The traces are sufficiently detailed to allow the verification of a vast range of complex, fine-grained properties on program behavior. A section of interesting real-life properties is proposed here. We expect to build up the benchmark with a greater range of properties.

We drew upon a wide variety of formalisms, including LTL properties, automata-based properties, and numerical properties.

Indeed, it is not clear if there exists a single specification language capable of stating each of the properties listed below. Nonetheless, mentioned in the accompanying readme file, several of the properties have been modeled using BeepBeep 3 processor chains, and are available for download.

The properties are explained in detail in the benchmark's accompanying proposal.

Property 1 (TetrisTrace): This property verifies that a tetris block, (called a tetragram) never enters the board game before the preeeceding block has stopped falling.

This property can be stated in LTL as:

$\phi \equiv \lnot \F(\texttt{hasFinishedFalling} \land \texttt{newPiece}) $

Property 2: (FractalTreeTrace): Early in the trace, a board is created, and the dimentions of this board are fixed in te creating method's parameter values. The aim of this property is to verify that the program never writes outside the board. This property could be stated using LTL-FO+ (other formalisms could be used as well) .

Property 2: (FractalTreeTrace): Early in the trace, a board is created, and the dimentions of this board are fixed in te creating method's parameter values. The aim of this property is to verify that the program never writes outside the board. This property could be stated using LTL-FO+ (other formalisms could be used as well) .

The size of the board is given in the 3rd and 4th parameters of the JFrame.panel() method.

The begin and end points of ant lines being drawn are given as the four paramters of the OnDraw() method.

Property3(ChatTrace): There are never more than N clients connected at the same time.
This property is given in TK-LTL as: phi \vDash \forall_{< N }(\C{\top}{\texttt{connect}} - \C{\top}{\texttt{disconnect}}).

Property 4 (ChatTrace): This is a numerical property, that whose presence in the behcnmark is to explore the limites of 2-valued and 3-valued logics. The goal if this "property" is to computing the sum of the length of these strings is useful for optimization purposes. These strings are included as the return values of 'return' lines from calls to StringBuilder(). (Attention, other methods may also include strings as return values; these should not be included in the count).

Property 5 & property 6(Encryption) isKey and isNotKey are a pair of program analysis properties that illustrate the expressiveness of the approach since few security property enforcement languages are sufficiently expressive to state both properties . The first property states that a given piece of information provided in the trace, such a parameter to a specific method or its return value, must never take the same value twice. Conversely, the isNotKey property imposes that any such value occurs in a matching pair. The tool allows users to specify the trace element to which they wish to apply the property. We verify both properties on trace 4, by checking that the same file is never encrypted twice.

Property7 (FractalTreeTrace) Next()-HasNext() is a very commonly used property is runtime verification research. It states that a call to Next() must always be preceded to a call to HasNext(). Over the same iterator. Multiple formalism of this property exists in the literature.

Property8 (SafeLock())Another commonly used property, SafeLock requires that the number of acquires and releases of a to an object are matched.