January 2020

Estimating the maximum rise in temperature according to climate models using abstract interpretation

Peter Revesz
University of Nebraska- Lincoln, revesz@cse.unl.edu

Robert J. Woodward
University of Nebraska - Lincoln, rwoodwar@cse.unl.edu

Acta Univ. Sapientiae, Informatica **11,** 1 (2019) 5–23

DOI: 10.2478/ausi-2019-0001

Estimating the maximum rise in temperature according to climate models using abstract interpretation

Peter Z. REVESZ
University of Nebraska–Lincoln
Department of Computer Science & Engineering
Lincoln NE 68588-0115, USA
email: revesz@cse.unl.edu

Robert J. WOODWARD
University of Nebraska–Lincoln
Department of Computer Science & Engineering
Lincoln NE 68588-0115, USA
email: rwoodwar@cse.unl.edu

Abstract. Current climate models are complex computer programs that are typically iterated time-step by time-step to predict the next set of values of the climate-related variables. Since these iterative methods are necessarily computed only for a fixed number of iterations, they are unable to answer the natural question whether there is a limit to the rise of global temperature. In order to answer that question we propose to combine climate models with software verification techniques that can find invariant conditions for the set of program variables. In particular, we apply the constraint database approach to software verification to find that the rise in global temperature is bounded according to the common Java Climate Model that implements the Wigley/Raper Upwelling-Diffusion Energy Balance Model climate model.

1 Introduction

The ability to predict climate change, which has potentially a huge impact on life on earth, is affecting the legislation of countries and their mitigation efforts

Computing Classification System 1998: G.2.2
Mathematics Subject Classification 2010: 68R15
Key words and phrases: constraint database, Datalog, climate model, invariant, MLPQ system, software verification

around the world [8]. The predictions of the impacts of climate change rely heavily on the simulations of global climate models. Regional climate models offer a finer level of detail than the global climate models, and are sometimes used to determine the impact of climate on smaller regions. Climate models are calibrated using historical weather data. The model scenarios have been standardized by the *Intergovernmental Panel on Climate Change (IPCC)*, which was established in 1988 by the World Meteorological Organization and the United Nations Environment Programme.

The IPCC used several global climate models in its Third Assessment Report (TAR) [8] and its successor the Fourth Assessment Report (AR4) [2]. The TAR and AR4 assessment reports continue to be updated to include new information and research conducted since their dates. Chapter 9 of [8] defines that climate change simulations are to be assessed over the period from 1990 to 2100.

Current climate models, including the ones in TAR and AR4 [8, 2], are computer programs that use iterative methods to compute the values of climate variables, such as the rise in global average temperature above a baseline year, one year at a time for a fixed number of iterations. The computer programs become nonterminating when we drop the restriction of a fixed number of iterations. Nevertheless, we need to drop the restriction of a fixed number of iterations if we want to ask some of the most basic questions about climate change, such as "Will the global average temperature rise without a bound?" Saying that under a certain scenario of carbon emissions, the global average temperature will rise only one degree during the next twenty-five, fifty or seventy-five years is not satisfying. Our generation cannot claim to have found a sustainable, long-term solution, even a model of a solution, to the problem of climate change if the global average temperature rise cannot be bounded. The goal of this paper is to determine if there is a maximum invariant value for the global average temperature change from a baseline using software verification techniques.

This paper is organized as follows. Section 2 reviews some basic concepts, including the constraint database approach to software verification and the Java Climate Model used by the IPCC. Section 3 describes the climate model's implementation in the MLPQ constraint database system. Section 4 discusses the implications of the results. Section 5 summarizes related work. Finally, Section 6 gives some conclusions and future work.

2 Basic concepts

Next we review some concepts of the constraint database approach to software verification [16] and on climate models [8, 2].

2.1 Addition-bound matrixes or ABMs

Addition-bound matrixes, or ABMs, are designed to represent a set or conjunction of addition constraints and (lower and upper) bound constraints. The following definitions are based on the standard textbook description by Revesz [17].

Any set of addition, lower bound and upper bound constraints over the variables $V = \{x_1, \ldots, x_n\}$ is representable by a set of difference constraints over variables $V^+ = \{x_1^+, x_1^-, \ldots, x_n^+, x_n^-\}$. Note that in the difference constraint representation each variable x_i has two forms, namely a positive one, which is denoted by x_i^+ and a negative one, which is denoted by x_i^-. Here the first form is equivalent to x_i, while the second form is equivalent to $-x_i$. The logical equivalences shown below explain the rewriting of the constraints over V into constraints over V^+.

$$\begin{aligned} -x \geq b \quad &\equiv \quad x^- - x^+ \geq 2b \\ x \geq b \quad &\equiv \quad x^+ - x^- \geq 2b \\ x - y \geq b \quad &\equiv \quad x^+ - y^+ \geq b \\ x + y \geq b \quad &\equiv \quad x^+ - y^- \geq b \\ -x - y \geq b \quad &\equiv \quad x^- - y^+ \geq b \\ -x + y \geq b \quad &\equiv \quad x^- - y^- \geq b \end{aligned}$$

After applying the above rewriting rules, it may happen that we have two constraints $x - y \geq b$ and $x - y \geq c$. Suppose without loss of generality that $b > c$. Then $x - y \geq c$ can be deleted because it is implied by $x - y \geq b$. After similarly deleting all constraints that are implied by other constraints, for each pair of variables x and y, there can be only one difference constraint with $x - y$ on the left side.

Therefore any set of addition, lower bound and upper bound constraints over V is representable by an *ABM* A with rows and columns labeled by the elements of V^+. Further, the $A[i, j]$ entry of this ABM contains the right side constant of the difference constraint associated with the ith row and the jth column labels.

Next we show on an example set of constraints how it can be rewritten into an ABM. Suppose we have:

$$-x \geq -25,\ y \geq 3,\ x - y \geq 4,\ x + y \geq 10,\ -x - y \geq -40$$

then by using the above rewriting rules and simplifications, it can be represented by the following set of difference constraints:

$$x^- - x^+ \geq -50,\ y^+ - y^- \geq 6,\ x^+ - y^+ \geq 4,\ x^+ - y^- \geq 10,\ x^- - y^+ \geq -40$$

Finally, the ABM A below can represent the above set of difference constraints.

	x^+	x^-	y^+	y^-
x^+	$-\infty$	$-\infty$	4	10
x^-	-50	$-\infty$	-40	$-\infty$
y^+	$-\infty$	$-\infty$	$-\infty$	6
y^-	$-\infty$	$-\infty$	$-\infty$	$-\infty$

2.2 Operations on ABMs

Below we review the main ABM operators [17] that are used in later sections.

Definition 1 *Given two ABMs* A *and* B*, the minimum of* A *and* B*, denoted by* $A \vee B$*, is:*

$$[A \vee B]\,[i,j] = \left\{ \begin{array}{ll} A[i,j] & \text{if } A[i,j] \leq B[i,j] \\ B[i,j] & \text{if } B[i,j] < A[i,j] \end{array} \right\}$$

Definition 2 *Given two ABMs* A *and* B*, the widening of* A *and* B*, denoted by* $A \triangledown B$*, is:*

$$[A \triangledown B]\,[i,j] = \left\{ \begin{array}{ll} A[i,j] & \text{if } A[i,j] \leq B[i,j] \\ -\infty & \text{if } B[i,j] < A[i,j] \end{array} \right\}$$

Definition 3 D *is a domain of an ABM* A *if each entry* $A[i,j] \in D$*. When for some integer constants* l *and* u *each* $A[i,j]$ *is greater than or equal to* l *and less than or equal to* u *or is equivalent to* $-\infty$*, then* $\{-\infty\} \cup \{l, l+1, \ldots, u-1, u\}$ *is a domain of* A*.*

Definition 4 *Let* $l < 0$ *and* $u > 0$ *be two integer numbers and let* A *be an ABM with domain* $\{-\infty\} \cup \{l, l+1, \ldots, u-1, u\}$*. Given also another ABM* B*, the l-u-widening of* A *by* B*, denoted by* $A \diamondsuit_{l,u} B$*, is:*

$$[A \diamondsuit_{l,u} B]\,[i,j] = \left\{ \begin{array}{ll} A[i,j] & \text{if } A[i,j] \leq B[i,j] \\ B[i,j] & \text{if } l \leq B[i,j] < A[i,j] \\ -\infty & \text{if } B[i,j] < l \leq A[i,j] \end{array} \right\}$$

Example 5 (Revesz [17]) *Consider again* A *at the end of Section 2.1 and also the following ABM* B*:*

	x^+	x^-	y^+	y^-
x^+	$-\infty$	$-\infty$	15	10
x^-	-60	$-\infty$	$-\infty$	$-\infty$
y^+	$-\infty$	7	$-\infty$	2
y^-	$-\infty$	$-\infty$	$-\infty$	$-\infty$

Here $A \vee B$ *is:*

	x^+	x^-	y^+	y^-
x^+	$-\infty$	$-\infty$	4	10
x^-	-60	$-\infty$	$-\infty$	$-\infty$
y^+	$-\infty$	$-\infty$	$-\infty$	2
y^-	$-\infty$	$-\infty$	$-\infty$	$-\infty$

and $A \triangledown B$ *is:*

	x^+	x^-	y^+	y^-
x^+	$-\infty$	$-\infty$	4	10
x^-	$-\infty$	$-\infty$	$-\infty$	$-\infty$
y^+	$-\infty$	$-\infty$	$-\infty$	$-\infty$
y^-	$-\infty$	$-\infty$	$-\infty$	$-\infty$

Finally, $A \diamondsuit_{-50,50} B$*, that is when* $l = -50$ *and* $u = 50$*, gives:*

	x^+	x^-	y^+	y^-
x^+	$-\infty$	$-\infty$	4	10
x^-	$-\infty$	$-\infty$	$-\infty$	$-\infty$
y^+	$-\infty$	$-\infty$	$-\infty$	2
y^-	$-\infty$	$-\infty$	$-\infty$	$-\infty$

In addition to the above operators, we also consider the union operator $\cup$ of two ABMs. When A and B are AMBs, then the union operator $A \cup B$ simply returns the set of constraints that either A or B contains. The following theorem from [17] shows the relationship among the different AMB operators.

Theorem 6 (Revesz [17]) *Let* $\mathcal{S}$ *be the set of assignments to the variables that satisfy all the constraints of an ABM or union of ABMs. For any* $l < 0$ *and* $u > 0$*, the following holds:*

$$\mathcal{S}(A \cup B) \subseteq \mathcal{S}(A \vee B) \subseteq \mathcal{S}(A \diamondsuit_{l,u} B) \subseteq \mathcal{S}(A \triangledown B).$$

2.3 Abstract fixed point semantics

Each procedural program has a *collecting semantics*, which consists of a set of em invariants. Each invariant is associated with a line l in the procedural program and is intended to describe all possible values of all the variables when the program enters line l. Software verification is based on finding an over-approximation of the collecting semantics.

A general method to compute an over-approximation is called *abstract interpretation*. Abstract interpretation evaluates the procedural program by an *abstract execution* that starts with some abstract representation of the input data. The abstract execution at each entry of line l generalizes the invariant associated with l using a *widening operator* until the line invariant cannot be further widened.

A widening operator generalizes at a program location an invariant constraint A with some constraint B that describes an additional set of possible values of the program variables at that location. There are different types of widening operators proposed by various authors.

When we use the widening operator $A \Diamond_{l,u} B$, then it always leads to a terminating program execution. Note that $A \cup B$ is not a suitable widening operator because it may lead to a non-terminating abstract program execution.

Theorem 7 *Let P be a program with n integer (or rational) variables, only addition bound constraints on these variables, and k lines. Let l and u be two constants, and let an addition-bound matrix A_i be assigned to each line $1 \leq i \leq k$ of the program. Let each A_i contain no constraints initially, and as we execute line i of program P, widen A_i by the constraints B implied in line i using the widening operator $A_i \Diamond_{l,u} B$. Then the abstract program execution will terminate.*

Programs with integer and rational variables, if statements, go to statements, while statements, and assignment statements, where a variable is assigned the value of a linear arithmetic expression, can be represented as a Datalog program with constraints [9, 15]. The abstract program execution finds an *abstract fixed point* semantics, which will contain the least fixed point semantics [17]. The containment allows us to answer some questions about the possible values that variables in the program could take.

One can compute an abstract fixed point semantics of any climate change model that is equivalent to a complex computer program that would not terminate under normal program execution. If the abstract fixed point semantics of that computer program does not contain the possibility that the global

temperature reaches x degrees Fahrenheit, then we can conclude that according to that model the global temperature will not reach x degrees Fahrenheit. However, if the abstract fixed point semantics *contains* x as a possibility, then we cannot conclude anything definite because the abstract fixed point semantics may be an over-approximation of the least fixed point semantics, which actually does not contain x as a possibility.

The constraint database approach to software verification [16], which is a novel way to perform an abstract interpretation [4], uses the above idea to verify that a program functions correctly on a valid input by avoiding certain program states, where a program state is the values assigned to the variables in the program at a specific line of the program code [1, 16]. The Management of Linear Programming Queries (MLPQ) database [19, 1] is a constraint database that implements the above described widening operator and can be applied to Datalog programs with addition constraints. Hence we need to convert any computer program to a Datalog with addition constraint program as part of the constraint database approach to software verification.

2.4 Climate models

A climate system is a physical system that consists of five major components. The first component is the *atmosphere*, which is the air and space surrounding the earth. The second component is the *hydrosphere*, which is the water surrounding the earth. The hydrosphere is an important component because oceans form about two-third of the earth's surface. The third component is the *cryosphere*, which consists of the parts of the earth where water is frozen. This needs to be tracked separately from the oceans because the oceans and the cryosphere have very different physical properties in terms of absorption and reflection of sun light. The fourth component is the *land surface*, which is the part of the earth that is covered by land. The fifth and final component is the *biosphere*, which is the parts of the earth covered by living organisms.

Climate models try to model the climate system and predict some values for the climate, such as the following:

1. Land-surface temperature and land-surface air temperature.
2. Sea-surface temperature and ocean air temperature.
3. Land and sea combined temperature.
4. Sub-surface ocean temperature.
5. Upper air temperature.
6. Snow cover, including snowfall.

7. Sea-ice extent and thickness.

A good model considers all the possible types of interactions among the components. For example, the biosphere affects the concentration of carbon dioxide in the atmosphere [8]. Figure 1 shows a schematic diagram of a climate model.

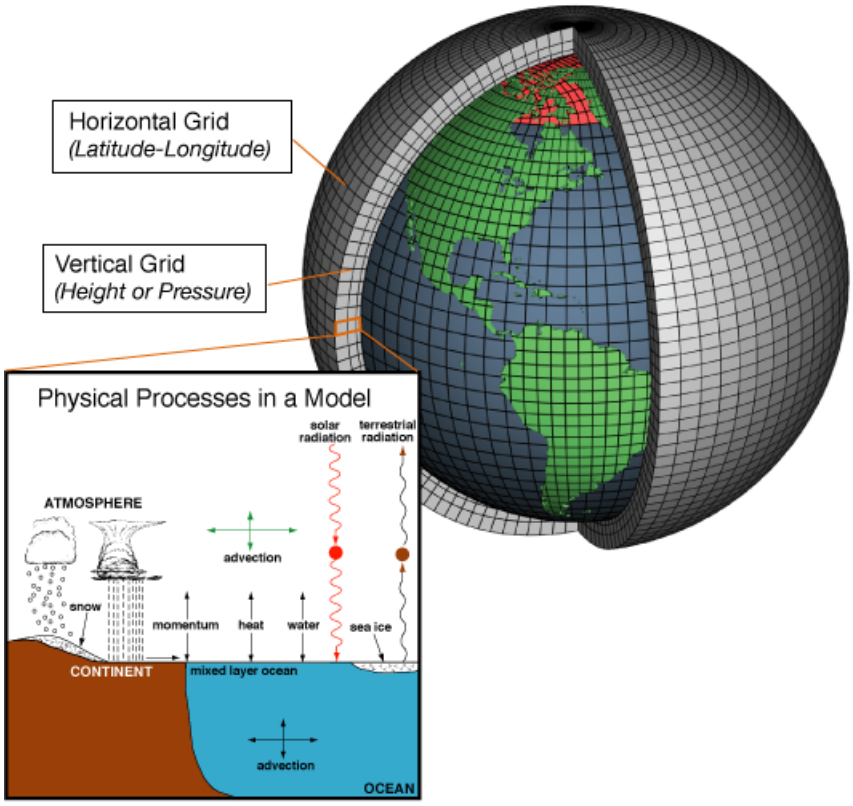

Figure 1: Some elements of a climate model from the Wikipedia entry "General Circulation Model."

A report of the UN's Intergovernmental Panel on Climate Change (IPCC) outlines some of the ways to accurately predict the above values [8]. The *Wigley/Raper Upwelling-Diffusion Energy Balance Model (UD/EBM)* climate model is a simple climate model that differentiates the hemispheres, and the land and ocean regions in each hemisphere [12]. The model uses heat flux equations to model the transfer from one year to the next and from one re-

gion to another. The UD/EBM climate model must be tuned to simulate an *atmosphere-ocean coupled general circulation model (AOGCM)*, without which it is not a complete model [8]. This combined model can iteratively compute each year's value, and the computation can be repeated without any termination.

The Java Climate Model (JCM)[1] implements the UD/EBM and was properly tuned to match a AOGCM [11]. Rather than using direct integration to compute the values for the heat fluxes, the JCM uses an eigenvector calculation method. This method finds the exact analytical solution, given the assumption that the non-linear fluxes change linearly within one time-step of a year [11].

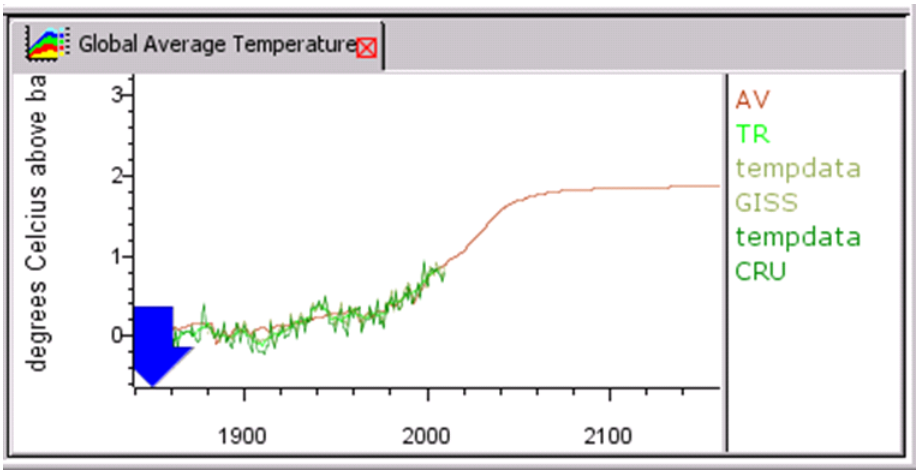

Figure 2: The global average temperature change given by the Java Climate Model.

The JCM was downloaded from http://jcm.climatemodel.info/. The SVN code repository for JCM was not operational, however, the source code was included inside of the distributed Java Archive (JAR) file. After extracting the source folders from the JAR file, a new project was created in NetBeans IDE 7.0.1 (`http://netbeans.org/`) and the source folders were imported. We needed to set up the following libraries:

- substance.jar – included in the JCM JAR.
- lucdata.jar – included in the JCM JAR.
- labdoc.jar – included in the JCM JAR.

[1] `http://jcm.climatemodel.info/`

- match-emitdata.jar – included in the JCM JAR.
- JCM.jar – included in the JCM JAR.
- javaws.jar – included in the Jave Runtime Environment (JRE) library folder.
- Jama-1.0.2.jar – downloaded from http://math.nist.gov/javanumerics/jama/.

The method that computes the average temperature change iteratively, one year at time, for the JCM is the ADJUST method, inside 'udebclimod.java,' and is shown in Algorithm 1. Line 1 of Algorithm 1 contains a for-loop to compute the global average temperature change for each year. The initial values of the variables used in ADJUST are set up in the SETUPFLUXES method, which is not shown here as their computation is not relevant. These intitial values were used as constants in our approach, which is discussed in more detail in Section 3.1. Figure 2 shows the predicted global average temperature change for each year given by the Java Climate Model until 2150. Note that the model seems to level off at a value around 2. However, there is no guarantee that the value is will leveled off at 2 or will spark later according to the model. Hence it is an important open question whether 2 is the maximum value. We will try to determine that in the rest of this paper.

3 The climate model's implementation in the MLPQ system

The goal of the experiment is to determine an invariant value on the average temperature change above a baseline year. Typically the value of the average temperature change above a baseline year is computed between 1990 and 2100 [8]. Instead, in our experiment the variant average temperature change above a baseline year will not depend on any year but will be an abstract fixed-point semantics upper-bound that will apply to all future years.

In this section, we first give an overview of how we will convert the ADJUST method from the Java Climate Model (JCM) code into Datalog to use with the MLPQ system, which can compute the abstract fixed point semantics to find all the possible values of the average climate temperature change. Second, we present the conversion process, showing the difference and gap-order constraints [13, 14]. Finally, we describe our implementation of the code using Datalog with constraints.

Algorithm 1: ADJUST

```
Input: numYears: Number of years in the future to compute the weather for
Output: The global temperature change computed for each year
for year = 0; year < numYears; year = year + 1 do
    guess = nstd + (nstd − nstdold);
    nstd = guess;
    nstdold = nstd;
    for o = 0; o < 2; o = o + 1 do
        for n = 0; n < nhb; n = n + 1 do
            hiq[o][n] = hpropf[o][n] ∗ hiq[o][n] + shicML[o][n] ∗ qinold[o];
        qinbase[o] = rf[o + 1];
        qinbase[o]+ = rf[o∗3]∗(frac[o∗3]/frac[o+1])∗klo/(kls∗frac[o∗3]+klo);
        qinbase[o]+ = spaceflux[o] ∗ qpt ∗ tstart;
    nit = 0;
    while |diff| > 0.01 && nit < 10 do
        for o = 0; o < 2; o = o + 1 do
            qin[o] = qinbase[o] + (o == 0? − 1.0 : 1.0) ∗ nstd ∗ kns/frac[o + 1];
            dqin = qin[o] − qinold[o];
            mlt[o] = 0;
            for n = 0; n < nhb; n = n + 1 do
                hiqi[o][n] = hiq[o][n] + rhicML[o][n] ∗ dqin; mlt[o]+ =
                hrML[o][n] ∗ hiqi[o][n];
            mlt[o]/ = qpt;
        diff = (mlt[0] − mlt[1]) ∗ cice − nstd; nstd+ = diff;
        nit = nit + 1;
    hiq = hiqi;
    qinold = qin;
    for o = 0; o < 2; o = o + 1 do mlt[o]− = tstart;
    bt[0][year] = mlt[0] ∗ cice ∗ klo + frac[0] ∗ rf[0])/(kls ∗ frac[0] + klo);
    bt[3][year] = mlt[1] ∗ cice ∗ klo + frac[3] ∗ rf[3])/(kls ∗ frac[3] + klo);
    bt[1][year] = mlt[0] ∗ cice;
    bt[2][year] = mlt[1] ∗ cice;
    globavtemp[year] = bt[0].get(year) ∗ frac[0] + bt[1].get(year) ∗ frac[1] +
    bt[2].get(year) ∗ frac[2] + bt[3].get(year) ∗ frac[3];
```

3.1 The experimental design

The code was examined to determine the constant values that do not change between iterations of the program (e.g., year-to-year). For the JCM, these values are typically the flux equations, or values that are specified by the user

to adjust the model to match an AOGCM. The default values were taken in this conversion to Datalog. Once these values were identified, because of the assumptions made by JCM to use the eigenvector calculation method instead of integration, all of the resulting equations were linear. Having linear equations was the goal of the model to study because MLPQ is a linear constraint database.

In the converted Datalog code, we refer to "linei" as the values of the variables at the start of line i of the program. Each line of the code was converted to Datalog to represent the change of values. These conversions are easy because we have only linear equations, For example, line 2 states the following:

```
guess=n+(n-nold)
```

and can be converted to Datalog:

```
line3(n,ndold,guess):- line2(n,ndold), guess=n+(n-ndold).
```

This Datalog code states that at the start of line 3, we are taking the same state as the start of line 2, except that we updated guess to have the value that is the value of the arithmetic expression on the right hand side of the equation in line 2 of the computer program.

The computer program we are converting from JCM contains two for-loops. The iterations of the two for-loops are independent from one-another. Since the method in JCM is called once for each year, after computing the last line in Datalog, we create a rule for the first line that propagates the values from the last line back as input. This creates the loop in the code that can then compute the abstract fixed point semantics. After the MLPQ system finished computing the values, we can look at the relation of the last line and see the possible values of the global average temperature change.

3.2 The coversion process

To give more details about the conversion process, we focus on the 29 lines of the ADJUST method in the JCM code (Algorithm 1), which was written in Java and consists of linear equations. The problem is that these linear equations are embedded in non-terminating for loops when we drop the condition which limits the numbers of years. In order to guarantee termination and execution in our Datalog with constraints program, we need to simplify the some parts of the computer program, where it contains either (1) global variables, or (2) large arrays. Lines 6 and 7 of the code offers a good example of these two types of simplifications:

for $n \leftarrow 0; n < nhb; n \leftarrow n + 1$ **do**
  $hiq[o][n] \leftarrow hpropf[o][n] * hiq[o][n] + shicML[o][n] * qinold[o];$

Global variables: The hpropf and shicML variables are both global variables, which are set from a different method call. These values are abstracted out as constants and loaded directly into the constraint database.

Large arrays: The for loop will iterate 40 times ($nhb = 40$, a constant) and thus the array for $hiq[o]$ will have 40 entries. Having that many variables seems too prohibitive as a first-step towards modeling the program. Therefore, we restrict $nhb = 3$, and only have three values in the $hiq[o]$ array.

Table 1 gives the gap-order constraints for Algorithm 1. In the table, the variable var_previous denotes the value of var from the previous line or previous iteration. Variables in all capital letters are treated as constants.

In a further simplification, we assumed that the ocean and the land areas for each the hemispheres took the same values. This simplification allowed the for-loops on line 5 and line 11 to be removed. However, this simplification was later removed by unrolling the content in the for-loops, once for each loop of the for-loop. The reason this simplification could be made was because the loops were independent from one another, which allowed the code to be unrolled.

Another simplification made, that is still in place, simplifies the for-loops of line 6 and 15 to only compute the first three values. Normally, these for-loops iterate over 40 values. All of the implementation details are in-place to remove this simplification.

3.3 A Datalog implementation

The simplified code was implemented in Datalog with the following steps: 1) allowing the insertion of constants into the Datalog program, 2) convert equations for MLPQ compatibility, and 3) allow more complicated arithmetic operations on constants (i.e., multiplication).

Prior to using the converter, we inserted constants into the Datalog program by fixing the variable assignment. For example, consider the constant $x = 123$:

```
CONST_X(x) :- 123
```

We found that using constants in this fashion over-complicated the program and caused significant overhead. Therefore, we wrote the converter such that

Line	Gap-Order Constraints
1	initialize rf, nstd, hiq, nstdold, qin, qinold
2	$guess - nstd - (nstd - nstdold) = 0$
3	$nstd - guess = 0$
4	$nstdold - nstd = 0$
5,6	For loop is unrolled
7	$hiq[o][n] - HPROPF[o][n] * hiq\_previous[o][n] - SHICML[o][n] * qinold[o] = 0$
8	$qinbase[o] - rf[o+1] = 0$
9	$qinbase[o] - qinbase\_previous[o] - (FRAC[o*3]/FRAC[o+1]) * KLO/(KLS * FRAC[o*3] + KLO) * rf[o*3] = 0$
10	$qinbase[o] - qinbase\_previous[o] * SPACEFLUX[o] * QPT * TSTART$
11	Not computed
12	Constraint posted on line 17
13	For loop is unrolled
14	$qin[o] - qinbase[o] - (o == 0? - 1.0 : 1.0) * KNS/FRAC[o+1] * nstd = 0$
15	$dqin - qin[o] + qinold[o] = 0$
16	$mlt[o] = 0$
17	For loop is unrolled
18	$hiqi[o][n] - hiq[o][n] - RHICML[o][n] * dqin \wedge mlt[o] - mlt\_previous[o] - HRML[o][n] * hiqi[o][n]$
19	$[1/QPT]mlt[o] = 0$
20	$diff - (mlt[0] - mlt[1]) * CICE + nstd = 0 \wedge nstd - nstd\_previous - diff = 0 \wedge diff > 0.001$
21	[not required]
22	$hiq - hiqi = 0$
23	$qinold - qin = 0$
24	$mlt[o] - mlt\_previous[o] = TSTART$
25	$bt[0] - ((CICE * KLO)/(KLS * FRAC[0] + KLO))mlt[0] - (FRAC0/(KLS * FRAC[0] + KLO))rf = 0$
26	$bt[3] - ((CICE * KLO)/(KLS * FRAC[3] + KLO))mlt[1] - (FRAC3/(KLS * FRAC[3] + KLO))rf = 0$
27	$bt[1] - CICEmlt[0] = 0$
28	$bt[2] - CICEmlt[1] = 0$
29	$globavtemp - FRAC[0] * bt[0] - FRAC[1] * bt[1] - FRAC[2] * bt[2] - FRAC[3] * bt[3] = 0$

Table 1: Conversion of Algorithm 1 to gap-order constraints.

it inserts the constants directly into the Datalog code. Note that we could have put the constants into the original Datalog program directly, but using a converter increases the readability of the code and gives us the ability to change the constants if required.

To accomplish multiplication, we tried to generalize an approach of multiplying two integer variables (See page 240 of [17]) to using floating point numbers, and first attempted to create a more general multiplication in Datalog as follows:

```
mult(x, y, z) :- y = 0, z = 0.
mult(x, y, z) :- y - y1 = 1, z - z0 - x = 0, mult(x, y1, z0).
```

However, in the above multiplication, where $x \times y = z$, the value of x is allowed to be a floating point number, but y is still required to be an integer. Since in some calculations both x and y need to be floating point numbers, we utilized the converter because all of our multiplications are on constants.

The converter has three parts:

1. A set of assignments used to convert constants to floating-point numbers. These assignments are stored in the 'ASSIGNMENTS' variable in the form 'VARIABLE=NUMBER'. VARIABLE is the text to search for, and NUMBER is a floating-point number that can be positive or negative. All fractions of numbers must have a leading '0' prior to the decimal point.

2. Converts double negation into a plus, for MLPQ compatibility. (E.g., $2 - -2$ becomes $2 + 2$.)

3. Evaluates arithmetic operations on numbers that are included in square brackets [ ]. This functionality allows more advanced arithmetic operations to be applied on constants (e.g., multiplication, division).

The steps in the converter could have been manually done when creating the Datalog file. However, the automated converter allows for more flexability when writing the Datalog code.

The converter script was created to work as a UNIX shell script. The script assumes a file named 'datalog.txt' is in the same working directory as the script, and will output a file named 'datalog_convert.txt' in the same working directory as the script. To run the script, simply type './convert.sh'. Note, the script requires the proper permissions set (e.g., chmod 700). **Note:** When using a Windows computer, the script might need to be converted not to have the Windows line returns (e.g., dos2unix convert.sh).

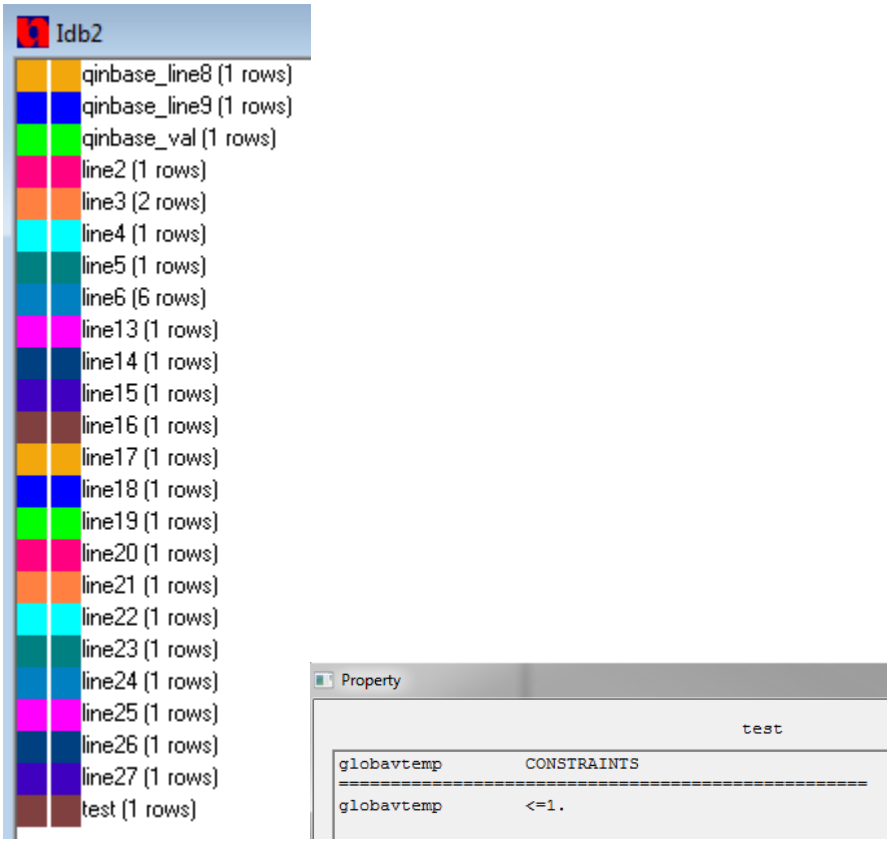

Figure 3: The relations loaded into MLPQ (left) and the resulting output of the global average temperature change above baseline of value one (right).

4 Discussion of the results

One of the problems we had early on when writing the Datalog code was not always determining when we had a typing error in one of the variable names in our code. MLPQ would then, correctly, interpret in the code the mistyped variable as a 'free-variable,' one that does not have any constraints on it. This problem caused errors early on in values not being computed properly. One way around this issue would be to use the Datalog Mode for Eclipse[2], which allows syntax highlighting for Datalog programs (but is not a Datalog interpreter).

[2] http://suif.stanford.edu/~livshits/work/datalogeditor/

Another struggle was getting MLPQ to properly load the relations. Some relations would take a huge amount of time for MLPQ to compute the value of the relation, which caused the program to look like it crashed. However, after waiting patiently, the program would load the relation. In order to get around this issue, we tweaked the order of the relations that were loaded and optimized the code by factoring out common code fragments and creating smaller relations.

In our tests, MLPQ returned the value of 1 for the bound on the global average temperature change as shown in Figure 3 for the results of the MLPQ system execution of the Datalog program. Although the idea of testing the long-range predictions of climate models using software verification is an intuitive and valid idea, the value of 1 should not be taken as conclusive because of the possible errors in the translation process from Java to Datalog and some simplifications we had to make to the original code. We need further tests of the algorithm to achieve confidence in its correctness and conclusions. Nevertheless, our experiment shows the soundness of the constraint database approach to being able to compute invariants for climate models.

5 Related work

There are a growing number of climate models. For example, the Intermediate Global Circulation Model (IGCM) (http://www.met.rdg.ac.uk/~mike/dyn_models/igcm/) implementing the baroclinic model of Hoskins and Simmons [7] and the Earth System Modeling Framework (ESMF) [3, 6] (http://www.earthsystemmodeling.org) are other climate models that are more complex than JCM.

A preliminary version of this paper was presented at [20]. To the best of our knowledge, no other researcher has previously attempted to compute an invariant value for any climate model. In fact, the calculation of invariants is not even considered in Chapter 9 of [8], where all simulations arbitrarily end at year 2100.

6 Conclusions and future work

This paper made the first attempt to investigate whether the climate models contain any inherent bounds. The paper combined climate modeling and software verification techniques, in particular software verification using the constraint database approach [1, 16]. Software verification techniques are able

to answer for even nonterminating computer programs what are the minimum and the maximum bounds on the variables.

The idea of combining climate models with abstract interpretation software verification techniques is a general contribution that is applicable to other climate models and other software verification techniques. Since the primary aim of our paper was only to show the feasibility of applying software verification techniques to testing the long-range implications of climate models, we started with the simpler JCM model. It remains a future work to investigate other combinations, for example using the IGCM or the ESMF climate models. Instead of being globally oriented like the JCM, some of the other climate models predict the future climate at a set of specific locations. These more refined climate models with values at specific locations at specific times may be also combined with spatio-temporal interpolation methods [5, 10, 18, 21] to generate temperature surface equations over each point of the globe.

References

[1] S. Anderson, P. Z. Revesz, CDB-PV: A constraint database-based program verifier, *Proc. of the 7th International Symposium on Abstraction, Reformulation and Approximation*, LNCS 4612, Springer, 2007, pp. 35–49. ⇒11, 21

[2] L. Bernstein et al., *Climate Change 2007: Synthesis Report*, Cambridge University Press, 2007. ⇒6, 7

[3] N. Collins et al., Design and implementation of components in the Earth System Modeling Framework, *International Journal of High Performance Computing Applications*, Fall/Winter 2005. DOI= 10.1177/1094342005056120. ⇒21

[4] P. Cousot, R. Cousot, Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints, *Proce. ACM Principles on Programming Languages*, ACM Press, 1977, pp. 238–252. ⇒11

[5] S. Haesevoets, B. Kuijpers, P. Z. Revesz, Affine-invariant triangulation of spatio-temporal data with an application to image retrieval, *ISPRS International Journal of Geo-Information* **6**, 4 (2017) 100. 37 pp. ⇒22

[6] C. Hill et al., Architecture of the Earth System Modeling Framework, *Computing in Science and Engineering* **6** 2004, Fall/Winter 2005. DOI= 10.1109/MCISE.2004.1255817. ⇒21

[7] B. J. Hoskins, A. J. Simmons, A multi-layer spectral model and the semi-implicit method, *Quarterly Journal of the Royal Meteorological Society* **101**, 429 (1975) 637–655. ⇒21

[8] J. T. Houghton et al. (editors), *Climate Change 2001: The Scientific Basis*, Cambridge University Press, 2001. ⇒6, 7, 12, 13, 14, 21

[9] P. C Kanellakis, G. M. Kuper, P. Z. Revesz, Constraint query languages, *Journal of Computer and System Sciences* **51,** 1 (1995) 26–52. ⇒10

[10] L. Li, P. Z. Revesz, Interpolation methods for spatio-temporal geographic data, *Computers, Environment and Urban Systems*, **28,** 3 (2004) 201–227. ⇒22

[11] B. Matthews, *Java Climate Model*, 2011. [Online]. Available: http://jcm.climatemodel.info/ ⇒13

[12] S. C. B. Raper, J. M. Gregory, T. J. Osborn, Use of an upwelling-diffusion energy balance climate model to simulate and diagnose A/OGCM results, *Climate Dynamics*, **17** (2001) 601–613. ⇒12

[13] P. Z. Revesz, A closed form evaluation for Datalog queries with integer (gap)-order constraints, *Theoretical Computer Science*, **116,** 1 (1993) 117–149. ⇒14

[14] P. Z. Revesz, Safe query languages for constraint databases, *ACM Transactions on Database Systems*, **23,** 1 (1998) 117–149. ⇒14

[15] P. Z. Revesz, *Introduction to Constraint Databases*, Springer, 2002. ⇒10

[16] P. Z. Revesz, The constraint database approach to software verification, *Proc. 8th International Conference on Verification, Model Checking, and Abstract Interpretation*, LNCS 4349, Springer, 2007, pp. 329–345. ⇒7, 11, 21

[17] P. Z. Revesz, *Introduction to Databases: From Biological to Spatio-Temporal*, Springer, 2010. ⇒7, 8, 9, 10, 19

[18] P. Z. Revesz, A recurrence equation-based solution for the cubic spline interpolation problem, *International Journal of Mathematical Models and Methods in Applied Sciences* **9** (2015) 446–452. ⇒22

[19] P. Z. Revesz, R. Chen, P. Kanjamala, Y. Li, Y. Liu, Y. Wang, The MLPQ/GIS constraint database system, *ACM SIGMOD Record*, **29,** 2 (2000) p. 601. ⇒11

[20] P. Z. Revesz, R. J. Woodward, Variable bounds analysis of a climate model using software verification techniques, *Proc. 13th International Conference on Software Engineering, Parallel and Distributed Systems*, Gdansk, Poland, 2014, pp. 31–36. ⇒21

[21] P. Z. Revesz, S. Wu, Spatiotemporal reasoning about epidemiological data, *Artificial Intelligence in Medicine*, **38,** 2 (2006) 157–170. ⇒22

Received: December 5, 2018 • Revised: April 10, 2019