Model Checking Sum and Product

H.P. van Ditmarsch[1] and J. Ruan[1⋆] and L.C. Verbrugge[2⋆⋆]

[1] University of Otago, New Zealand, {hans,jruan}@cs.otago.ac.nz
[2] University of Groningen, Netherlands, rineke@ai.rug.nl

Abstract. We model the well-known Sum-and-Product problem in a modal logic, and verify its solution in a model checker. The modal logic is public announcement logic. This logic contains operators for knowledge, but also 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.
The riddle is then implemented and its solution verified in the epistemic model checker DEMO. This can be done, we think, surprisingly elegantly. It involves reformulations to facilitate the computation.

1 Introduction

The following problem, or riddle, was first stated, in the Dutch language, in [1] and subsequently solved in [2]. 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:
> 1. P says: "I do not know it."
> 2. S says: "I knew you didn't."
> 3. P says: "I now know it."
> 4. S says: "I now also know it."
>
> Determine the pair (x, y).

This problem is considered a riddle, or puzzle, because the agents' announcements *appear* to be uninformative, as they are about ignorance and knowledge and not about (numerical) facts, whereas *actually* they are very informative: the agents learn facts from the other's announcements. For example, the numbers

⋆ 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).

cannot be 14 and 16: if they were, their sum would be 30. This is also the sum of 7 and 23. If those were the numbers their product would have been 161 which, as these are prime numbers, *only* is the product of 7 and 23. So 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 an example, to be able to determine the pair of numbers: the unique solution of the problem is the pair $(4, 13)$.

Versions of the problem – different announcements, different ranges for the numbers – elicited much discussion since its inception in [1]. This includes [3–5] and a website `www.mathematic.uni-bielefeld.de/~sillke/PUZZLES/logic-_sum_product` that contains many other references. More geared towards an epistemic logical audience are [6–11]. In [6] the problem is elegantly modelled in modal logic for processing 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. In [7] the problem is modelled in a dynamic epistemic logic that is the precursor of the public announcement logic presented here (namely without common knowledge). In [8] common knowledge involved in the Sum-and-Product problem is investigated in detail; [9] suggests a solution in temporal epistemic logic. As far as we know, we are the first to use an automated model checker to tackle the Sum-and-Product problem.

The knowledge that agents have about other agents' mental states 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. 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.

In Section 2 we introduce public announcement logic. In Section 3 we model the Sum-and-Product problem in public announcement logic. In Section 4 we introduce the epistemic model checker DEMO. In Section 5 we implement the Sum-and-Product specification of Section 3 in DEMO and verify its epistemic features.

2 Public Announcement Logic

Public announcement logic is a dynamic epistemic logic and is an extension of standard multi-agent epistemic logic. Intuitive explanations of the epistemic part of the semantics can be found in [12, 10, 11]. We give a concise overview of, in that order, the language, the structures on which the language is interpreted, and the semantics.

Given are a set of agents N and a set of atoms Q. The language of public announcement logic is inductively defined as

$$\varphi ::= q \mid \neg\varphi \mid (\varphi \wedge \psi) \mid K_n\varphi \mid C_G\varphi \mid [\varphi]\psi$$

where $q \in Q$, $n \in N$, and $G \subseteq N$ are arbitrary. For $K_n\varphi$, read 'agent n knows formula φ'. For $C_G\varphi$, read 'group of agents G commonly know formula φ'. For $[\varphi]\psi$, read 'after public announcement of φ, formula ψ (is true)'.

Next, we introduce the structures. An *epistemic model* $M = \langle W, \sim, V\rangle$ consists of a *domain* W of (factual) *states* (or 'worlds'), *accessibility* $\sim : N \to \mathcal{P}(W \times W)$, 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). For $\sim(n)$ we write $\sim_n$, and for $V(q)$ we write V_q. So, access $\sim$ can be seen as a set of equivalence relations $\sim_n$, and V as a set of valuations V_q. Given two states w, w' in the domain, $w \sim_n w'$ means that w is indistinguishable from w' for agent n 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 $\sim_G$ is the transitive and reflexive closure of the union of all access for the individuals in G: $\sim_G \equiv (\bigcup_{n\in G} \sim_n)^*$. This is access to interpret common knowledge for group G.

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

$$\begin{array}{ll}
M, w \models q & \text{iff } w \in V_q \\
M, w \models \neg\varphi & \text{iff } M, w \not\models \varphi \\
M, w \models \varphi \wedge \psi & \text{iff } M, w \models \varphi \text{ and } M, w \models \psi \\
M, w \models K_n\varphi & \text{iff for all } v \in W : w \sim_n v \text{ implies } M, v \models \varphi \\
M, w \models C_G\varphi & \text{iff for all } v \in W : w \sim_G v \text{ implies } M, v \models \varphi \\
M, w \models [\varphi]\psi & \text{iff } M, w \models \varphi \text{ implies } M|\varphi, w \models \psi
\end{array}$$

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

$$\begin{array}{l}
W' = \{w' \in W \mid M, w' \models \varphi\} \\
\sim'_n = \sim_n \cap (W' \times W') \\
V'_q = V_q \cap W'
\end{array}$$

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 [13], with precursors – namely for public announcement logic *without* common knowledge – in [7, 14]. For a concise completeness proof, see [11].

3 Sum and Product in Public Announcement Logic

We give a specification of the Sum-and-Product problem in public announcement logic. 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 $\{S, P\}$. 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 $K_P(x, y) \equiv \bigvee_{(i,j)\in I} K_P(x_i \wedge y_j)$. This is sufficient to formalize the announcements made towards a solution of the problem:

1. P says: "I do not know it": $\neg K_P(x, y)$
2. S says: "I knew you didn't": $K_S \neg K_P(x, y)$
3. P says: "I now know it": $K_P(x, y)$
4. 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 $V_{y_j} = \{(x, y) \in I \mid y = j\}$.

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

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

or, more properly expressing that $(4, 13)$ is the only solution, as the model validity

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

Note that announcement 1 by Product is superfluous in the analysis. The 'knew' in announcement 2, by Sum, refers to the truth of that announcement in the *initial* epistemic state, not in the epistemic state *resulting* from announcement 1, by Product.

Sum and Product as an interpreted system 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

$(x+y, xy) = (x'+y', x'y')$. Let without loss of generality x be the smaller of x and x', so that $x' = x + v$. Then from $xy = x'y' = (x+v)(y-v)$ follows that $yv - xv - v^2 = 0$, so that $v = 0$ or $v = y - x$. The second merely reverses the role of x and y; in our terms, it cannot be satisfied, because x was required to be strictly smaller than y. This observation paves the way for a different modelling of the problem than the traditional one with '(smaller number, larger number)' pairs (x, y).

We now let atomic propositions represent the sum and product of the different numbers, instead of representing these numbers. For example, 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 original formulation of the problem) 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 $V'_{p_{xy}} = \{(s, p) \in W' \mid p = xy\}$.

We have now modelled the problem as an *interpreted system* where agents at least know their local state. Interpreted systems were introduced in theoretical computer science as an abstract architecture for distributed systems [12]. Sum's local state is the sum of the two numbers, Product's local state is the product of the two numbers. A global state for the problem is a pair of local states, 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.

'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 similar to above:

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

An advantage of this representation is that we can apply known results for interpreted systems, such that agents at least know their local state, and the availability of characteristic formulas describing interpreted systems. That agent S knows its local state, means that S knows the sum of the two numbers, whatever they are: $s_{x+y} \rightarrow K_S s_{x+y}$. From this 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 $K_P s_{x+y}$.

Concerning the characteristic formula, we can apply results from [15].[3] The characteristic formula $\delta(\mathcal{SP}_{(s,p)})$ is defined as

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

The first conjunct of $\delta(\mathcal{SP}_{(s,p)})$ sums up the valuations of the different states in the domain. The second conjunct says (entails) that S knows its local state if and only if it considers possible any global state with that local state. For example $K_S s_{17} \leftrightarrow \neg K_S \neg(s_{17} \wedge p_{52})$; 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 traditional modelling of Sum and Product relates to the interpreted system modelling in a precise technical sense. Expand the language to one for all atoms for numbers x, y *and* for all atoms for 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 then follows that the models are also *bisimilar* [16] – a slightly weaker notion of 'sameness of models' that still guarantees that the theories describing the models are logically equivalent.) Without going into great detail, it suffices to define the isomorhphism 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.

[3] A characteristic formula 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 exist for finite epistemic models. We also have that $\delta(M, w) \leftrightarrow (\delta(w) \wedge C_N \delta(M))$, where $\delta(w)$ is *the description of state* w, for example summing up its valuation, or some other formula only true in w.

4 The Epistemic Model Checker DEMO

Recently, epistemic model checkers have been developed to verify properties of interpreted systems, knowledge-based protocols, and various other multi-agent systems. The model checkers MCK [17] and MCMAS [18] use the interpreted system architecture; MCK does this in a setting of linear and branching time temporal logic. The exploration of the search space in both MCK and MCMAS is based on ordered binary decision diagrams.

A different model checker, not based on a temporal epistemic architecture, is DEMO. It is developed by Jan van Eijck [19]. DEMO is short for Dynamic Epistemic MOdelling. It allows modelling epistemic updates, graphical display of Kripke structures involved, and formula evaluation in epistemic states. DEMO is written in the functional programming language Haskell.

The model checker DEMO implements the dynamic epistemic logic of [20]. In this 'action model logic' the global state of a multi-agent system is represented by an epistemic model as in Section 2. But an epistemic action can be more general than a public announcement, and 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 an operation called the *update product*. This is a restricted modal product. In this submission we restrict our attention to action models for public announcements. Such action models have a singleton domain, and the precondition of that point is the announced formula. We refrain from details and proceed with (a relevant part of – recursive clauses describing the effect of updates have been omitted) the recursive definition of formulas in DEMO.

```
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 announcement of φ,

```
module SNP
where
import DEMO

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

solution = showM (upds msnp [amrs1e, amrp2e, amrs3e])
```

Fig. 1. The DEMO program `SNP.hs`. Comment lines have been removed.

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.

5 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 1 contains the implementation.

A list is a standard data structure in Haskell, unlike a set. 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 the list

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

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 ]
```

where `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 2, where the valuation V was defined as a function mapping each atom to the set of states where it is true. The correspondence $q \in$ `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 the previous section 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.

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 $(x_i \wedge y_j) \to \neg K_P(x_i \wedge y_j)$.[4] The latter is computationally cheaper 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 $\bigwedge_{(i,j)\in I} \neg K_P(x_i \wedge y_j)$ can be replaced by one on $\bigwedge_{(i,j)\in I}((x_i \wedge y_j) \to \neg K_P(x_i \wedge y_j))$. 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 $\bigwedge_{(i,j)\in I}((x_i \wedge y_j) \to K_P(x_i \wedge y_j))$.[5] Using these observations, and writing an implication $\varphi \to \psi$ as $\neg\varphi \vee \psi$, the three problem announcements 2, 3, and 4 listed on page 4 are checked in DEMO in by the formulas `fmrs1e`, `fmrp2e`, and `fmrs3e`, respectively, as listed in Figure 1. The corresponding singleton action models are obtained by applying the function `public`, namely as `amrs1e = public (fmrs1e)`, `amrp2e = public (fmrp2e)`, and `amrs3e = public (fmrs3e)`. This is also shown in the figure.

Finally, we show a relevant part of DEMO interaction with this implementation. The complete (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:

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

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

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

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 indentity) 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 "Now I know it." Part of the `showM` results for that model are

[4] We use the $S5$-validity $\neg K\varphi \leftrightarrow (\varphi \to \neg K\varphi)$, that can be shown as follows: $\neg K\varphi$ iff $\neg\top \vee \neg K\varphi$ iff $(\varphi \vee \neg\varphi) \to \neg K\varphi$ iff $(\varphi \to \neg K\varphi) \wedge (\neg\varphi \to \neg K\varphi)$ iff $(\varphi \to \neg K\varphi) \wedge (K\varphi \to \varphi)$ iff (in $S5$!) $(\varphi \to \neg K\varphi)$.

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

```
*SNP> showM (upds msnp [amrs1e,amrp2e])
==> [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,
(...)
(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.

6 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 described the announcements made towards a solution of the problem as 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 complex.

References

1. Freudenthal, H.: (formulation of the sum-and-product problem). Nieuw Archief voor Wiskunde **3(17)** (1969) 152
2. Freudenthal, H.: (solution of the sum-and-product problem). Nieuw Archief voor Wiskunde **3(18)** (1970) 102–106
3. Gardner, M.: Mathematical games. Scientific American **241** (December) (1979) 20–24 Also addressed in the March (page 24) and May (pages 20–21) issues of volume 242, 1980.
4. Sallows, L.: The impossible problem. The Mathematical Intelligencer **17(1)** (1995) 27–33
5. Isaacs, I.: The impossible problem revisited again. The Mathematical Intelligencer **17(4)** (1995) 4–6
6. McCarthy, J.: Formalization of two puzzles involving knowledge. In Lifschitz, V., ed.: Formalizing Common Sense : Papers by John McCarthy. Ablex series in artificial intelligence. Ablex Publishing Corporation, Norwood, N.J. (1990) original manuscript dated 1978–1981.
7. Plaza, J.: Logics of public communications. In Emrich, M., Pfeifer, M., Hadzikadic, M., Ras, Z., eds.: Proceedings of the 4th International Symposium on Methodologies for Intelligent Systems. (1989) 201–216
8. Panti, G.: Solution of a number theoretic problem involving knowledge. International Journal of Foundations of Computer Science **2(4)** (1991) 419–424
9. van der Meyden, R.: Mutual belief revision. In Doyle, J., Sandewall, E., Torasso, P., eds.: Proceedings of the 4th international conference on principles of knowledge representation and reasoning (KR), Morgan Kaufmann (1994) 595–606
10. van der Hoek, W., Verbrugge, L.: Epistemic logic: a survey. In Petrosjan, L., Mazalov, V., eds.: Game theory and applications. Volume 8. (2002) 53–94
11. van Ditmarsch, H., van der Hoek, W., Kooi, B.: Dynamic epistemic logic. Manuscript (2005)
12. Fagin, R., Halpern, J., Moses, Y., Vardi, M.: Reasoning about Knowledge. MIT Press, Cambridge MA (1995)
13. Baltag, A., Moss, L., Solecki, S.: The logic of public announcements and common knowledge. Proceedings of TARK 98 (1998)
14. Gerbrandy, J.: Bisimulations on Planet Kripke. PhD thesis, University of Amsterdam (1999) ILLC Dissertation Series DS-1999-01.
15. van Ditmarsch, H., van der Hoek, W., Kooi, B.: Descriptions of game states. In Mints, G., Muskens, R., eds.: Logic, Games, and Constructive Sets, Stanford, CSLI Publications (2003) 43–58 CSLI Lecture Notes No. 161.
16. Blackburn, P., de Rijke, M., Venema, Y.: Modal Logic. Cambridge University Press, Cambridge (2001) Cambridge Tracts in Theoretical Computer Science 53.
17. Gammie, P., van der Meyden, R.: MCK: Model checking the logic of knowledge. In Alur, R., Peled, D., eds.: Proceedings of the 16th International conference on Computer Aided Verification (CAV 2004), Springer (2004) 479–483
18. Raimondi, F., Lomuscio, A.: Verification of multiagent systems via ordered binary decision diagrams: An algorithm and its implementation. In: 3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004), IEEE Computer Society (2004) 630–637
19. van Eijck, J.: Dynamic epistemic modelling. Technical report, Centrum voor Wiskunde en Informatica, Amsterdam (2004) CWI Report SEN-E0424.
20. Baltag, A., Moss, L.: Logics for epistemic programs. Synthese **139** (2004) 165–224 Knowledge, Rationality & Action 1–60.