. Let $\iota(\phi) := X(V(\phi)) = V(\phi)$:

$$\alpha = \mathbf{Read}(A, B) \rightarrow \forall A' \notin \{A, B\} : \iota(A')$$

For a branching rule, things are more complicated since the behaviour is different on the two branches: in particular, on the pending branch, we have to keep only those facts that have been locked (now or previously), and forget the others. We use a formula β^1_{cs} that, based on the outcomes of the action and on the value of the direction flag d, recognises whether we are considering the first or pending branch.

$$\beta^1_{cs} := (d \wedge X(V(A)) = \top) \vee (\neg d \wedge X(V(A)) = \bot))$$

$$\alpha = \mathbf{Split}(A, B, d, s_1, \ldots, s_{|\Omega|}) \wedge \beta^1_{cs} \rightarrow$$
$$\forall A' \notin \{A, B\} : \iota(A')$$

$$\alpha = \mathbf{Split}(A, B, d, s_1, \ldots, s_{|\Omega|}) \wedge \neg\beta^1_{cs} \rightarrow$$
$$\forall A' \notin \{A, B\} : (s_i \vee L(A_i)) \rightarrow \iota(A_i)$$

$$\alpha = \mathbf{Split}(A, B, d, s_1, \ldots, s_{|\Omega|}) \wedge \neg\beta^1_{cs} \rightarrow$$
$$\forall A' \notin \{A, B\} : \neg(s_i \vee L(A_i)) \rightarrow X(U(A_i))$$

In addition, we have to represent that our knowledge remains consistent over time, i.e., we do not allow true facts become false and vice versa. To this end, for each $A \in \Omega$, we add two constraints of the form $\top(A) \rightarrow X(\neg\bot(A))$ and $\bot(A) \rightarrow X(\neg\top(A))$.

The way the stack evolves is represented by the "write lock" bits and the S counter. For non-branching actions, this is trivial, as the stack keeps its content:

$$\alpha = \mathbf{Read}(A, B) \rightarrow (X(\neg L(A)) \wedge X(\neg L(B)) \wedge$$
$$\forall A' \notin \{A, B\} : X(L(A')) = L(A'))$$

$$\alpha = \mathbf{Read}(A, B) \rightarrow X(S) = S$$

Again, for branching actions, things are more complicated because we have to treat each of the branches differently. In the first branch, formulas are locked if previously locked, or if the current action locks them; moreover, one formula is added to the stack that must be considered exclusively for backtracking.

$$\alpha = \mathbf{Split}(A, B, d, s_1, \ldots, s_{|\Omega|}) \wedge \beta^1_{cs} \rightarrow$$
$$\forall A_i : X(L(A_i)) = ((L(A_i) \vee s_i) \wedge A_i \notin \{A, B\})$$
$$\alpha = \mathbf{Split}(A, B, d, s_1, \ldots, s_{|\Omega|}) \wedge \beta^1_{cs} \rightarrow X(S) = S + 1$$

In the pending branch, the stack stays the same.

$$\alpha = \mathbf{Split}(A, B, d, s_1, \ldots, s_{|\Omega|}) \wedge \neg\beta^1_{cs} \rightarrow$$
$$\forall A_i : X(L(A_i)) = (L(A_i) \wedge s_i \wedge A_i \notin \{A, B\})$$
$$\alpha = \mathbf{Split}(A, B, d, s_1, \ldots, s_{|\Omega|}) \wedge \neg\beta^1_{cs} \rightarrow X(S) = S$$

The modelling of other actions is essentially analogous.

Given the encoding above, the planning problem is described by an initial state where $\forall A \in \Omega : U(A)$, a goal state $\top(A_G)$, and the memory bound is enforced by a constraint of the form $S + |\{A : \neg U(A)\}| \leq n$.

We remark that the planning domain $(\Sigma, \mathcal{A}, \mathcal{T})$ represents the way the reasoner evolves its epistemic state during proof construction, in a way that is completely decoupled from the search performed by the planner over possible deductions. That is, our planner is not forced to pursue a depth-first search strategy; indeed, among the many possible search strategies supported by MBP, we have adopted MBP's breadth-first backward search to guarantee that the plan we discover is optimal in terms of the number of execution steps. The computational burden imposed by such a search strategy is effectively constrained by the use of symbolic representation techniques that allow a very compact encoding, and an efficient handling of extremely large state sets; details can be found in (Cimatti *et al.* 2003).

Experiments

In this section we consider some examples illustrating the relationship between the memory bound and the length of derivation, and the impact of different strategies of locking on memory usage.

Suppose a designer of a system which uses a knowledge base K containing the formulas

$$X_1 \vee X_2, X_1 \rightarrow Y_1, X_2 \rightarrow Y_2 \qquad (1)$$

wishes to verify that the system, which is running on a PDA with a memory of size n, can infer $Y_1 \vee Y_2$ from K. In addition, the designer may be interested in how increases in memory size affect the number of steps required for the derivation, e.g., if they wish to trade memory for response time.

Using MBP, we verified that deriving the target formula requires a memory of at least size 2. For memory size of size 1 MBP quickly determines that there are no possible derivations of the target formula. With 2 memory cells the shallowest derivation tree has depth 7. With a memory of size 3, the depth of the tree drops to 5. Further increases in the amount of memory do not result in further reductions in the length of derivation.

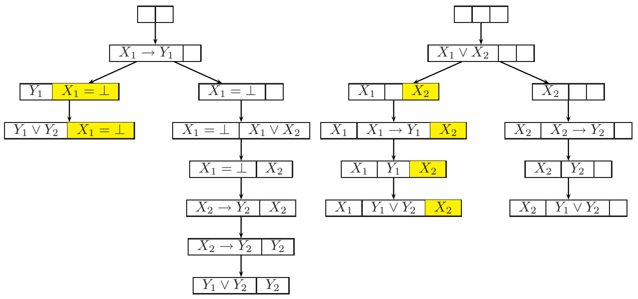
$X_1 \rightarrow Y_1$
Y_1 | $X_1 = \bot$
$Y_1 \vee Y_2$ | $X_1 = \bot$
$X_1 = \bot$
$X_1 = \bot$ | $X_1 \vee X_2$
$X_1 = \bot$ | X_2
$X_2 \rightarrow Y_2$ | X_2
$X_2 \rightarrow Y_2$ | Y_2
$Y_1 \vee Y_2$ | Y_2
$X_1 \vee X_2$
X_1 | X_2
X_1 | $X_1 \rightarrow Y_1$ | X_2
X_1 | Y_1 | X_2
X_1 | $Y_1 \vee Y_2$ | X_2
X_2
X_2 | $X_2 \rightarrow Y_2$
X_2 | Y_2
X_2 | $Y_1 \vee Y_2$

Figure 2: Two deductions of $Y_1 \vee Y_2$

Figure 2 shows a pictorial representation of two derivations of $Y_1 \vee Y_2$ found by the MBP planner. The tree on the left shows the shallowest derivation with 2 memory cells, while the tree on the right depicts a derivation which uses 3 cells. For lack of space we represent only the formulas in memory and omit the actions. In both cases the left branch of the tree is explored first. Also, we omit the truth value of positive (true) formulas and specify only the truth value $\bot$ of formulas. It is interesting to note that while the intuitive proof of $Y_1 \vee Y_2$ which starts from the initial formula $X_1 \vee X_2$, explores the two possible cases (X_1 true, and X_2 true) in parallel and concludes in both cases that $Y_1 \vee Y_2$ is true, results in the shortest derivation tree (as shown on the right of Figure 2), it is not the optimal deduction in terms of memory size. In fact the application of Modus Ponens requires two memory cells in both branches, and the initial splitting requires 1 cell of the stack to remember the formula X_2 which represents the alternative case of the initial disjunction. Using MBP, we were able to find the optimal strategy in terms of memory usage (depicted on the left of Figure 2). This strategy first considers the formula $X_1 \rightarrow Y_1$ and examines the two alternative possible models which make this formula true, that is, Y_1 true and X_1 false. In the first case the system can infer immediately the goal using only two memory cells (one for the formula used in the deduction process, and one for the stack (the left branch of the tree). In the second case a (long) proof exists of the goal which uses only two memory cells (the right branch of the tree).

Let us consider now the following two formulas:

$$(X_1 \wedge X_2) \vee (X_1 \wedge \neg X_2) \vee \neg X_1 \qquad (2)$$

$$(X_1 \wedge X_2) \vee (X_1 \wedge \neg X_2) \vee (\neg X_1 \wedge X_2) \vee (\neg X_1 \wedge \neg X_2) \qquad (3)$$

These are logically true formulas, and thus can be derived by all (logically sound and complete) reasoners despite the content of their knowledge base. However, the memory requirements for (2) and (3) as identified by MBP are quite different, as shown in table 2 and they also depend upon different strategies of locking used by the reasoners. In this case we have examined 'store all' reasoners, which save the entire content of the state on the stack, and 'smart store' reasoners which guess which formulas to lock.

Let us consider 'smart store' reasoners first. Deriving the target formulas requires memory of at least size 3 for both

	Memory size	Smart store	Store all
Equation (2)	1	no proof	no proof
	2	no proof	no proof
	3	6 steps	6 steps
	$\geq$4	5 steps	5 steps
Equation (3)	1	no proof	no proof
	2	no proof	no proof
	3	7 steps	no proof
	$\geq$4	6 steps	6 steps

Table 2: Results for Equations (2) and (3)

formulas, but while for equation (2) the proof with 3 memory cells produces trees of depth 6, for equation (3) we have trees of depth 7. Similar differences exist for the case of memory of size 4. If we consider 'store all' reasoner, the difference between the two formulas is even more remarkable. In this case not only the depth of the proofs is different, but also the minimum amount of memory required to prove the goal (3 memory cells for Equation (2) and 4 memory cells for Equation (3)). Thus memory requirements can change for two logically equivalent goals, and similarly for logically equivalent knowledge bases or sets of inference rules (we omit examples due to the lack of space).

Related Work

Our work is related to other work on logics of knowledge and belief, for example (Fagin *et al.* 1995). However, most epistemic logics assume that the reasoner's knowledge is deductively closed, and therefore do not try to model the time and space restrictions on the reasoner's ability to derive consequences of its beliefs.

Both time and space limitations on a reasoner's knowledge were considered in *step logics* (Elgot-Drapkin, Miller, & Perlis 1991). However, rather than being interested in verification of space requirements for solving a certain problem, they are concerned with restricting the size of memory to isolate any possible contradictions, thus avoiding the problem of *swamping*: deriving all possible consequences from a contradiction.

Our work is related to work on formal verification of multi agent systems (Benerecetti, Giunchiglia, & Serafini 1998; Kacprzak, Lomuscio, & Penczek 2004). However, work in this area is mainly focused on logically omniscient reasoners and time and space limitations are not taken into account.

The connection between deduction and planning has long been established for a variety of logics, e.g., temporal, linear and propositional logics, see (Jacopin 1993; Bibel 1986; Kautz & Selman 1996). Existing work, however, has focused on using effective theorem provers to build plans, rather than on exploiting a planner to construct a derivation. To the best of our knowledge, ours is the first experiment in this direction.

Worst case bounds on space complexity of proofs are studied in proof complexity (Alekhnovich *et al.* 2002). For proof systems such as resolution which do not have an unrestricted conjunction introduction rule, the size of input is taken to be the number of clauses (formulas). For proof systems with an unrestricted conjunction introduction rule this would lead to constant space complexity, so the size of input is taken to be the number of variables.

Conclusion

In this paper, we have explored the implications of the idea that reasoning is a process which requires memory, and developed a framework for representing and verifying memory-bounded reasoners. While the temporal aspect of reasoning has been studied before, we believe that our treatment of the memory aspect is novel. We have proposed a new kind of epistemic logic where memory is explicitly modelled. The logic is interpreted on state transition systems, where the reasoner's state can contain only a fixed finite number of formulas (beliefs), and transitions correspond to application of inference rules by the agent. By specifying the state transition system as an input to the MBP planner, we can automatically verify the lower bounds on memory required by the reasoner to derive a certain formula.

In future work, we plan to model multi-agent systems, including agents which are capable of reasoning about other agents' beliefs, and reasoners in description logics. We also plan to investigate models of memory which take the size of formulas into account, rather than just the number of formulas.

Acknowledgements This work was supported by the Royal Society UK-Italy Joint Project grant 'Model-checking resource-bounded agents'.

References

Ågotnes, T., and Alechina, N. 2006. Knowing minimum/maximum n formulae. In *Proc. of ECAI-06*. (To appear).

Alekhnovich, M.; Ben-Sasson, E.; Razborov, A. A.; and Wigderson, A. 2002. Space complexity in propositional calculus. *SIAM J. Comput.* 31(4):1184–1211.

Benerecetti, M.; Giunchiglia, F.; and Serafini, L. 1998. Model Checking Multiagent Systems. *Journal of Logic and Computation, Special Issue on Computational & Logical Aspects of Multi-Agent Systems* 8(3):401–423.

Bibel, W. 1986. A deductive solution for plan generation. *New Generation Computing* 4:115–132.

Cimatti, A.; Pistore, M.; Roveri, M.; and Traverso, P. 2003. Weak, Strong, and Strong Cyclic Planning via Symbolic Model Checking. *Artificial Intelligence* 147(1,2):35–84.

Elgot-Drapkin, J.; Miller, M.; and Perlis, D. 1991. Memory, reason and time: the Step-Logic approach. In *Philosophy and AI: Essays at the Interface*. Cambridge, Mass.: MIT Press. 79–103.

Fagin, R.; Halpern, J. Y.; Moses, Y.; and Vardi, M. Y. 1995. *Reasoning about Knowledge*. Cambridge, Mass.: MIT Press.

Jacopin, E. 1993. Classical AI planning as theorem proving: The case of a fragment of linear logic. In *AAAI Fall Symposium on Automated Deduction in Nonstandard Logics*. AAAI Press.

Kacprzak, M.; Lomuscio, A.; and Penczek, W. 2004. Verification of multiagent systems via unbounded model checking. In *Proc. of AAMAS'04*, 638–645. New York: ACM.

Kautz, H., and Selman, B. 1996. Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proc. of AAAI'96*, 1194–1201. AAAI Press.