Sum and Product in Dynamic Epistemic Logic*

H.P. van Ditmarsch[†], J. Ruan[‡], and R. Verbrugge[§]

Abstract

The Sum-and-Product riddle was first published in [Fre69]. We provide an overview on the history of the dissemination of this riddle through the academic and puzzle-math community. This includes some references to precursors of the riddle, that were previously (as far as we know) unknown.

We then model the Sum-and-Product riddle in a modal logic called public announcement logic. This logic contains operators for knowledge, but also operators for the informational consequences of public announcements. The logic is interpreted on multi-agent Kripke models. The information in the riddle can be represented in the traditional way by number pairs, so that Sum knows their sum and Product their product, but also as an interpreted system, so that Sum and Product at least know their local state. We show that the different representations are isomorphic. We also provide characteristic formulas of the initial epistemic state of the riddle. We analyze one of the announcements towards the solution of the riddle as a so-called unsuccessful update: a formula that becomes false because it is announced.

The riddle is then implemented and its solution verified in the epistemic model checker DEMO. This can be done, we think, surprisingly elegantly. The results are compared with other work in epistemic model checking and the complexity is experimentally investigated for several representations and parameter settings.

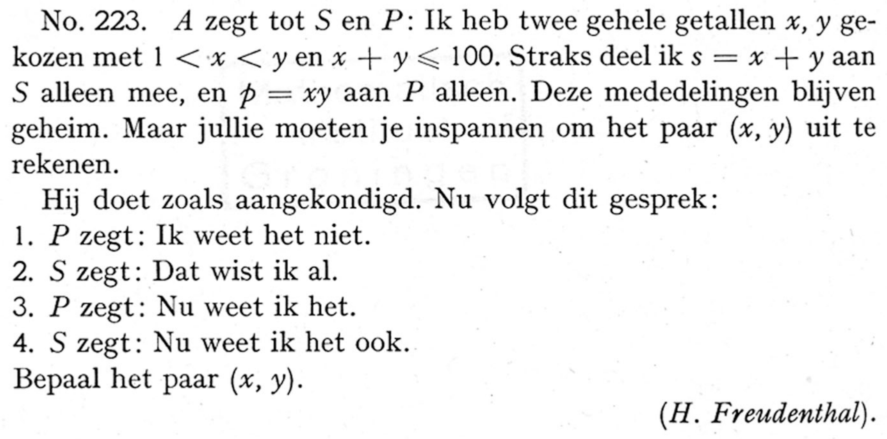
Keywords: modal logic, puzzle math, dynamic epistemic logic, characteristic formula, model checking

*Contact author is Hans van Ditmarsch. Hans and Ji appreciate support from AOARD research grant AOARD-05-4017. Hans and Rineke appreciate support from the Netherlands Organization for Scientific Research (NWO). Rineke also thanks the Netherlands Institute for Advanced Study in the Humanities and Social Sciences (NIAS) for support. A shortened version of Sections 7 and 8, including Figure 2, have previously appeared in conference proceedings [vDRV05]. We thank three anonymous referees of the Journal for Logic and Computation for their valuable comments. For their helpfulness in our attempts to uncover the history of the riddle or other helpful comments we thank Henk Barendregt, Johan van Benthem, J. Boersma, N.G. de Bruin, John Conway, Dirk van Dalen, Paul Drijvers, Jan van Eijck, Peter van Emde Boas, Solomon Feferman, Martin Gardner, Mimi Gardner, Joe Halpern, Marty Isaacs, Dick de Jongh, J.H. van Lint, John McCarthy, Alice ter Meulen, Ron van der Meyden, Rohit Parikh, Mike Paterson, Albert de Rijk, Lee Sallows, Larry Tesler, Frank Thuijsman, Jaap Top, Fer-Jan de Vries, and Rob van der Waall.

[†]Computer Science, University of Otago, New Zealand, `hans@cs.otago.ac.nz`

[‡]Computer Science, University of Liverpool, United Kingdom, `jruan@csc.liv.ac.uk`

[§]Artificial Intelligence, University of Groningen, the Netherlands, `rineke@ai.rug.nl`


No. 223. A zegt tot S en P: Ik heb twee gehele getallen x, y gekozen met $1 < x < y$ en $x + y \leqslant 100$. Straks deel ik $s = x + y$ aan S alleen mee, en $p = xy$ aan P alleen. Deze mededelingen blijven geheim. Maar jullie moeten je inspannen om het paar (x, y) uit te rekenen.

Hij doet zoals aangekondigd. Nu volgt dit gesprek:

1. P zegt: Ik weet het niet.
2. S zegt: Dat wist ik al.
3. P zegt: Nu weet ik het.
4. S zegt: Nu weet ik het ook.

Bepaal het paar (x, y).

(*H. Freudenthal*).

Figure 1: The original publication

1 Introduction

The following problem, or riddle, was first stated in the Dutch-language mathematics journal *Nieuw Archief voor Wiskunde* in 1969 [Fre69] and subsequently solved in [Fre70]. A translation of the original formulation is:

> A says to S and P: I have chosen two integers x, y such that $1 < x < y$ and $x + y \leq 100$. In a moment, I will inform S only of $s = x + y$, and P only of $p = xy$. These announcements remain private. You are required to determine the pair (x, y).
>
> He acts as said. The following conversation now takes place:
>
> i. P says: "I do not know it."
> ii. S says: "I knew you didn't."
> iii. P says: "I now know it."
> iv. S says: "I now also know it."
>
> Determine the pair (x, y).

The announcements by the agents appear to be about ignorance and knowledge only. But actually the agents learn numerical facts from each other's announcements. For example, the numbers cannot be 2 and 3, or any other pair of prime numbers, nor for example 2 and 4, because in all those cases Product would immediately have deduced the pair from their product. As a somewhat more complicated example, the numbers cannot be 14 and 16: if they were, their sum would be 30. This is also the sum of the prime numbers 7 and 23. But then, as in the previous example, Product (P) would have known the numbers, and therefore Sum (S)—if the sum had been 30—would have considered it possible that Product knew the numbers. But Sum said that he *knew* that Product didn't know the numbers. So the numbers cannot be 14 and 16. Sum and Product learn enough, by eliminations of which we gave some examples, to be able to determine the pair of numbers: the unique solution of the problem is the pair $(4, 13)$.

The knowledge that agents have about mental states of other agents and, in particular, about the effect of communications, is vital for solving important problems in multi-agent systems, both for cooperative and for competitive groups. Dynamic epistemic logic was developed to study the changes brought about by communication in such higher-order knowledge of other agent's and of group knowledge [BMS98, Ger99]. The Sum-and-Product puzzle presents a complex illustrative case of the strength of specifications in dynamic epistemic logic and of the possibilities of automated model checking, and both can also be used in real multi-agent system applications. As far as we know, we are the first to use an automated model checker to tackle the Sum-and-Product problem.

Overview of content Section 2 gives an overview of the dissemination of the riddle through the academic community, and suggests some precursors. In Section 3 we introduce public announcement logic. In Section 4 we model the Sum-and-Product problem in public announcement logic. Section 5 provides the general setting of unsuccessful updates of which some announcements in the riddle provide examples. Section 6 models the Sum-and-Product problem, alternatively, as an interpreted system. In Section 7 we introduce the epistemic model checker DEMO. In Section 8 we implement the Sum-and-Product specification of Section 4 in DEMO, and we verify its epistemic features. In particular, we compare the model checking results in DEMO to our experiences with other epistemic model checkers. In Section 9, we investigate the complexity of model checking for the Sum-and-Product problem in DEMO both theoretically and experimentally.

2 History

John McCarthy wrote the earliest full-length treatment of the Sum-and-Product riddle in the years 1978–1981 [McC90]. McCarthy formulates the problem as follows:

> Two numbers m and n are chosen such that $2 \leq m \leq n \leq 99$. Mr. S is told their sum and Mr. P is told their product. The following dialogue ensues:
>
> i. Mr. P: I don't know the numbers.
>
> ii. Mr. S: I knew you didn't know. I don't know either.
>
> iii. Mr. P: Now I know the numbers.
>
> iv. Mr. S: Now I know them too.

In view of the above dialogue, what are the numbers?

In [McC90] the problem is elegantly modelled in modal logic in such a way that it can be processed in the (first-order) logic theorem prover FOL. This includes an—almost off-hand—introduction of what corresponds to the essential concept of 'common knowledge': what Sum and Product commonly know is crucial to a clear understanding of the problem. Common knowledge had received a very interesting treatment already in Lewis' 1969 book *Convention* [Lew69] (and also appears in other philosophical literature from that period, e.g. in Schiffer's work [Sch72]), but McCarthy seems to have re-invented it in his 1981 article, thereby inspiring research on common knowledge in Artificial Intelligence.

There are several differences between the McCarthy and the Freudenthal version. In the McCarthy version the upper bound for both numbers is 99 (a), in the Freudenthal version the upper bound for their *sum* is 100. Also, unlike Freudenthal, McCarthy allows the two numbers to be the same (b). Many more number pairs are therefore allowed in the McCarthy version, e.g., $(99, 99)$ with sum 198. Also, in the second announcement, Sum gives some additional information that does not appear in the Freudenthal-version of the dialogue, namely "I don't know either" (c). It can be shown that none of these three changes $((a), (b), (c))$, or their interaction, affects the solution. In particular, the addition "I don't know either" is superfluous. After Product's first announcement it is already common knowledge among Sum and Product (and to the reader) that 'Sum doesn't know either', because Sum only knows the two numbers from the start when they are 2 and 3 (i.e., in the Freudenthal version; in the McCarthy version: when they are 2 and 2, or 2 and 3; see also [Pan91], below), and these numbers being prime, Product would have derived them immediately, as already mentioned. One can also show (rather more technically) that the truth of 'Sum doesn't know either' persists after the "I knew you didn't know" part of Sum's announcement.

Many different versions of the puzzle elicited much discussion from the late seventies onwards. The variations are caused by different announcements, different ranges for the numbers, and different choices for what is considered to be common knowledge at the starting-point. For yet another example, for a certain larger range of possible numbers than $2 \dots 99$ one finds one or more solutions different from $(4, 13)$ where those solutions may after all be in the $2 \dots 99$ range. Discussions of several variants of the problem can be found in the literature on recreational mathematics, see e.g. [Gar79, Sal95, Isa95], and on a website `www.mathematik.uni-bielefeld.de/~sillke/PUZZLES/logic_sum_product` that contains many other references. For a superbly elegant mathematical analysis we recommend Isaacs [Isa95].

More geared towards an epistemic logical audience are [Pla89, Pan91, vdM94, vdHV02]. Plaza [Pla89] and Panti [Pan91] were students of Rohit Parikh and have both made interesting contributions to epistemic logic. Parikh discussed issues related to the conversational setting of 'Sum and Product' in [Par92]. He is the motivating source for much work in early and current research in dynamic epistemic logic. In [Pla89] the Sum-and-Product problem is modelled in a dynamic epistemic logic that is the precursor of the public announcement logic presented here, namely without an operator for common knowledge. We owe to Plaza a description of the initial model for solving the Sum-and-Product puzzle and the formalization of the announcements in epistemic logic, to be presented in Section 4. Plaza does not assume that an upper bound for the numbers, or their sum, is (commonly) known to Sum and Product; instead, he asks the *problem solver* for solutions with numbers smaller than 100, to which the answer is: the four pairs $(4, 13)$, $(4, 61)$, $(16, 73)$, $(64, 73)$.[1] These are the first four of the much longer list provided by Isaacs in the above-mentioned [Isa95]. For example, $(64, 73)$ is clearly not a solution if Sum and Product had been aware of the upper bound, as it is the unique number pair with product $(64 \cdot 73)$ and factors below 100—73 is a prime number—so that Product would then immediately have known the numbers. In [Pan91] the common knowledge involved in the Sum-and-Product puzzle is investigated in detail, with an emphasis on the arithmetic involved. For example, for the formulation of the problem where the range of numbers (up to 100) is not considered to be common knowledge at the start, Panti proves that if the sum of the numbers is greater or equal than 7, then this (and its logical consequences) is the *only* fact that is common knowledge among Sum and Product. Finally, Van der Meyden [vdM94] suggests a solution in temporal epistemic logic.

2.1 Looking for the origin of Sum and Product

In both of the two first full-length publications on the Sum-and -Product riddle [McC90, Gar79], the authors explicitly wondered about but could not give its exact origins. John McCarthy explains in a footnote in his paper [McC90]:

> I have not been able to trace Mr. S and Mr. P back beyond its alleged appearance on a bulletin board at Xerox PARC.

Martin Gardner, in his 1979 "Mathematical Games" column [Gar79], writes:

> This beautiful problem, which I call "impossible" because it seems to lack sufficient information for a solution, began making the rounds of mathematics meetings a year or so ago. I do not know its origin.

After the appearance of [Gar79], the fact that the puzzle had been published already in 1969 by Dutch topologist and specialist on mathematics education Hans Freudenthal, was brought to Gardner's attention by Dutch algebraist Robert van der Waall. Van der Waall was one of those who had sent in a correct solution after the puzzle's first appearance in 1969 [Fre69].

We have tried to fill in two missing pieces in the history of the Sum-and-Product riddle:

[1] Plaza also mentions the McCarthy version with publicly known upper bound of 100 [Pla89, p.14], but incorrectly suggests that these four pairs then still constitute the answer: only $(4, 13)$ remains in that case.

i. If [Fre69] is indeed the first published appearance of the problem, then how did the problem migrate from the Dutch mathematics community of the late 1960s and early 1970s to "a bulletin board at Xerox Parc" and "the rounds of mathematics meetings" in the United States in the late 1970s?

ii. Did Freudenthal invent the problem? And if so, has he possibly been inspired by (less complex) precursors?

Despite several requests on international e-mail lists, we have not been able to answer the first question. As to the second question, we did not find positive confirmation about the puzzle in Freudenthal's educational and autobiographic writings, nor from our contacts with the Freudenthal Institute. A subscriber to *Nieuw Archief* wrote to us that he remembered having seen the Sum-and-Product riddle in the puzzle column "Breinbrouwsels" (brain brews) in the now defunct Dutch-language weekly *De Katholieke Illustratie* ('Illustrated Catholic Magazine') in the 1950s. This memory proved to be incorrect. We consulted, directly or indirectly (by their answers), all of the 626 "Breinbrouwsels" that G. van Tilburg published from 1954 until 1965, and did not find the Sum-and-Product riddle. Instead, we found several precursors to the Sum-and-Product riddle on which we intend to report separately in *Nieuw Archief* and that are similar to the riddles of British origin to be discussed next. Interestingly, we also found a slightly older (1956) version [vT56] of another epistemic riddle, the 'Muddy Children' problem, than the Gamow & Stern (1958) reference [GS58] commonly given in the literature.

It turned out, however, that the answer to the second question is highly likely to be: "yes, he did invent the problem." Professor N.G. de Bruin observed (personal communication) that if no other source was mentioned in the problem section of *Nieuw Archief* at the time, the contributor of a problem was always its originator. Possibly, Freudenthal was inspired by Van Tilburg's "Breinbrouwsels" or by some even earlier riddles of British origin, to which we turn our attention now.

2.2 Precursors of Sum and Product

David Singmaster's bibliographies on recreational mathematics (see `www.g4g4.com/MyCD5/SOURCES/singmaterial.htm`) point to some candidate epistemic puzzles that appeared even earlier than Van Tilburg's. The earliest precursor of the Sum-and-Product riddle that we have been able to trace is the following one, probably invented by Williams and Savage and first published in book-form in 1940 in *The Penguin Problems Book* [WS40, p.53]:

> **The church afloat**
> "I'm taking three females on the river to-morrow," said the vicar to his curate; "would you care to join our party?"
>
> "What are their ages?" asked the curate, cautiously. "Far be it from me to disclose a lady's age!" said the vicar, "but I can tell you this—

> the product of their ages is 840, and the sum is twice the number of years in your own age. You, a mathematician, should be able to find their ages for yourself."
>
> "Sounds like casuistry, Vicar," said the curate; "but, as a matter of fact, I can't find their ages from your data. By the way, is the eldest older than you?"
>
> "No, younger." "Ah, now I know their ages!" said the curate. "Thanks, I will come with pleasure."
>
> What was the curate's age? How old were the ladies? And what can be deduced about the vicar's age?

Here follows Williams' and Savage's answer [WS40, p.135]:

> Sum of ages must be even.
>
> Uncertainty, resolved by the vicar's final statement, must be due to the fact of there being more than one such sum which was twice the curate's age.
>
> Of the possible sets of 3 factors of 840, there are only two cases of the same *even* sum occurring more than once. The sums in these cases are 46 and 30. Now the curate's age could not be 15; therefore he was 23.
>
> The sets of female ages giving a sum of 46 are 35, 8, 3 and 30, 14, 2. Since the vicar's answer excluded one of these, that one must be the former. Therefore the ladies' ages were 30, 14, 2, and the vicar's age must lie between 30 and 35.

Note that some world knowledge is used implicitly here, namely the fact that mathematicians (and curates) are always older than 15 years, and the fact that the curate, being a mathematician, reasons correctly.

Another problem, that was published in 1944 in *The Second Penguin Problems Book* [WS44, p.27], also hinges on the fact that only for some number combinations there is more than one way to make the same sum. In a way, the next problem is less attractive than the previous one, because the uncertainty is not completely dissolved at the end: readers are asked to derive the sum of the ages only.

> **Domiciliary**
> "I have told you my age," said Mr. Ptolemy to the inspector who had just knocked on his door. "Besides myself, there are three persons living in this house; the product of their ages is one thousand two hundred ninety-six, and the sum of their ages is the number of the house."
>
> "But it is impossible for me to be *sure* of their ages without further information," said the inspector. "Is any one of them the same age as yourself?"

> "No," said Mr. Ptolemy.
>
> "Thanks; now I know their ages," said the inspector.
>
> What was the number of Mr. Ptolemy's house?

This time, the explanation is as follows [WS44, p.116]; again, the authors implicitly use some world knowledge:

> There are many ways of splitting 1296 into three factors, but only *possible* ones need be considered. Two of these sets of factors have the same sum, namely 1, 18, 72 and 2, 8, 81, adding up to 91. The other sums are all different. As the inspector could not be sure of the ages from the fact that they added up to the number of the house (which he, of course, knew), this number must have been 91. [Mr. Ptolemy's age - also known to the inspector - must have been 72 or 81 (unless it was 18 or 8 - both unlikely), but we have no means of deciding this point.]

The above two puzzles are roughly of the same kind as Van Tilburg's, but still different. In fact, Van Tilburg may have been inspired to create his puzzles after reading the British gentlemen. Essentially the same problem as "Domiciliary", but in a somewhat different guise, was printed in Greenblatt's *Mathematical Entertainments* [Gre68], first published in the United States in 1965. Greenblatt starts with some historical speculation:

> One of the few amusing things to come out of World War II was a new type of brain twister - the "census-taker" problem. (The time and place of origin of a problem are difficult to specify. To the best of the author's knowledge, this problem was born on the M.I.T. campus in one of the war projects.)

As we now know, the type of problem probably stems from at least somewhat before the start of World War II, and from Great Britain instead of the United States. After all, *The Penguin Problems Book*, although published during the war in 1940, was mostly based on earlier puzzles from Williams' and Savage's column "Perplexities" that used to appear in *The Strand Magazine*.

After this detailed overview of the dissemination of the Sum-and-Product riddle, which we hope may prevent some of this information from gradually disappearing into the fog of war on academic battlegrounds, we continue with the more technical core of this paper, that consists of an introduction into public announcement logic, modelling the riddle in this logic, and verifying its properties in a model checker.

3 Public Announcement Logic

Public announcement logic is an extension of standard multi-agent epistemic logic with dynamic modal operators to model the effects of announcements. It was originally proposed by Plaza [Pla89]. Plaza used a different notation, without dynamic modal operators, and did not incorporate common knowledge. Later milestones, with common knowledge and also involving further generalizations, are [GG97] (who did unfortunately not know about [Pla89] at that time), and [BMS98]. Intuitive explanations of the non-dynamic part of the semantics can be found in [FHMV95, vdHV02, vDvdHK06]. We give a concise overview of the language, the structures on which the language is interpreted, and the semantics.

Given are a finite set of agents A and a finite or countably infinite set of atoms Q. The language of public announcement logic is inductively defined as

$$\varphi ::= q \mid \neg\varphi \mid (\varphi \wedge \varphi) \mid K_a\varphi \mid C_B\varphi \mid [\varphi]\psi$$

where $q \in Q$, $a \in A$, and $B \subseteq A$ are arbitrary. For $K_a\varphi$, read 'agent a knows formula φ'. For $C_B\varphi$, read 'group of agents B commonly know formula φ'. For $[\varphi]\psi$, read 'after public announcement of φ, formula ψ (is true)'.

Next, we introduce the (Kripke) structures. An *epistemic model* $M = \langle W, \sim, V\rangle$ consists of a *domain* W of (factual) *states* (or 'worlds'), *accessibility* $\sim : A \to \mathcal{P}(W \times W)$, where each $\sim(a)$ is an equivalence relation, and a *valuation* $V : Q \to \mathcal{P}(W)$. For $w \in W$, (M, w) is an *epistemic state* (also known as a pointed Kripke model, or possible-worlds model).[2] For $\sim(a)$ we write $\sim_a$, and for $V(q)$ we write V_q. So, the accessibility function $\sim$ can be seen as a set of equivalence relations $\sim_a$, and V as a set of valuations V_q. Given two states w, w' in the domain, $w \sim_a w'$ means that w is indistinguishable from w' for agent a on the basis of its information. For example, at the beginning of the riddle, pairs $(14, 16)$ and $(7, 23)$ are indistinguishable for Sum but not for Product. Therefore, assuming a domain of number pairs, we have that $(14, 16) \sim_S (7, 23)$ but that $(14, 16) \not\sim_P (7, 23)$. The group accessibility relation $\sim_B$ is the transitive and reflexive closure of the union of all accessibility relations for the individuals in B: $\sim_B \equiv (\bigcup_{a \in B} \sim_a)^*$. This relation is used to interpret common knowledge for group B.

Finally, we give the semantics. Assume an epistemic model $M = \langle W, \sim, V\rangle$.

$$\begin{array}{lll}
M, w \models q & \text{iff} & w \in V_q \\
M, w \models \neg\varphi & \text{iff} & M, w \not\models \varphi \\
M, w \models \varphi \wedge \psi & \text{iff} & M, w \models \varphi \text{ and } M, w \models \psi \\
M, w \models K_a\varphi & \text{iff} & \text{for all } \ v \in W : w \sim_a v \text{ implies } M, v \models \varphi \\
M, w \models C_B\varphi & \text{iff} & \text{for all } \ v \in W : w \sim_B v \text{ implies } M, v \models \varphi \\
M, w \models [\varphi]\psi & \text{iff} & M, w \models \varphi \text{ implies } M|\varphi, w \models \psi
\end{array}$$

Here, epistemic model $M|\varphi = \langle W', \sim', V'\rangle$ is defined as

$$\begin{array}{lcl}
W' & = & \{w' \in W \mid M, w' \models \varphi\} \\
\sim'_a & = & \sim_a \cap (W' \times W') \\
V'_q & = & V_q \cap W'
\end{array}$$

[2]'Epistemic state' in our sense is not to be confused with the set of worlds/states indistinguishable for a given agent from a given state, i.e., the set $\{v \mid w \sim_a v\}$, which is also commonly called an epistemic state. We call that an *epistemic class*.

The dynamic modal operator $[\varphi]$ is interpreted as an epistemic state transformer. Announcements are assumed to be truthful, and this is commonly known by all agents. Therefore, the model $M|\varphi$ is the model M restricted to all the states where φ is true, including access between states. The dual of $[\varphi]$ is $\langle\varphi\rangle$: $M, w \models \langle\varphi\rangle\psi$ iff $M, w \models \varphi$ and $M|\varphi, w \models \psi$.

Formula φ is valid on model M, notation $M \models \varphi$, if and only if for all states w in the domain of M: $M, w \models \varphi$. Formula φ is valid, notation $\models \varphi$, if and only if for all models M: $M \models \varphi$. Logical consequence $\Psi \models \varphi$ is defined as "for all (M, w), if $M, w \models \psi$ for all $\psi \in \Psi$, then $M, w \models \varphi$." For $\{\psi\} \models \varphi$, write $\psi \models \varphi$. A proof system for this logic is presented, and shown to be complete, in [BMS98], with precursors—namely for public announcement logic *without* common knowledge—in [Pla89, GG97]. For a concise completeness proof, see [vDvdHK06]. Some relevant principles of public announcement logic are

i. $[\varphi]\psi \leftrightarrow (\varphi \rightarrow [\varphi]\psi)$

ii. $[\varphi][\psi]\chi \leftrightarrow [\varphi \wedge [\varphi]\psi]\chi$

iii. $[\varphi]K_a\psi \leftrightarrow (\varphi \rightarrow K_a[\varphi]\psi)$

iv. $[C_A\varphi]C_A\varphi$

Item *i* expresses that the interpretation of the dynamic operator $[\varphi]$ is a *partial* function. Item *ii* expresses that a sequence of two announcements φ and ψ can be replaced by the single announcement 'φ, and after φ, ψ'. Item *iii* expresses the preconditions and postconditions of announcements with respect to individual knowledge (for common knowledge, this relation is more complex). Item *iv* expresses that *public* knowledge (i.e., common knowledge for the entire group of agents) remains true after announcement. Not all formulas remain true after their announcement, in other words, $[\varphi]\varphi$ is *not* a principle of this logic. This matter will be addressed in Section 5. One of the announcements in the Sum-and-Product problem provides a concrete counterexample.

4 Sum and Product in Public Announcement Logic

We give a specification of the Sum-and-Product problem in public announcement logic. Modulo inessential differences, explicitly mentioned below, this specification was first suggested by Plaza in [Pla89], and in this section we merely elaborate on his results, that are unfortunately not as well-known as they deserve to be.

First we need to determine the set of atomic propositions and the set of agents. In the formulation of the problem, x, y are two integers such that $1 < x < y$ and $x + y \leq 100$. Define $I \equiv \{(x, y) \in \mathbb{N}^2 \mid 1 < x < y \text{ and } x + y \leq 100\}$. Consider the variable x. If its value is 3, we can represent this information as the (truth of) the atomic proposition '$x = 3$'. Slightly more formally we can think of '$x = 3$' as a propositional letter x_3. Thus we create a (finite) set of atoms $\{x_i \mid (i,j) \in I\} \cup \{ y_j \mid (i,j) \in I\}$.

Concerning the agents, the role of the announcer A is to guarantee that the background knowledge for solving the problem is commonly known among Sum and Product. The announcer need not be introduced as an agent in the logical modelling of the system. That leaves $\{S, P\}$ as the set of agents. Agents S and P will also be referred to as Sum and Product, respectively.

The proposition 'Sum knows that the numbers are 4 and 13' is represented as $K_S(x_4 \wedge y_{13})$. The proposition 'Sum knows the (pair of) numbers' is described as $K_S(x,y) \equiv \bigvee_{(i,j)\in I} K_S(x_i \wedge y_j)$. Similarly, 'Product knows the numbers' is represented by $K_P(x,y) \equiv \bigvee_{(i,j)\in I} K_P(x_i \wedge y_j)$. Furthermore, note that the 'knew' in announcement ii, by Sum, refers to the truth of $K_S \neg K_P(x,y)$ in the *initial* epistemic state, not in the epistemic state *resulting* from announcement i, by Product.

Because of the property that all known propositions are true ('$K_a\varphi \rightarrow \varphi$' is valid), announcement i is entailed by announcement ii. Because of that, and as Product's announcement iii is a response to Sum's ii, and Sum's iv to Product's iii, the initial announcement i by Product is superfluous in the subsequent analysis. [3] This is sufficient to formalize the announcements made towards a solution of the problem:

i. P says: "I do not know it": $\neg K_P(x,y)$

ii. S says: "I knew you didn't": $K_S \neg K_P(x,y)$

iii. P says: "I now know it": $K_P(x,y)$

iv. S says: "I now also know it": $K_S(x,y)$

We can interpret these statements on an epistemic model $\mathcal{SP}_{(x,y)} \equiv \langle I, \sim, V\rangle$ consisting of a domain of all pairs $(x,y) \in I$ (as above), with accessibility relations $\sim_S$ and $\sim_P$ such that for Sum: $(x,y) \sim_S (x',y')$ iff $x+y = x'+y'$, and for Product: $(x,y) \sim_P (x',y')$ iff $xy = x'y'$; and with valuation V such that $V_{x_i} = \{(x,y) \in I \mid x = i\}$ and $V_{y_j} = \{(x,y) \in I \mid y = j\}$.[4]

[3] Additional to this justification that announcement i is superfluous, we cannot formalize that announcement *ii follows* announcement i in our logical language, as we cannot refer to the past. In dynamic epistemic logic *with assignment* one can indirectly model such past tense epistemic statements [Koo06]; see also [Yap06].

[4] Plaza does not assume that an upper bound for the numbers is given to the agents (as mentioned in the historical Section 2). The model in [Pla89] corresponding to $\mathcal{SP}_{(x,y)}$ is therefore infinite. He also specifies a finite model, for the McCarthy version of the riddle. Now given an infinite model, the corresponding announcements would be *infinitary* formulas, i.e., not well-formed formulas in our definition of the logical language. Indeed, Plaza does not model the announcements based on disjunctions of individual epistemic knowledge statements, but with 'know-value' individual knowledge operators binding non-rigid designators such that, e.g., $\mathsf{Kv}_{Product}\mathsf{numbers}$ stands for 'Product knows the *value* of the non-rigid designator / number variable $\mathsf{numbers}$' [Pla89, p.13–14]. The corresponding logic has a semantics, but is not known to be complete [Pla89, p.13].

We can describe the solution of the problem as the truth of the statement

$$\mathcal{SP}_{(x,y)}, (4, 13) \models \langle K_S \neg K_P(x, y)\rangle\langle K_P(x, y)\rangle\langle K_S(x, y)\rangle\top$$

This expresses that, if $(4, 13)$ is the initial state, then it is possible to publicly announce *ii*, *iii*, and *iv*, in that order. This statement does not express that $(4, 13)$ is the *only* solution. We can express more properly that $(4, 13)$ is the only solution (and from here on our observations go beyond [Pla89] again) as the model validity

$$\mathcal{SP}_{(x,y)} \models [K_S \neg K_P(x, y)][K_P(x, y)][K_S(x, y)](x_4 \wedge y_{13})$$

This expresses that in all *other points* of the model than $(4, 13)$ the sequence of three announcements cannot be truthfully made. Technically this relates to the well-known modal property that formulas of form $\Box\varphi$ are true for *all* φ in all worlds where there are no accessible worlds. If all announcements are true, three consecutive epistemic state transformations result in a final epistemic state (M, w) (namely $(\mathcal{SP}_{(x,y)}|ii|iii|iv, w)$) wherein $x_4 \wedge y_{13}$ should hold. Clearly, this is only the case when $w = (4, 13)$. In all other states of the domain I of model $\mathcal{SP}_{(x,y)}$, at least one announcement cannot be truthfully made; but that means that *any* postcondition of the dynamic 'necessity-type' modal operator corresponding to that announcement, even 'false', is true in that state.

For example, we observed that in state $(7, 23)$ Product would know the numbers (as they are both prime). Therefore, $\neg K_P(x, y)$ is false in $(\mathcal{SP}_{(x,y)}, (7, 23))$, and therefore $K_S \neg K_P(x, y)$ (announcement *ii*) is also false in $(\mathcal{SP}_{(x,y)}, (7, 23))$. The semantics gives us $\mathcal{SP}_{(x,y)}, (7, 23) \models [K_S \neg K_P(x, y)]([K_P(x, y)][K_S(x, y)](x_4 \wedge y_{13}))$, and even $\mathcal{SP}_{(x,y)}, (7, 23) \models [K_S \neg K_P(x, y)]\bot$.

An issue of general logical interest is that some announcements in Sum and Product are examples of formulas that become false when announced. This will now first be addressed, in the next Section 5. In Section 6 after that, we will model Sum and Product as an interpreted system. This will be followed by several sections on model checking.

5 Unsuccessful updates

Not all formulas remain true after their announcement, in other words, $[\varphi]\varphi$ is *not* a principle of public announcement logic. A poignant example is the announcement of $q \wedge \neg K_a q$, for 'q is true and you don't know that'.[5] After the announcement, you know that the fact in question is true—$K_a q$—and therefore the formula of the announcement has become false: $K_a q$ entails $\neg q \vee K_a q$, which is equivalent to $\neg(q \wedge \neg K_a q)$, the negation of the announcement. In a somewhat

[5]This commonly occurs in conversational settings such as "You don't know that the Highlanders just beat the Lions!". Unlike in epistemic logic, the standard conversational setting presumes, by the Gricean assumption of cooperation, that the factual information of which you are said to be ignorant is actually the case, i.e., the above normally means "The Highlanders just beat the Lions and you don't know that the Highlanders just beat the Lions."

different setting that the formula $q \wedge \neg K_a q$ cannot be consistently known, this phenomenon has been known in philosophical circles for a long time, namely as the Moore-paradox [Moo42, Hin62]. In the underlying dynamic setting it has been described as an unsuccessful update in [Ger99, Ger06]. General terminology is proposed in [vDK06]. Let φ be a formula in the language of public announcement logic:

- *Successful formula*
  φ is successful iff $[\varphi]\varphi$ is valid.
- *Unsuccessful formula*
  φ is unsuccessful iff it is not successful.
- *Successful update*
  φ is successful in epistemic state (M, w) iff $M, w \models \langle\varphi\rangle\varphi$
- *Unsuccessful update*
  φ is unsuccessful in (M, w) iff $M, w \models \langle\varphi\rangle\neg\varphi$.

Note that an unsuccessful formula may be a successful update in one epistemic state and an unsuccessful update in another epistemic state. It can be shown that $[\varphi]\varphi$ is valid iff $[\varphi]C_B\varphi$ is valid iff $\varphi \rightarrow [\varphi]C_B\varphi$ is valid. (See [vDK06], the second equivalence follows directly from the principle $[\varphi]\psi \leftrightarrow (\varphi \rightarrow [\varphi]\psi)$, listed as item i on page 10 in Section 3.) Therefore, the successful formulas capture the notion 'formulas that remain true after their announcement'.

Clearly, also in the course of solving the Sum-and-Product problem the agents appear to learn things that they did not know before. So some reversal of ignorance into knowledge seems to take place: i.e., an unsuccessful update with something of the form $\neg K\varphi$, or similar. We investigate which of the announcements made towards the solution of the problem are unsuccessful updates. In this section we refer to those four successive announcements as (how they have been enumerated before, e.g. on page 11, namely as) (*i*) $\neg K_P(x, y)$, (*ii*) $K_S\neg K_P(x, y)$, (*iii*) $K_P(x, y)$, and (*iv*) $K_S(x, y)$.

The case i Formula i is successful. It equals $\neg K_P(x, y)$, where $K_P(x, y)$ is defined as $\bigvee_{(i,j)\in I} K_P(x_i \wedge y_j)$. Therefore, it has form $\neg K_a\varphi \wedge \neg K_a\psi \wedge \ldots$, with $\varphi, \psi, \ldots$ booleans. We show that this formula is successful for two conjuncts, i.e., formula $[\neg K_a\varphi \wedge \neg K_a\psi](\neg K_a\varphi \wedge \neg K_a\psi)$ is valid for booleans φ and ψ; the case for the longer finite conjunction follows by induction on the number of conjuncts.

Let M, w be arbitrary. Assume $M, w \models \neg K_a\varphi \wedge \neg K_a\psi$. We have to prove that $M|(\neg K_a\varphi \wedge \neg K_a\psi), w \models \neg K_a\varphi \wedge \neg K_a\psi$. From $M, w \models \neg K_a\varphi \wedge \neg K_a\psi$ follows that there are v and v' in the domain $\mathcal{D}(M)$ of M such that $v \sim_a w$ and $M, v \models \neg\varphi$, and $v' \sim_a w$ and $M, v' \models \neg\psi$, respectively. As $\sim_a$ is an equivalence relation, we also have that $v \sim_a v$ and $v \sim_a v'$, we have as well $M, v \models \neg K_a\varphi \wedge \neg K_a\psi$; similarly, $M, v' \models \neg K_a\varphi \wedge \neg K_a\psi$. In other words, both v and v' are in the domain of $M|(\neg K_a\varphi \wedge \neg K_a\psi)$. As the value of boolean propositions only depends on the current factual state, from $M, v \models \neg\varphi$ and $v \in \mathcal{D}(M|(\neg K_a\varphi \wedge \neg K_a\psi))$ follows $M|(\neg K_a\varphi \wedge \neg K_a\psi), v \models \neg\varphi$; and from the last follows $M|(\neg K_a\varphi \wedge \neg K_a\psi), w \models \neg K_a\varphi$. Similarly, $M|(\neg K_a\varphi \wedge \neg K_a\psi), v' \models \neg\psi$; from which follows $M|(\neg K_a\varphi \wedge \neg K_a\psi), w \models \neg K_a\psi$. Therefore $M|(\neg K_a\varphi \wedge \neg K_a\psi), w \models \neg K_a\varphi \wedge \neg K_a\psi$, as required.

The case ii Announcement *ii* becomes false when it is announced in the initial epistemic state for the problem. This can be easily observed: after *ii*, Product knows the numbers (formula *iii*), so it can no longer be true that Sum knows that Product does not know the numbers: in other words, formula *ii* is now false. Ergo, *ii* is an unsuccessful update in the initial epistemic state.

The cases iii and iv The last two announcements *iii* and *iv* are successful formulas: this is because they are *preserved* formulas: they are truth preserving under submodel restrictions, an inductively defined fragment with—among other clauses—inductive clauses that atomic propositions are always preserved, and that if φ and ψ are preserved, then also $\varphi \wedge \psi$, $\varphi \vee \psi$, and $K_a\varphi$ [vB02]. The announcements *iii* and *iv* are disjunctions of formulas of the form $K_a(x_i \wedge y_j)$, and are therefore preserved. All preserved formulas are successful [vDK06]. And all successful formulas induce successful updates in all epistemic states.

No inductive definition of the successful formulas is known—in particular, if φ and ψ are both successful, $[\varphi]\psi$ may be unsuccessful. Having said that, it is remarkable that the sequence of the three announcements *ii* ; *iii* ; *iv* is an unsuccessful update. This sequence of three announcements is equivalent to the single formula: $ii \wedge [ii]iii \wedge [ii \wedge [ii]iii]iv$ (using the validity $[\varphi][\psi]\chi \leftrightarrow [\varphi \wedge [\varphi]\psi]\chi$ stated on page 10) that is unsuccessful in the initial epistemic state. This formula becomes false after its announcement: after that, just like after *ii*, Sum knows that Product knows the numbers, so it is now false that Sum knows that Product does not know the numbers: *ii* has become false, and therefore the entire conjunction corresponding to the sequence *ii* ; *iii* ; *iv*. The first announcement *i* can also be added to the conjunction (although somewhat improperly joining two different timelines), so that $i \wedge ii \wedge [ii]iii \wedge [ii \wedge [ii]iii]iv$ is also unsuccessful in the initial epistemic state.

This last observation captures, we think, more than anything else our intuition that the Sum-and-Product problem is puzzling.

6 Sum and Product as an interpreted system

Interpreted systems were introduced in theoretical computer science as an abstract architecture for distributed systems [FHMV95]. In an interpreted system agents at least know their local state. We can model the epistemic states in the Sum-and-Product riddle as (static) interpreted systems when we consider the sum of the two numbers as Sum's local state, and the product of the two numbers as Product's local state. A global state for the problem is a tuple of local states, in our case a pair: one for Sum and one for Product. The set of global states is a subset of the full cartesian product of local state values: the dependencies between local states enable Sum and Product to communicate their local state to each other without explicitly referring to it. A standard way to represent a static interpreted system as an epistemic model ($S5$ Kripke model) is given in [Lom99]. Each local state value for agent or processor then corresponds to an atomic proposition.

Therefore, when modelling the Sum-and-Product puzzle as an interpreted system, we introduce a different language in that a new set of atomic propositions is differently interpreted than in the model of Section 4. Specifically, atomic propositions represent the sum and product of the different numbers, instead of representing these numbers themselves. For example, the atomic proposition s_7 represents that the sum of the two numbers is 7. We allow a slight abuse of the language: if $i + j = k$ then we also write s_{i+j} for s_k. Similarly, we write p_{ij} for p_l when $ij = l$. Thus we create a set of atoms $\{s_{x+y} \mid (x, y) \in I\} \cup \{ p_{xy} \mid (x, y) \in I\}$.

The obvious way to interpret such atoms is on an epistemic model $\mathcal{SP}_{(s,p)} \equiv \langle W', \sim', V'\rangle$ with a domain W' consisting of all pairs (s, p) such that $s = x + y$ and $p = xy$ (as in the formulation of the problem in Section 4) for all $(x, y) \in I$, i.e., with $1 < x < y$ and $x + y \leq 100$; with *accessibility relations* $\sim'_S$ and $\sim'_P$ such that for Sum: $(s, p) \sim'_S (s', p')$ iff $s = s'$, and for Product: $(s, p) \sim'_P (s', p')$ iff $p = p'$; and with valuation such that $V'_{s_{x+y}} = \{(s, p) \in W' \mid s = x + y\}$ and $V'_{p_{xy}} = \{(s, p) \in W' \mid p = xy\}$.

'Sum knows the (pair of) numbers' can be represented by 'Sum knows the global state of the system', i.e., as $K_S(s, p) \equiv \bigvee_{(x,y)\in I} K_S(s_{x+y} \wedge p_{xy})$, and, similarly, 'Product knows the numbers' by $K_P(s, p) \equiv \bigvee_{(x,y)\in I} K_P(s_{x+y} \wedge p_{xy})$. The formalization of the announcements made towards a solution of the problem is then:

$$\mathcal{SP}_{(s,p)} \models [K_S\neg K_P(s, p)][K_P(s, p)][K_S(s, p)](s_{4+13} \wedge p_{4\cdot 13})$$

This is of course fairly similar to the formalization

$$\mathcal{SP}_{(x,y)} \models [K_S\neg K_P(x, y)][K_P(x, y)][K_S(x, y)](x_4 \wedge y_{13})$$

on page 12— the precise relation requires a detour on characteristic formulas for interpreted systems. This will reveal the advantage of the interpreted system representation.

6.1 Characteristic formulas

In interpreted systems agents at least know their local state. That agent S knows its local state, means that S knows the sum of the two numbers, whatever they are: $\mathcal{SP}_{(s,p)} \models s_{x+y} \rightarrow K_S s_{x+y}$. From this it follows that in the models for our problem a requirement $K_S(s_{x+y} \wedge p_{xy})$, that is equivalent to $K_S s_{x+y} \wedge K_S p_{xy}$, is equivalent to $K_S p_{xy}$. Similarly, $p_{xy} \rightarrow K_P p_{xy}$, and therefore, in the models, $K_P(s_{x+y} \wedge p_{xy})$ is equivalent to $K_P s_{x+y}$.

Such local state knowledge and its interaction with global states is incorporated in the characteristic formula of a finite interpreted system. Characteristic formulas for modal structures are found in [BM96, vB98]. We introduce them in the setting given in [vDvdHK03]. A *characteristic formula*, or description, of a pointed model (M, w) is a formula $\delta(M, w)$ such that $M, w \models \psi$ iff $\delta(M, w) \models \psi$, in other words, any ψ true in (M, w) is entailed by $\delta(M, w)$. A similar notion equates model validity with entailment by way of $M \models \psi$ iff $\delta(M) \models \psi$. These descriptions always exist for *finite* models, such as finite epistemic models, in which case formula $\delta(M, w)$ essentially requires common knowledge operators, in particular, they cannot be expressed in Plaza's dynamic epistemic language [Pla89]. We then also have that $\delta(M, w) \leftrightarrow (\delta(w) \wedge C_A \delta(M))$, where $\delta(w)$ is *the description of state* w, for example summing up its valuation, or some other formula only true in w. In other words, we can separate a commonly known part ('background knowledge') from a part specific to the actual state.

In general these characteristic formulas are rather unwieldy and do not provide very concrete information. For example, it would be unclear what they are for the epistemic model for Sum and Product in Section 4. But for the specific case of an interpreted system, such as the interpreted system model $\mathcal{SP}_{(s,p)}$ for Sum and Product, we can apply results similar to those in [vDvdHK03]. The characteristic formula $\delta(\mathcal{SP}_{(s,p)})$ is defined as

$$\begin{array}{rcl} \delta(\mathcal{SP}_{(s,p)}) & \equiv & \bigvee_{(x,y)\in I} \delta(x+y, xy) \ \wedge \\ & & \bigwedge_{(x,y)\in I} (K_S s_{x+y} \leftrightarrow \neg K_S \neg(s_{x+y} \wedge p_{xy})) \ \wedge \\ & & \bigwedge_{(x,y)\in I} (K_P p_{xy} \leftrightarrow \neg K_P \neg(s_{x+y} \wedge p_{xy})) \end{array}$$

The first conjunct of $\delta(\mathcal{SP}_{(s,p)})$ sums up the valuations of the different states in the domain: $\delta(x + y, xy)$ is defined as the conjunction of: s_{x+y}, and p_{xy}, and for all other sums k and products l the negated atoms $\neg s_k$ and $\neg p_l$. This is also known as the characteristic function of the valuation of the state $(x+y, xy)$ of the model. The second line of the formula says that S knows its local state if and only if it considers possible any global state with that local state. For example, one of its conjuncts is $K_S s_{17} \leftrightarrow \neg K_S \neg(s_{17} \wedge p_{52})$; another conjunct is $K_S s_{17} \leftrightarrow \neg K_S \neg(s_{17} \wedge p_{60})$. From this follows that $K_S s_{17}$ implies $\neg K_S \neg p_{52} \wedge \neg K_S \neg p_{60} \wedge \ldots$: if the sum of the two numbers is 17, S considers it possible that their product is 52, or 60, etc. The last line of the characteristic formula has a meaning similar to its second line: it says that P knows its local state if and only if it considers possible any global state with that local state.

6.2 Comparing the two models of Sum and Product

We now compare the modelling of Sum and Product with 'smaller and larger number pairs' in Section 4 to the interpreted system modelling in this section.

A relevant observation is that a pair of numbers (x, y) with $x < y$ corresponds to exactly one sum-product pair (s, p). In one direction this is trivial. For the other direction: assume that $(x+y, xy) = (x'+y', x'y')$, with $x < y$ and $x' < y'$.

Let without loss of generality x be the smaller of x and x', so that $x' = x + v$ and therefore $y' = y - v$. Then from $xy = x'y' = (x + v)(y - v)$ it follows that $yv - xv - v^2 = 0$, so that $v = 0$ or $v = y - x$. If $v = 0$ then we are done, since this implies $x = x'$. If $v = y - x$ then $x' = x + (y - x) = y$ and $y' = y - (y - x) = x$ which contradicts that $x' < y'$.

We can now make technically precise how the different modellings compare. Expand the language to one containing atoms for all numbers x, y *and* atoms for all sums and products s, p of those numbers. Extend the models $\mathcal{SP}_{(x,y)}$ and $\mathcal{SP}_{(s,p)}$ to $\mathcal{SP}^+_{(x,y)}$ and $\mathcal{SP}^+_{(s,p)}$, respectively, by adding valuations for all sum and product atoms in the former, and for all smaller and larger number atoms in the latter. For example, to define $\mathcal{SP}^+_{(x,y)}$ we have to add valuations for all atoms s and p such that $(x, y) \in V^+_{s_{x+y}}$ iff $s = x + y$ and $(x, y) \in V^+_{p_{xy}}$ iff $p = xy$. We now have that $\mathcal{SP}^+_{(x,y)}$ and $\mathcal{SP}^+_{(s,p)}$ are isomorphic. From this it follows that the models are also *bisimilar* [BdRV01]; as a reminder, bisimilarity is a slightly weaker notion of 'sameness of models' than isomorphism, that still guarantees that the theories describing the models are logically equivalent.

Without going into great detail, it suffices to define the isomorphism as $\mathfrak{R} : I \to W'$ such that $\mathfrak{R} : (x, y) \mapsto (x + y, xy)$, to observe that this relation is a bijection, that $(x, y) \sim_S (x', y')$ iff $\mathfrak{R}(x, y) \sim_S \mathfrak{R}(x', y')$ iff $(x + y, xy) \sim_S (x' + y', x'y')$, and similarly for Product, and that the valuation of all facts remains the same for any states (x, y) and $(x + y, xy)$. The characteristic formula for the interpreted system $\mathcal{SP}^+_{(s,p)}$ in the expanded logical language is the previous one, $\delta(\mathcal{SP}_{(s,p)})$, in conjunction with

$$\bigwedge_{(i,j)\in I} ((x_i \wedge y_j) \leftrightarrow (s_{i+j} \wedge p_{ij}))$$

This *propositional* equivalence relates a number pair to its unique corresponding sum and product pair. To conclude, using the interpreted system representation, we can describe the initial situation for the Sum-and-Product puzzle in a very precise way by a characteristic formula. Moreover, the traditional representation and the interpreted system one are in a sense interchangeable: they have the same logical theory.

Towards computational aspects Combining the different strands of the story, the epistemic state $(\mathcal{SP}_{(x,y)}, (4, 13))$ of Section 4 can alternatively be represented as an epistemic state $(\mathcal{SP}_{(s,p)}, (17, 52))$ that corresponds to an interpreted system, and there is a corresponding simple relation between different logical languages such that the formula $\langle K_S \neg K_P(x, y)\rangle\langle K_P(x, y)\rangle\langle K_S(x, y)\rangle\top$ from Section 4 corresponds to $\langle K_S \neg K_P(s, p)\rangle\langle K_P(s, p)\rangle\langle K_S(s, p)\rangle\top$ (similar to the formula on page 15, above). Schematically, starting out with some $\mathcal{SP}, w \models \varphi$, we first have an interpreted system version $\mathcal{SP}', w' \models \varphi'$ of that, which then corresponds to $\delta(\mathcal{SP}', w') \models \varphi'$ given the availability of 'readable' characteristic formulas for interpreted systems. Given the complete proof system for the logic of public announcements, this corresponds as well to $\delta(\mathcal{SP}', w') \vdash \varphi'$: one should be able to derive the required postcondition from a logical theory representing the initial model for Sum and Product.

This also suggests that proof tools may play a role in checking the solution of the Sum-and-Product riddle. In principle, one can show that $\delta(\mathcal{SP}', w') \vdash \varphi'$ in the proof system for the logic. This is not an illuminating exercise, as it requires a very lengthy derivation with explicit reference to all (more than 100) points of the model to their counterparts in the logical language, and properly handling their (non-uniform) interaction with epistemic and dynamic operators. The derivation involves eliminating the dynamic operators from φ', and the interaction between those operators and the description $\delta(\mathcal{SP}', w')$ makes the result very inelegant. Also, there are no automated proof checkers for epistemic logics with common knowledge operators.

There are, however, proof tools for model checking. In contrast to the situation in epistemic theorem proving, recent developments in dynamic epistemic model checking allow for a remodelling of the Sum-and-Product riddle in such a model checker. In dynamic epistemic model checking one can stick to the initial model $\mathcal{SP}$ and the succinct list of four announcements, and thus bypass any need to be explicit about the number theory involved: one avoids the 'real thinking' and cumbersome computation involved in solving the problem for a human problem solver, or when using a non-dedicated programming language.

'Non-dedicated' includes non-dynamic epistemic (or temporal modal) model checkers, that would again require us to eliminate the dynamic operators, thus introducing the inelegance already mentioned for theorem proving. We now turn towards implementation of our models using the epistemic model checker DEMO.

7 The Epistemic Model Checker DEMO

Recently, epistemic model checkers with dynamic facilities have been developed to verify properties of interpreted systems, knowledge-based protocols, and various other multi-agent systems. Examples are MCK [GvdM04], MCMAS [RL04], and recent work by Su [Su04]. All those model checkers use the interpreted systems architecture, and exploration of the search space is based on ordered binary decision diagrams. Their dynamics are expressed in temporal or temporal epistemic (linear and/or branching time) logics.

A different model checker, not based on a temporal epistemic architecture, is DEMO. It has been developed by Van Eijck [vE04]. DEMO is short for Dynamic Epistemic MOdelling. It allows modelling epistemic updates, graphical display of Kripke structures involved, and formula evaluation in epistemic states. This general purpose model checker has also many other facilities. DEMO is written in the functional programming language Haskell.

The model checker DEMO implements the dynamic epistemic logic of [BM04]. In this 'action model logic' the global state of a multi-agent system is represented by an epistemic model as in Section 3. But more epistemic actions are allowed than just public announcements, and each epistemic action is represented by an *action model*. Just like an epistemic model, an action model is also based on a multi-agent Kripke frame, but instead of carrying a valuation it has a precondition function that assigns a precondition to each point in the action model. A point in the action model domain stands for an atomic action.

The epistemic state change in the system is via a general operation called the *update product*: this is a way to produce a single structure (the next epistemic model) from two given structures (the current epistemic model and the current action model). We do not give details, as we restrict our attention to very simple action models, namely those corresponding to public announcements. Such action models have a singleton domain, and the precondition of that point is the announced formula. Now the way to produce the next epistemic model from the current epistemic model and the singleton action model for the announcement, is the familiar model restriction introduced in Section 3. We proceed with a relevant part of the recursive definition of formulas in DEMO, omitting the recursive clauses describing the effect of updates.

```
Form = Top | Prop Prop | Neg Form | Conj [Form] | Disj [Form]
         | K Agent Form | CK [Agent] Form
```

Formula `Top` stands for $\top$, `Prop Prop` for atomic propositional letters (the first occurrence of `Prop` means that the datatype is 'propositional atom', whereas the second occurrence of `Prop` is the placeholder for an actual proposition letter, such as `P 3`), `Neg` for negation, `Conj [Form]` stands for the conjunction of a list of formulas of type `Form`, similarly for `Disj`, `K Agent` stands for the individual knowledge operator for agent `Agent`, and `CK [Agent]` for the common knowledge operator for the group of agents listed in `[Agent]`.

The pointed and singleton action model for a public announcement is created by a function `public` with a precondition (the announced formula) as argument. The update operation is specified as

```
upd :: EpistM -> PoAM -> EpistM
```

Here, `EpistM` is an epistemic state and `PoAM` is a pointed action model, and the update generates a new epistemic state. If the input epistemic state `EpistM` corresponds to some (M, w), then in case of the truthful public announcement of φ the resulting `EpistM` has the form $(M|\varphi, w)$. We can also update with a list of pointed action models:

```
upds :: EpistM -> [PoAM] -> EpistM
```

An example is the sequence of three announcements in the Sum-and-Product problem.

8 Sum and Product in DEMO

We implement the Sum-and-Product riddle in DEMO and show how the implementation finds the unique solution $(4, 13)$. Figure 2 contains the implementation.

```
module SNP
where
import DEMO

upb = 100
pairs = [(x,y)|x<-[2..100], y<-[2..100], x<y, x+y<=upb]
numpairs = llength(pairs)
llength [] =0
llength (x:xs) = 1+ llength xs
ipairs = zip [0..numpairs-1] pairs

msnp :: EpistM
msnp = (Pmod [0..numpairs-1] val acc [0..numpairs-1])
 where
 val = [(w,[P x, Q y]) | (w,(x,y))<- ipairs]
 acc = [(a,w,v)| (w,(x1,y1))<-ipairs, (v,(x2,y2))<-ipairs, x1+y1==x2+y2 ]++
       [(b,w,v)| (w,(x1,y1))<-ipairs, (v,(x2,y2))<-ipairs, x1*y1==x2*y2 ]

fmrs1e = K a (Conj [Disj[Neg (Conj [Prop (P x),Prop (Q y)]),
                 Neg (K b (Conj [Prop (P x),Prop (Q y)]))]| (x,y)<-pairs])
amrs1e = public (fmrs1e)
fmrp2e = Conj [(Disj[Neg (Conj [Prop (P x),Prop (Q y)]),
             K b (Conj [Prop (P x),Prop (Q y)]) ] )|(x,y)<-pairs]
amrp2e = public (fmrp2e)
fmrs3e = Conj [(Disj[Neg (Conj [Prop (P x),Prop (Q y)]),
             K a (Conj [Prop (P x),Prop (Q y)]) ] )|(x,y)<-pairs]
amrs3e = public (fmrs3e)
solutione = showM (upds msnp [amrs1e, amrp2e, amrs3e])

fmrs1 = K a (Neg (Disj [ (K b (Conj [Prop (P x),Prop (Q y)]))| (x,y)<-pairs]))
amrs1 = public (fmrs1)
fmrp2 = Disj [K b (Conj [Prop (P x),Prop (Q y)])|(x,y)<-pairs]
amrp2 = public (fmrp2)
fmrs3 = Disj [K a (Conj [Prop (P x),Prop (Q y)])|(x,y)<-pairs]
amrs3 = public (fmrs3)
solution = showM (upds msnp [amrs1, amrp2, amrs3])
```

Figure 2: The DEMO program `SNP.hs`. The last part, starting from `fmrs1`, implements a less efficient variant.

8.1 Representing the epistemic models

The set $I \equiv \{(x, y) \in \mathbb{N}^2 \mid 1 < x < y \text{ and } x + y \leq 100\}$ is realized in DEMO as

```
upb = 100
pairs = [(x,y)| x<-[2..100], y<-[2..100], x<y, x+y<=upb]
```

`upb` is the maximal sum considered, in this case `upb=100`; `pairs` is a *list* of pairs: a list is a standard data structure in Haskell, unlike a set. Thus, $\{$ and $\}$ are replaced by [and], $\in$ is replaced by `<-`, and instead of I we name it `pairs`. A pair such as `(4,18)` is not a proper name for a domain element. In DEMO, natural numbers are such proper names. Therefore, we associate each element in `pairs` with a natural number and make a new list.

```
ipairs = zip [0..numpairs-1] pairs
```

Here, `numpairs` is the number of elements in `pairs`, and the function `zip` pairs the i-th element in `[0..numpairs-1]` with the i-th element in `pairs`, and makes that the i-th element of `ipairs`. For example, the first element in `ipairs` is `(0,(2,3))`. The initial model of the Sum-and-Product riddle is represented as

```
msnp :: EpistM
msnp = (Pmod [0..numpairs-1] val acc [0..numpairs-1])
 where
 val = [(w,[P x, Q y]) | (w,(x,y))<- ipairs]
 acc = [(a,w,v)|(w,(x1,y1))<-ipairs, (v,(x2,y2))<-ipairs, x1+y1==x2+y2 ]++
       [(b,w,v)|(w,(x1,y1))<-ipairs, (v,(x2,y2))<-ipairs, x1*y1==x2*y2 ]
```

Here, `msnp` is a multi-pointed epistemic model, that consists of a domain `[0..numpairs-1]`, a valuation function `val`, an accessibility relation function `acc`, and `[0..numpairs-1]` points. As the points of the model are the entire domain, we may think of this initial epistemic state as the (not-pointed) epistemic model underlying it.

The valuation function `val` maps each state in the domain to the subset of atoms that are true in that state. This is different from Section 3, where the valuation V was defined as a function mapping each atom to the set of states where it is true. The correspondence $q \in \texttt{val}(w)$ iff $w \in V(q)$ is elementary. An element `(w,[P x, Q y])` in `val` means that in state `w`, atoms `P x` and `Q y` are true. For example, given that `(0,(2,3))` is in `ipairs`, `P 2` and `Q 3` are true in state `0`, where `P 2` stands for 'the smaller number is 2' and `Q 3` stands for 'the larger number is 3'. These same facts were described in Section 4 by x_2 and y_3, respectively, as that gave the closest match with the original problem formulation. In DEMO, names of atoms *must* start with capital P, Q, R, but the correspondence between names will be obvious.

The function `acc` specifies the accessibility relations. Agent `a` represents Sum and agent `b` represents Product. For `(w,(x1,y1))` and `(v,(x2,y2))` in `ipairs`, if their sum is the same: `x1+y1==x2+y2`, then they cannot be distinguished by Sum: `(a,w,v)` in `acc`; and if their product is the same: `x1*y1==x2*y2`, then they cannot be distinguished by Product: `(b,w,v)` in `acc`. Function `++` is an operation merging two lists.

8.2 Representing the announcements

Sum and Product's announcements are modelled as singleton action models, generated by the announced formula (precondition) φ and the operation `public`. Consider $K_S \neg \bigvee_{(i,j)\in I} K_P(x_i \wedge y_j)$, expressing that Sum says: "I knew you didn't." This is equivalent to $K_S \bigwedge_{(i,j)\in I} \neg K_P(x_i \wedge y_j)$. A conjunct $\neg K_P(x_i \wedge y_j)$ in that expression, for 'Product does not know that the pair is (i,j)', is equivalent to $(x_i \wedge y_j) \rightarrow \neg K_P(x_i \wedge y_j)$.[6] The latter is computationally cheaper

[6]We use the T-validity $\neg K\varphi \leftrightarrow (\varphi \rightarrow \neg K\varphi)$, that can be shown as follows: $\neg K\varphi$ iff $(\varphi \vee \neg\varphi) \rightarrow \neg K\varphi$ iff $(\varphi \rightarrow \neg K\varphi) \wedge (\neg\varphi \rightarrow \neg K\varphi)$ iff $(\varphi \rightarrow \neg K\varphi) \wedge (K\varphi \rightarrow \varphi)$ iff (in T!) $(\varphi \rightarrow \neg K\varphi)$.

to check in the model, than the former: in all states but (i, j) of the model, the latter requires a check on two booleans only, whereas the former requires a check *in each of those states* of Product's ignorance, that relates to his equivalence class for that state, and that typically consists of several states.

This explains that the check on $K_S \neg \bigvee_{(i,j)\in I} K_P(x_i \wedge y_j)$ can be replaced by one on $K_S \bigwedge_{(i,j)\in I}((x_i \wedge y_j) \rightarrow \neg K_P(x_i \wedge y_j))$. Similary, using a model validity, the check on $\bigvee_{(i,j)\in I} K_P(x_i \wedge y_j)$ (Product knows the numbers) can also be replaced, namely by a check $\bigwedge_{(i,j)\in I}((x_i \wedge y_j) \rightarrow K_P(x_i \wedge y_j))$.[7] Using these observations, and writing an implication $\varphi \rightarrow \psi$ as $\neg\varphi \vee \psi$ (because DEMO does not support implication directly), we represent the three problem announcements *ii*, *iii*, and *iv* listed on page 2 as `fmrs1e`, `fmrp2e`, and `fmrs3e`, respectively, as listed in Figure 2. The corresponding singleton action models are obtained by applying the function `public`, e.g. `amrs1e = public (fmrs1e)`. This is also shown in the figure. The line with `solutione` abbreviates the computation of the successive model restrictions. In other words, `(upds msnp [amrs1e, amrp2e, amrs3e])` stands for epistemic model $\mathcal{SP}|ii|iii|iv$. The final part of Figure 2 encodes the less efficient version of the public announcements discussed above, e.g., `fmrs1` stands for $K_S \neg \bigvee_{(i,j)\in I} K_P(x_i \wedge y_j)$. In Section 9 we will discuss the precise computational properties of the different versions.

8.3 DEMO's interaction with the implemented model

We continue by showing a relevant part of DEMO interaction with this implementation. The full (three-page) output of this interaction can be found on `www.cs.otago.ac.nz/staffpriv/hans/sumpro/`.

The riddle is solved by updating the initial model `msnp` with the action models corresponding to the three successive announcements. Below, `showM (upds msnp [amrs1e, amrp2e, amrs3e])` is user input and the lines from `==> [0]` is the system response to that input.

```
*SNP> showM (upds msnp [amrs1e, amrp2e, amrs3e])
==> [0]
[0]
(0,[p4,q13])
(a,[[0]])
(b,[[0]])
```

The function `showM` displays a pointed epistemic model as:

```
==> [<points>]
[<domain>]
[<valuation>]
[<accessibility relations represented as equivalence classes>]
```

[7]We now use that $\varphi \overline{\vee} \psi$—where $\overline{\vee}$ is exclusive disjunction—entails that ($K\varphi \vee K\psi$ iff $(\varphi \rightarrow K\varphi) \wedge (\psi \rightarrow K\psi)$).

The list `[p4,q13]` represents the facts `P 4` and `Q 13`, i.e., the solution pair $(4, 13)$. Sum and Product have full knowledge (their access is the identity) on this singleton domain consisting of state `0`. That this state is named `0` is not a coincidence: after each update, states are renumbered starting from 0.

For another example, `(upds msnp [amrs1e,amrp2e])` represents the model that results from Product's announcement (*iii*) "Now I know it." Part of the `showM` results for that model are

```
*SNP> showM (upds msnp [amrs1e,amrp2e])
==> [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24
(...)
(0,[p2,q9])(1,[p2,q25])(2,[p2,q27])(3,[p3,q8])(4,[p3,q32])
(5,[p3,q38])(6,[p4,q7])(7,[p4,q13])(8,[p4,q19])(9,[p4,q23])
(...)
(a,[[0,3,6],[1,9,14,23,27,32,37,44,50],[2,10,17,24,28,38,45,46,51],[4
,11,18,29,33,39,47,55,60,65],[5,12,25,35,41,48,52,56,57,62,67,70,73],
[7],[8,22,36],[13,20,26,42,53,58,63,68,71,74,76,79,81],[15,19,30,34,4
0,61,66],[16,21,31,43,49,54,59,64,69,72,75,77,78,80,82,83,84,85]])
(b,[[0],[1],[2],[3],[4],[5],[6],[7],[8],[9],[10],[11],[12],[13],[14],
(...)
```

After two announcements, 86 pairs (x, y) remain possible. All remaining states are renumbered, from 0 to 85, of which part is shown. Product's (`b`) access consists of singleton sets only, of which part is shown. That should be obvious, as he just announced that he knew the number pair. Sum's (`b`) equivalence class `[0,3,6]` is that for sum 11: note that `(0,[p2,q9])`, `(3,[p3,q8])`, and `(6,[p4,q7])` occur in the shown part of the valuation. Sum's access has one singleton equivalence class, namely `[7]`. That corresponds to the state for pair $(4, 13)$: see `(7,[p4,q13])` in the valuation. Therefore, Sum can now truthfully announce to know the pair of numbers, after which the singleton final epistemic state (that was already displayed) results.

8.4 Versions of the riddle in DEMO

How versatile the model checker DEMO is, may become clear by showing how easily the program `SNP.hs` in Figure 2 can be adapted to accommodate different versions of the riddle, as discussed in the historical Section 2. The least upper bound for the Freudenthal version is 65. This we can check by replacing `upb = 100` in the program `SNP.hs` by `upb = 65`. More precisely we then also have to run the program for `upb = 64` after which it will appear that the model computed in `solutione = showM (upds msnp [amrs1e, amrp2e, amrs3e])` is now empty. The McCarthy version of the riddle can be checked by replacing

```
pairs = [(x,y)|x<-[2..100], y<-[2..100], x<y, x+y<=upb]
```

by

```
pairs = [(x,y)|x<-[2..100], y<-[2..100], x<=y, x<=upb, y<=upb]
```

Of course, the additions `x<=upb, y<=upb` are now superfluous, but, for another example, it is also easy to check that the least upper bound for which the McCarthy version has a solution (and now we can *not* remove `x<=upb, y<=upb`) is `upb = 62`. This we find when also changing the upper bound from 100 into 62 (and checking that `upb = 61` gives an empty model).

The interpreted system version of the model can be implemented by constructing the domain differently, namely as

```
pairs = [(v,w)|x<-[2..100],y<-[2..100],x<y,x+y<=upb,v=x+y,w=x*y]
```

and now, accessibility, later on, is simply defined as correspondence in the first argument for Sum and in the second argument for Product:

```
acc = [(a,w,v)| (w,(x1,y1))<-ipairs,(v,(x2,y2))<-ipairs,x1==x2]++
      [(b,w,v)| (w,(x1,y1))<-ipairs,(v,(x2,y2))<-ipairs,y1==y2]
```

In principle, one can also check that, when increasing the upper bound, more solutions or different solutions from $(4, 13)$ may emerge. One is then likely to run quickly into complexity problems; for that see the next section. The Plaza version with an unlimited model cannot be checked, as DEMO requires models to be finite. Also, as already mentioned, the epistemic formulas would be infinitary, which is not allowed either.[8]

Other epistemic riddles consisting of public announcements being made in some initial epistemic state, such as the Muddy Children problem, are similarly implemented by adapting the domain construction and the announcement formulas. For many examples, see [vE04].

8.5 Other model checkers

As mentioned in the introduction of Section 7, other epistemic model checkers with dynamic features include MCK [GvdM04] and MCMAS [RL04]. The question is whether we could also implement this problem in those model checkers. For the latest versions of these model checkers in both case the answer appears to be 'no'.

The current version of MCK is 0.2.0. In MCK, a state of the environment is an assignment to a set of variables declared in the environment section. These variables are usually assumed to be partially accessible to the individual agents, and agents could share some variables. The change of the state of the multi-agent system is either made by agents or the environment, in the form of changing these variables. There are two ways to make such changes. One is to send signals to the environment using the action construct by agents in conjunction with the transitions construct by the environment, which provides a way to describe how the environment variables are updated. The other is a specialized form for actions from the perspective that environment variables

[8]Obviously, Plaza used some finite approximation of the problem, but although [Pla89] mentions 'a program', it gives no details.

are shared variables, by providing read and write operations on those shared variables. In both cases, we need guarded statements to make the change. For example, a simple deterministic statement has the form:

$$\text{if } cond \to C \text{ [otherwise} \to C_o \text{] fi}$$

where command C is eligible for execution only if the corresponding condition *cond* evaluates to true in the current state. Otherwise, the command C_o will be executed. If we would like to model the Sum-and-Product problem in MCK, the effect of a public announcement should be recorded in a variable which is accessible to all agents. Suppose the effect of P's public announcement : "I now know it" ($K_P(x, y)$) is recorded in variable v. Then in a state just after this announcement, the variable v will be set to $True$ if $K_P(x, y)$ holds in the previous state, and otherwise to $False$. Clearly, we need that statement in the above *epistemic* form, with *cond* involving knowledge checking. Unfortunately, even though in MCK we can check epistemic postconditions, the current version of MCK does not support checking epistemic formulas as preconditions, as in *cond*. This might possibly be related to inherent difficulties to incorporate knowledge in *cond*, but an extension seems called for.

The latest MCMAS is version 0.7. The underlying theory has been developed by Lomuscio. It can be seen as a continuation of his PhD work on hypercube systems, which are a special class of interpreted systems [Lom99]. Similarly to MCK, MCMAS also does not support actions with knowledge-based preconditions to transit from one global state to another global state. Other recent work, by Su [Su04], was not included in our comparisons. However, his model checking results use an architecture similar to MCK and MCMAS.

9 Complexity

In this section, we analyze the complexity of finding solutions using our DEMO implementation. The theoretical boundaries are covered in Subsection 9.1. We also include some experimental results, in Subsection 9.2. Computational complexity of epistemic model checking is currently a focus of the research community; for temporal epistemic model checking we refer to [vdHW02b, vdHW02a, LR06, vdHLW06]. These results are as such inapplicable to our setting, because even apart from the different logical (namely temporal) setting, they also focus on other modelling aspects, e.g. [LR06] is not based on actual epistemic models but on succinct descriptions of such models in concurrent programs, and [vdHW02b] is more concerned with the complexities involved when reformulating planning problems in a model checking context.

9.1 Theoretical analysis

The Sum-and-Product problem is solved by updating the initial model with a sequence of three public announcements, i.e. by `upds msnp [amrs1e, amrp2e, amrs3e]`. Each such model restriction $M|\varphi$ requires determining the set $\{w \in$

$\mathcal{D}(M) \mid M, w \models \varphi\}$. Given a model M, a state w, and a formula φ, checking whether $M, w \models \varphi$ can be solved in time $O(|M| \times |\varphi|)$, where $|M|$ is the size of the model as measured in the size of its domain plus the number of pairs in its accessibility relations, and where $|\varphi|$ is the length of the formula φ. This result has been established by the well-known labelling method [HV91, FHMV95]. This method is based on dividing φ into subformulas. One then orders all these subformulas, of which there are at most $|\varphi|$, by increasing length. For each subformula, all states are labelled with either the formula or its negation, according to the valuation of the model and based on the results of previous steps. This is a bottom-up approach, in the sense that the labelling starts from the smallest subformulas. So it ensures that each subformula is checked only once in each state.

In DEMO v1.02, the algorithm for checking whether $M, w \models \varphi$ does not employ the bottom-up approach described above. Instead, it uses a top-down approach, starting with the formula φ and recursively checking its largest subformulas. For example, to check whether $M, w \models K_a\psi$, the algorithm checks whether $M, w' \models \psi$ for all w' such that $w \sim_a w'$, and then recursively checks the subformulas of ψ. This algorithm is $O(|M|^{|\varphi|})$, since each subformula may need to be checked $|M|$ times, and there are at most $|\varphi|$ subformulas of φ. So, theoretically, DEMO's algorithm is quite expensive.

In practice it is less expensive, because the Haskell language and its compiler and interpreter support a cache mechanism: after evaluating a function, it caches some results in memory, for reuse. For a study on the cache mechanism in Haskell programs we refer to [NM03]. Since it is hard to predict what results will be cached and for how long, we cannot give an estimate how much the cache mechanism influences our experimental results. But we can still show some interesting experimental results on the DEMO algorithm.

9.2 Experimental results

Our experimental results were based on a PC configured as Windows XP, AMD CPU 3000+ (1.8Ghz), 1G RAM. We used DEMO v1.02, and the Glasgow Haskell Compiler Interactive (GHCi) version 6.4.1, enabling the option ":set +s" to display information after evaluating each expression, including the elapsed time and number of bytes allocated.[9] We have run Windows XP in 'safe mode' to minimize the disturbance from other irrelevant processes when measuring computation.

More or less efficient formulas In Section 8 we observed that checking a formula such as $K_S \bigwedge_{(i,j)\in I}((x_i \wedge y_j) \rightarrow \neg K_P(x_i \wedge y_j))$, `fmrs1e` in Figure 2, is computationally cheaper than checking its (in T) logically equivalent form

[9]The allocation figure is only accurate to the size of the storage manager's allocation area, because it is calculated at every garbage collection. The RAM occupation is normally 60 Mbytes for GHCi when loading the SNP.hs and DEMO modules. For evaluating a particular expression, the figure might be quite large, for example in table 1, in the case of upb=80 and fmrs1e, the result is around 3219 Mb, due to the repeated garbage collection.

	fmrs1e		**fmrs1**	
`upb`	time(secs)	space(bytes)	time(secs)	space(bytes)
20	0.66	13,597,832	3.13	14,430,216
40	75.30	141,967,304	954.41	172,666,424
60	714.80	858,501,572	21,389.60	1,074,424,452
80	7,232.69	3,374,852,696	not available	not available

Table 1: Experimental results on formula checking

`upb`	| `msnp` |	time(secs)	space(bytes)
65	23,367	1,609.03	1,325,890,048
80	43,674	7,288.08	3,150,903,744
86	54,298	10,636.03	3,905,964,300
94	70,936	20,123.02	6,048,639,068
100	85,406	34,962.38	9,047,930,216

Table 2: Experimental results on the trend of time-space consumption

$K_S \neg \bigvee_{(i,j)\in I} K_P(x_i \wedge y_j)$, `fmrs1` in Figure 2. Our experiments confirm this result. In Table 1 we show the results for time and space consumption of function `upds msnp [public(fmrs1e)]` and `upds msnp [public(fmrs1)]` for different upper bounds `upb` in the initial model `msnp`: namely for `upb` 20, 40, 60 and 80. It is easy to see that checking with `fmrs1e` is substantially less costly than checking with `fmrs1` in terms of time, and slightly less costly in terms of space. We estimate that it may take more than a week to run the case `fmrs1` with `upb=80`. Cases 80 and 100 are only feasible for the more efficient form `fmrs1e`. Our next experiment is based on the more efficient `fmrs1e,fmrp2e,fmrs3e` only.

Trends for time and space consumption The smallest `upb` for which the Freudenthal version of the problem has a solution is **65**. We investigated the trend of time-space consumption between `upb`=65 and `upb`=100, as this relates to the size of the initial model `msnp`. The results for running `solutione` (see Figure 2) are shown in Table 2. In the figure, | `msnp` | is the size of the model `msnp`, measured as the number of states in the domain plus the number of pairs in the accessibility relations (for Sum and for Product). The proportional increase of the figures in Table 2 is clearer in Table 3, wherein they are normalized to the case `upb=65`. Note that time consumption increases faster than space consumption when the size of the model increases. Our sample is too small to consider other, more general, conclusions.

Finally, we also investigated the computational differences between the standard 'smaller/larger number' modelling of the puzzle (Section 4) and the 'interpreted system' modelling of the puzzle (Section 6). In that case we did not find

upb	| msnp |/N	time/N	space/N
65	1	1	1
80	1.87	4.53	2.38
86	2.32	6.61	2.95
94	3.04	12.51	4.56
100	3.65	21.73	6.82

Table 3: Normalization of the experimental results in Table 2.

significant results.

10 Conclusions

We have modelled the Sum-and-Product problem in public announcement logic and verified its properties in the epistemic model checker DEMO. The problem can be represented in the traditional way by number pairs, so that Sum knows their sum and Product their product, but also as an interpreted system with (sum, product) pairs. Subject to the union of languages, the representations are bisimilar, and even isomorphic. We also analyzed which announcements made towards a solution of the problem were unsuccessful updates—formulas that become false because they are announced.

A final word on model checking such problems: originally, an analysis involving elementary number theory and combinatorics was necessary to solve the problem. Indeed, that was the whole fun of the problem. Solving it in a model checker instead, wherein one can, in a way, simply state the problem in its original epistemic formulation, hides all that combinatorial structure and makes it appear almost trivial. Far from trying to show that the problem is therefore actually trivial or uninteresting, this rather shows how powerful model checking tools may be, when knowledge specifications are clear and simple but their structural ramifications are complex.

References

[BdRV01] P. Blackburn, M. de Rijke, and Y. Venema. *Modal Logic*. Cambridge University Press, Cambridge, 2001. Cambridge Tracts in Theoretical Computer Science 53.

[BM96] J. Barwise and L.S. Moss. *Vicious Circles*. CSLI Publications, Stanford, 1996.

[BM04] A. Baltag and L.S. Moss. Logics for epistemic programs. *Synthese*, 139:165–224, 2004. Knowledge, Rationality & Action 1–60.

[BMS98] A. Baltag, L.S. Moss, and S. Solecki. The logic of common knowledge, public announcements, and private suspicions. In I. Gilboa, editor, *Proceedings of the 7th Conference on Theoretical Aspects of Rationality and Knowledge (TARK 98)*, pages 43–56, 1998.

[FHMV95] R. Fagin, J.Y. Halpern, Y. Moses, and M.Y. Vardi. *Reasoning about Knowledge*. MIT Press, Cambridge MA, 1995.

[Fre69] H. Freudenthal. (formulation of the sum-and-product problem). *Nieuw Archief voor Wiskunde*, 3(17):152, 1969.

[Fre70] H. Freudenthal. (solution of the sum-and-product problem). *Nieuw Archief voor Wiskunde*, 3(18):102–106, 1970.

[Gar79] M. Gardner. Mathematical games. *Scientific American*, 241 (December):20–24, 1979. Also addressed in the March (page 24) and May (pages 20–21) issues of volume 242, 1980.

[Ger99] J.D. Gerbrandy. *Bisimulations on Planet Kripke*. PhD thesis, University of Amsterdam, 1999. ILLC Dissertation Series DS-1999-01.

[Ger06] J.D. Gerbrandy. The surprise examination in dynamic epistemic logic. *Synthese*, 2006. To appear.

[GG97] J.D. Gerbrandy and W. Groeneveld. Reasoning about information change. *Journal of Logic, Language, and Information*, 6:147–169, 1997.

[Gre68] M. H. Greenblatt. *Mathematical Entertainments: A Collection of Illuminating Puzzles New and Old*. George Allen and Unwin Ltd, London, 1968.

[GS58] G. Gamow and M. Stern. *Puzzle-Math*. Macmillan, London, 1958.

[GvdM04] P. Gammie and R. van der Meyden. MCK: Model checking the logic of knowledge. In R. Alur and D. Peled, editors, *Proceedings of the 16th International Conference on Computer Aided Verification (CAV 2004)*, pages 479–483. Springer, 2004.

[Hin62] J. Hintikka. *Knowledge and Belief*. Cornell University Press, Ithaca, NY, 1962.

[HV91] J.Y. Halpern and M.Y. Vardi. Model checking vs. theorem proving: a manifesto. In V. Lifschitz, editor, *Artificial intelligence and mathematical theory of computation: papers in honor of John McCarthy*, pages 151–176, San Diego, CA, USA, 1991. Academic Press Professional, Inc.

[Isa95] I.M. Isaacs. The impossible problem revisited again. *The Mathematical Intelligencer*, 17(4):4–6, 1995.

[Koo06] B.P. Kooi. On public update logics. *Journal of Applied Non-Classical Logics*, 2006. To appear.

[Lew69] D.K. Lewis. *Convention, a Philosophical Study.* Harvard University Press, Cambridge (MA), 1969.

[Lom99] A.R. Lomuscio. *Knowledge Sharing among Ideal Agents.* PhD thesis, University of Birmingham, Birmingham, UK, 1999.

[LR06] A.R. Lomuscio and F. Raimondi. The complexity of model checking concurrent programs against ctlk specifications. To be published in the Post-proceedings of DALT'06, the fourth workshop on Declarative Agent Languages and Technologies, 2006.

[McC90] J. McCarthy. Formalization of two puzzles involving knowledge. In Vladimir Lifschitz, editor, *Formalizing Common Sense : Papers by John McCarthy*, Ablex Series in Artificial Intelligence. Ablex Publishing Corporation, Norwood, N.J., 1990. original manuscript dated 1978–1981.

[Moo42] G.E. Moore. A reply to my critics. In P.A. Schilpp, editor, *The Philosophy of G.E. Moore*, pages 535–677. Northwestern University, Evanston IL, 1942. The Library of Living Philosophers (volume 4).

[NM03] N. Nethercote and A. Mycroft. The cache behaviour of large lazy functional programs on stock hardware. *SIGPLAN Notices*, 38(2 supplement):44–55, 2003.

[Pan91] G. Panti. Solution of a number theoretic problem involving knowledge. *International Journal of Foundations of Computer Science*, 2(4):419–424, 1991.

[Par92] R. Parikh. Finite and infinite dialogues. In E. Moschovakis, editor, *Workshop on Logic from Computer Science*, pages 481–498. MSRI Publications / Springer, 1992.

[Pla89] J.A. Plaza. Logics of public communications. In M.L. Emrich, M.S. Pfeifer, M. Hadzikadic, and Z.W. Ras, editors, *Proceedings of the 4th International Symposium on Methodologies for Intelligent Systems*, pages 201–216, 1989.

[RL04] F. Raimondi and A.R. Lomuscio. Verification of multiagent systems via ordered binary decision diagrams: An algorithm and its implementation. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 04)*, pages 630–637. IEEE Computer Society, 2004.

[Sal95] L. Sallows. The impossible problem. *The Mathematical Intelligencer*, 17(1):27–33, 1995.

[Sch72] S. Schiffer. *Meaning.* Oxford University Press, Oxford, 1972.

[Su04] K. Su. Model checking temporal logics of knowledge in distributed systems. In Deborah L. McGuinness and George Ferguson, editors, *Proceedings of the Nineteenth National Conference on Artificial Intelligence (AAAI 04)*, pages 98–103. AAAI Press / The MIT Press, 2004.

[vB98] J.F.A.K. van Benthem. Dynamic odds and ends. Technical report, University of Amsterdam, 1998. ILLC Research Report ML-1998-08.

[vB02] J.F.A.K. van Benthem. One is a lonely number: on the logic of communication. Technical report, University of Amsterdam, 2002. ILLC Research Report PP-2002-27 (material presented at the Logic Colloquium 2002).

[vdHLW06] W. van der Hoek, A.R. Lomuscio, and M.J. Wooldridge. On the complexity of practical atl model checking. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents & Multiagent Systems (AAMAS 2006)*, pages 201–208. ACM, 2006.

[vdHV02] W. van der Hoek and R. Verbrugge. Epistemic logic: a survey. In L.A. Petrosjan and V.V. Mazalov, editors, *Game theory and Applications*, volume 8, pages 53–94, 2002.

[vdHW02a] W. van der Hoek and M. Wooldridge. Model checking knowledge and time. In D. Bošnački and S. Leue, editors, *Model Checking Software, Proceedings of SPIN 2002 (LNCS Volume 2318)*, pages 95–111. Springer, 2002.

[vdHW02b] W. van der Hoek and M.J. Wooldridge. Tractable multiagent planning for epistemic goals. In *Proceedings of the First International Joint Conference on Autonomous Agents & Multiagent Systems (AAMAS 2002)*, pages 1167–1174. ACM, 2002.

[vDK06] H.P. van Ditmarsch and B.P. Kooi. The secret of my success. *Synthese*, 151:201–232, 2006.

[vdM94] R. van der Meyden. Mutual belief revision. In Jon Doyle, Erik Sandewall, and Pietro Torasso, editors, *Proceedings of the 4th International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pages 595–606. Morgan Kaufmann, 1994.

[vDRV05] H.P. van Ditmarsch, J. Ruan, and R. Verbrugge. Model checking sum and product. In *Proceedings of the 18th Australian Joint Conference on Artificial Intelligence (AI 2005)*, pages 790–795. Springer Verlag, 2005. LNAI 3809.

[vDvdHK03] H.P. van Ditmarsch, W. van der Hoek, and B.P. Kooi. Descriptions of game states. In G. Mints and R. Muskens, editors, *Logic, Games, and Constructive Sets*, pages 43–58, Stanford, 2003. CSLI Publications. CSLI Lecture Notes No. 161.

[vDvdHK06] H.P. van Ditmarsch, W. van der Hoek, and B.P. Kooi. Dynamic epistemic logic. Manuscript, 2006.

[vE04] J. van Eijck. Dynamic epistemic modelling. Technical report, Centrum voor Wiskunde en Informatica, Amsterdam, 2004. CWI Report SEN-E0424.

[vT56] G. van Tilburg. Doe wel en zie niet om (do well and don't look back). *Katholieke Illustratie (Catholic Illustrated Journal)*, 90(32):47, 1956. Breinbrouwsel 137 ('Brain Cracker' 137).

[WS40] W. T. Williams and G. H. Savage. *The Penguin Problems Book: A Modern Anthology of Perplexities and Tantalizers*. Penguin Books (Allen Lane), Harmondsworth, 1940.

[WS44] W. T. Williams and G. H. Savage. *The Second Penguin Problems Book*. Penguin Books, Harmondsworth, 1944.

[Yap06] A. Yap. Product update and looking backward. Technical report, University of Amsterdam, 2006. ILLC Research Report PP-2006-39.