Test template processor

A test template is described as a hierarchy of blocks. Like any other hierarchical structure, a block can be defined recursively. I.e. a block is defined either as a *basic block* or as an ordered collection of blocks with specified parameters (see the «*Parameters*» section). A basic block is an instruction (or, to be more precise, an abstract instruction call with partially specified arguments). Here is an example of a block (ARM):

```
block {                 # upper level block
  block {                 # nested block
    add x0, x1, _, _        # basic block
    add x1, x2, _, _
    add x2, x3, _, _
  }
  block {
    sub x0, x1, _, _
    sub x1, x2, _, _
    sub x2, x3, _, _
  }
  block {
    cmp x0, 0, _
    cmp x1, 0, _
    cmp x2, 0, _
  }
}
```

Note: a non-basic block can be empty: **block** {}.

Each block implements a sequence iterator: iterators of non-basic blocks are constructed from iterators of nested blocks by applying operations specified by parameters (see the «*Parameters*» section) to them; a basic block iterator returns only one sequence that contains only one instruction call.

There are two special cases of block:

- *sequence*
- *atomic*
- *iterate*

Sequence

A block of the *sequence* type can be considered basic: its iterator returns only a single sequence that consists of instructions specified in the block. An example for ARM is below.

```
sequence {          # single sequence
  add x0, x1, _, _
  add x1, x2, _, _
  add x2, x3, _, _
}
```

Atomic

A block of the *atomic* type is similar for a *sequence* block. Its iterator also returns only a single sequence that consists of instructions specified in the block. An important distinction is that this sequence is atomic. This means that it will never be mixed with other instructions.

Iterate

A block of the *iterate* type iterates through sequences returned by its nested blocks. The number of sequences it returns equals $N_1 + \cdots + N_k$, where N_i is the number of sequences returned by the iterator of the i-th nested block. An example for ARM is below.

```
iterate {           # three sequences of one instruction
  add x0, x1, _, _
  add x1, x2, _, _
  add x2, x3, _, _
}
```

Parameters

Non-basic blocks can have parameters. They define the algorithm of constructing a block iterator from iterators of nested blocks. The list of supported parameters:

- *combinator* — method of combining sequences returned by nested block iterators (see «*Combinator*» section);
- *permutator* — method of modifying a sequence combination (see the «*Permutator*» section);
- *compositor* — method of composing a sequence (see the «*Compositor*» section);
- *rearranger* — method of rearranging sequences (see the «*Rearranger*» section);
- *obfuscator* — method of modifying a sequence (see the «*Obfuscator*» section).

A method of constructing a block iterator from nested block iterators is presented in the «*Description*» section.

Description

The test template processor does its job in the following steps:

1. iterating through instruction sequences;
    a. processing each of the sequences with one of the test processors (*default*, *branch* and *memory*).

Constructing an iterator of a non-basic block is performed by sequentially applying the following components:

1. *combinator* (see the «Combinator» section);
2. *permutator* (see the «Permutator» section);
3. *compositor* (see the «Compositor» section);
4. *rearranger* (see the «Rearranger» section);
5. *obfuscator* (see the «Obfuscator» section).

Note: it is possible to extend the tool with custom combinators, permutators, compositors, rearrangers and obfuscators.

Example

Facilities of combinator, permutator, compositor, rearranger and obfuscator are demonstrated in the following example:

```
# several combined sequences
block(
    :combinator => 'combinator-name',
    :permutator => 'permutator-name',
    :compositor => 'compositor-name',
```

```
    :rearranger => 'rearranger-name',
    :obfuscator => 'obfuscator-name') {
  # 3 sequences of length 1: {A11}, {A21}, and {A31}
  iterate { # block A
    A11,
    A21,
    A31,
  }
  # 2 sequences of length 2: {B11, B12}, and {B21, B22}
  iterate { # block B
    sequence { B11, B12 }
    sequence { B21, B22 }
  }
  # 1 sequence of length 3: {C11, C12, C13}
  iterate { # block C
    sequence { C11, C12, C13 }
  }
}
```

Combinator

Combinator is a component of the test template processor which combines sequences returned by iterators of nested blocks:

- *input*: ordered collection of sequence iterators;
- *output*: iterator of sequence combinations.

A *sequence combination* is a tuple of several sequences (their number matches the number of nested blocks).

Available combinators (possible values of *combinator-name*):

- $diagonal$ (see the «*Combinator* $diagonal$» section);
- $product$ (see the «*Combinator* $product$» section);
- $random$ (see the «*Combinator* $random$» section).

The default combinator is $diagonal$.

Combinator $diagonal$

Combinator $diagonal$ synchronously iterates over sequences retuned by nested blocks. Combining is finished when all of nested iterators are exhausted. Each time when a separate nested iterator is exhausted, it is reinitialized.

The number of combinations returned by combinator $diagonal$ equals $max(N_1, \ldots, N_k)$, where N_i is the number of sequences returned by the iterator of the i-th block.

Example

Combinator $diagonal$ returns the following combinations:

#1	***#2***	***#3***
$A11$	$A21$	$A31$
$B11$ $B12$	$B21$ $B22$	$B11$ $B12$
$C11$ $C12$ $C13$	$C11$ $C12$ $C13$	$C11$ $C12$ $C13$

Combinator $product$

Combinator $product$ constructs all possible combinations of sequences returned by iterators of nested blocks.

The number of combinations produced by combinator $product$ equals $N_1 \times \ldots \times N_k$, where N_i is the number of sequences returned by the iterator of the i-th block.

Example

Combinator $product$ returns the following combinations:

#1	#2	#3	#4	#5	#6
$A11$	$A11$	$A21$	$A21$	$A31$	$A31$
$B11$ $B12$	$B21$ $B22$	$B11$ $B12$	$B21$ $B22$	$B11$ $B12$	$B21$ $B22$
$C11$ $C12$ $C13$	$C11$ $C12$ $C13$	$C11$ $C12$ $C13$	$C11$ $C12$ $C13$	$C11$ $C12$ $C13$	$C11$ $C12$ $C13$

Combinator $random$

Combinator $random$ produces one random combination of sequences returned by iterators of nested blocks.

The number of combinations produced by combinator $random$ equals 1.

Example

Combinator $random$ can return the following combination:

#1
$A31$
$B21$ $B22$
$C11$ $C12$ $C13$

Permutator

Permutator is a component of the test template processor which modifies combinations returned by combinator by rearranging some sequences:

- *input*: iterator of sequence combinations;
- *output*: iterator of modified sequence combinations.

Available permutator (possible values of *permutator-name*):

- $trivial$ (see the «*Permutator* $trivial$» section);
- $random$ (see the «*Permutator* $random$» section).

The default permutator is $trivial$.

Permutator $trivial$

Permutator $trivial$ leaves each combination unchanged.

Example

Combinator $product$ и permutator $trivial$ applied together return exactly the same combinations as combinator $product$ applied alone:

#1	#2	#3	#4	#5	#6
$A11$	$A11$	$A21$	$A21$	$A31$	$A31$
$B11$ $B12$	$B21$ $B22$	$B11$ $B12$	$B21$ $B22$	$B11$ $B12$	$B21$ $B22$
$C11$ $C12$ $C13$	$C11$ $C12$ $C13$	$C11$ $C12$ $C13$	$C11$ $C12$ $C13$	$C11$ $C12$ $C13$	$C11$ $C12$ $C13$

Permutator $random$

Permutator $random$ changes the order of sequences in a combination in a random manner.

Example

Combinator $product$ and permutator $random$ applied together can return the following combinations:

#1	#2	#3	#4	#5	#6
$C11$ $C12$ $C13$	$B21$ $B22$	$C11$ $C12$ $C13$	$A21$	$C11$ $C12$ $C13$	$A31$
$B11$ $B12$	$A11$	$A21$	$B21$ $B22$	$B11$ $B12$	$C11$ $C12$ $C13$
$A11$	$C11$ $C12$ $C13$	$B11$ $B12$	$C11$ $C12$ $C13$	$A31$	$B21$ $B22$

Compositor

Compositor is a component of the test template processor that merges (multiplexes) sequences belonging to a combination into a single sequence preserving the initial order of instructions in each sequence:

- *input*: combination of sequences;
- *output*: merged sequence.

Available compositors (possible values of *compositor-name*):

- $catenation$ (see the «Compositor $catenation$» section);
- $rotation$ (see the «Compositor $rotation$» section);
- $random$ (see the «Compositor $random$» section).

The default compositor is $catenation$.

Compositor $catenation$

Compositor $catenation$ merges sequences by placing them one after another.

Example

$\langle\{A\},\{B\},\{C\}\rangle$
$A11$
$B11$ $B12$
$C11$ $C12$ $C13$

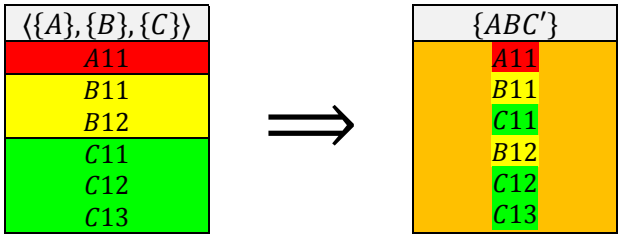
$\Longrightarrow$

$\{ABC\}$
$A11$ $B11$ $B12$ $C11$ $C12$ $C13$

Compositor $rotation$

Compositor $rotation$ merges sequences in a *round robin* fashion: the next in turn instruction of the next in turn sequence starting from the first sequence is added to the resulting sequence until all sequences are exhausted. If a separate sequence is exhausted, it is removed from processing.

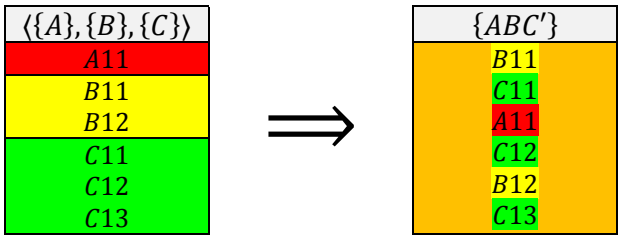
Example



Compositor $random$

Compositor $random$ merges sequences in a random manner preserving the initial order of instructions in each sequence.

Example



Rearranger

Rearranger is a component of the test template processor that rearranges sequences constructed by compositor:

- *input*: collection of sequences;
- *output*: modified collection of sequences.

Available rearrangers (possible values of *rearranger-name*):

- $trivial$ (see the «*Rearranger* $trivial$» section);
- $expand$ (see the « *Rearranger* $expand$» section).

The default rearranger is $trivial$.

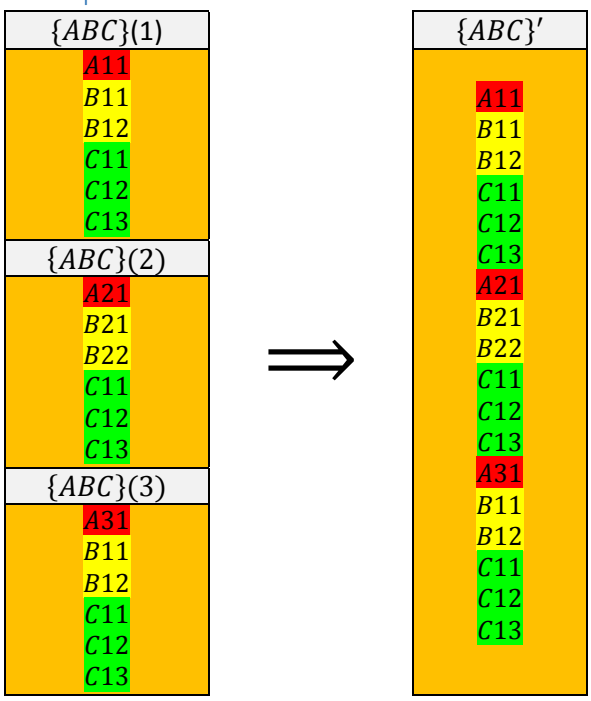
Rearranger $trivial$

Rearranger $trivial$ leaves the collection of sequences unmodified.

Rearranger $expand$

Rearranger $expand$ merges a collection of sequences into a single sequence by concatenating them.

Example



Obfuscator

Obfuscator is a component of the test template processor that modifies sequences returned by rearranger by permuting some instructions:

- *input*: sequence;
- *output*: modified sequence.

Available obfuscators (possible values of *obfuscator-name*):

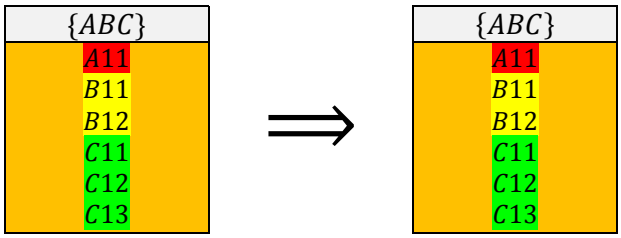
- $trivial$ (see the «*Obfuscator* $trivial$» section);
- $random$ (see the «*Obfuscator* $random$» section).

The default obfuscator is $trivial$.

Obfuscator $trivial$

Obfuscator $trivial$ leaves instruction sequence unchanged.

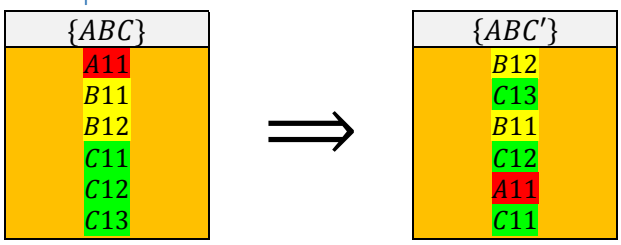
Example



Obfuscator $random$

Obfuscator $random$ changes the order of instructions in a sequence in a random manner.

Example



Requirements

At present, test templates must fulfill the following requirements:

1. Blocks *sequence* and *atomic* must not contain nested blocks.
2. Blocks *sequence* and *atomic* support only the *obfuscator* parameter.
3. Blocks *iterate* support only the *rearranger* and *obfuscator* parameters.

Restrictions

1. Blocks *block* with no parameters are temporary forbidden.
2. *Deprecated*: empty blocks are not supported.

Example

Let us illustrate how the test template processor works with the following parameter values:

- combinator *diagonal*;
- permutator *random*;
- compositor *catenation*;
- rearranger *trivial*;
- obfuscator *trivial*.

Combinator ***diagonal***	***Permutator*** ***random***	***Compositor*** ***catenation***	***Rearranger*** ***trivial***	***Obfuscator*** ***trivial***
Iterators of nested blocks A, B and C are initialized. A combination of next in turn sequences of blocks $A(1)$, $B(1)$ and $C(1)$ is constructed.	A random permutation of sequences is constructed	The sequences in the combination are concatenated	Collection of sequences is left unchanged	The sequence is left unchanged
$A11$ $B11$ $B12$ $C11$ $C12$ $C13$	$B11$ $B12$ $A11$ $C11$ $C12$ $C13$	$B11$ $B12$ $A11$ $C11$ $C12$ $C13$	$B11$ $B12$ $A11$ $C11$ $C12$ $C13$	$B11$ $B12$ $A11$ $C11$ $C12$ $C13$
Iterator of block C is exhausted; it is reinitialized. A combination of next in turn sequences of blocks $A(2)$, $B(2)$ and $C(1)$ is constructed.	A random permutation of sequences is constructed	The sequences in the combination are concatenated	Collection of sequences is left unchanged	The sequence is left unchanged
$A21$ $B21$ $B22$ $C11$ $C12$ $C13$	$C11$ $C12$ $C13$ $B21$ $B22$ $A21$	$C11$ $C12$ $C13$ $B21$ $B22$ $A21$	$C11$ $C12$ $C13$ $B21$ $B22$ $A21$	$C11$ $C12$ $C13$ $B21$ $B22$ $A21$

Iterators of blocks B and C are exhausted; they are reinitialized. A combination of next in turn sequences of blocks $A(3)$, $B(1)$ and $C(1)$ is constructed.	A random permutation of sequences is constructed	The sequences in the combination are concatenated	Collection of sequences is left unchanged	The sequence is left unchanged
$A31$ $B11$ $B12$ $C11$ $C12$ $C13$	$A31$ $B11$ $B12$ $C11$ $C12$ $C13$	$A31$ $B11$ $B12$ $C11$ $C12$ $C13$	$A31$ $B11$ $B12$ $C11$ $C12$ $C13$	$A31$ $B11$ $B12$ $C11$ $C12$ $C13$
Iterators of all nested blocks are exhausted; constructing combinations is finished.				