BPMN Check Validator

Este módulo tem por finalidade validar uma modelagem BPMN 2.0 a partir de um fluxo de execução pré-determinado.

A notação BPMN (Business Process Model and Notation) específica os processos de negócio de uma organização através de uma gramática de símbolos em um diagrama. Por ser um padrão convencionado, é utilizado uma API para a leitura dos modelos de entrada e fluxo de encadeamento de atividades e eventos, evitando assim o retrabalho em reescrever um padrão já implementado em várias APIs. Dentre as APIs disponíveis, à escolhida foi a Camunda (versão 7.8.0), o principal motivador da escolha é o fato da disponibilidade de uma versão open source. Por ser um framework Java-Based, Java se tornou o melhor candidato para o desenvolvimento deste módulo validador.

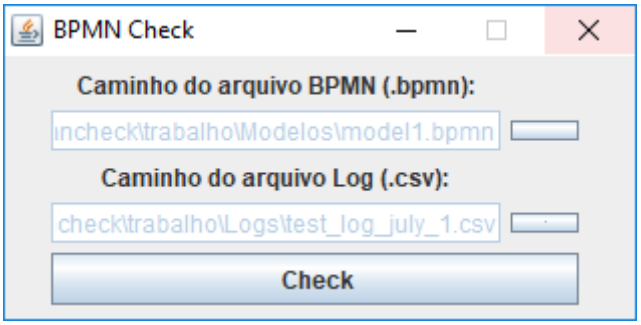
Instalação

É necessário compilar a solução para executá-la. Para tal, o projeto foi desenvolvido com um gerenciador de pacotes e compilação, chamado Maven. Com ele disponível na estação de trabalho e uma versão de JDK superior a 1.8.0 instalada, basta executar o comando a seguir dentro da pasta raiz do projeto que será gerado um executável java (.jar).

```
$ mvn package
```

Premissas

É esperado duas entradas, a primeira um arquivo no padrão BPMN 2.0 (extensão .bpmn) com a modelagem que então seguirá o fluxo especificado pela segunda entrada, um arquivo no formato Excel (extensão .CSV) aqui reconhecido como um Log, convencionado em duas colunas que especifica o Caso e o Evento respectivamente.



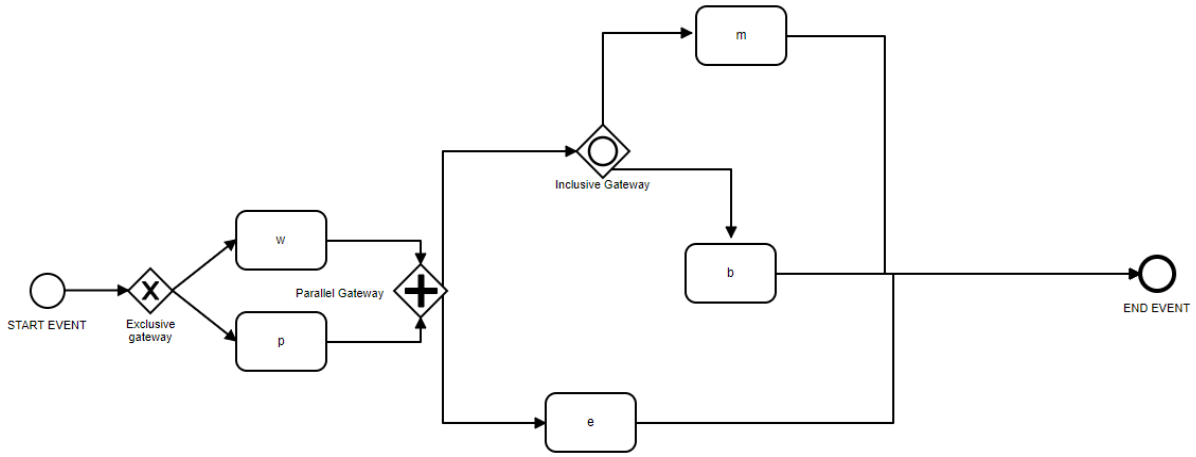
	A	B
1	case	event
2	1	w
3	1	e
4	1	m
5	2	w
6	2	c
7	2	s

No exemplo acima, irá validar para o Caso 1, a sequência de eventos na ordem de leitura, linhas 2, 3 e 4 eventos ‘w’,’e’ e ‘m’ e também o Caso 2, linhas 5, 6 e 7 eventos ‘w’, ‘c’ e ‘s’.

Já como saída o validador apresenta uma tela gráfica mostrando os casos válidos e inválidos além de separar em arquivos, no mesmo caminho do Log passado como entrada, adicionando ao nome do arquivo: ‘_true’ para os válidos e ‘_false’ para os não válidos.

Exemplo

Suponha o seguinte modelo (Representação Gráfica).



Para validar o caso 1, fluxo ‘w’, ‘e’, ‘m’.

	A	B
1	case	event
2	1	w
3	1	e
4	1	m

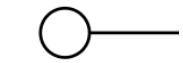
A primeira entrada, ou seja o modelo BPMN, é lido pela API Camunda e disposto em seu próprio objeto.

```
public BpmnModelInstance readFile(String path) {
    File file = new File(path);
    BpmnModelInstance modelInstance = Bpmn.readModelFromFile(file);
    return modelInstance;
}
```

A segunda entrada, o arquivo Excel, com o fluxo a ser validado é lido e armazenado em um objeto do tipo Log, criado especificamente para isso.

```java
public List<Log> readLogCsvFile(String path) {
    List<Log> logs = new ArrayList<Log>();

    try {
        File file = new File(path);
        InputStream fileStream = new FileInputStream(file);
        BufferedReader bufferFile = new BufferedReader(new InputStreamReader(fileStream));
        bufferFile.readLine(); //Pula o cabeçalho do arquivo
        String line = bufferFile.readLine();
        while (line != null) {
            Log log = new Log();
            log.setId(line.split(",")[0]);
            int index = logs.indexOf(log);
            if (index != -1) {
                logs.get(index).getEvents().add(line.split(",")[1]);
            } else {
                log.getEvents().add(line.split(",")[1]);
                logs.add(log);
            }
            line = bufferFile.readLine();
        }
        bufferFile.close();
    } catch (IOException e) {
        System.out.println("Erro na leitura do CSV.");
```

Start Node



Se ambas as entradas estiverem de acordo com as premissas e nenhuma exceção é gerada, é chegado o momento de percorrer o BPMN e validar o fluxo de execução. Para iniciar a varredura é necessário encontrar Start Node, responsável pelo início do processo.

```java
FlowNode startNode = null;
for (FlowNode flowNode : flowNodes) {
    if (flowNode.getClass() == StartEventImpl.class) {
        startNode = flowNode;
        break;
    }
}
```

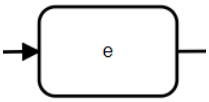
A busca é realizada a partir dos objetos disponibilizados pela API Camunda. Então a partir da lista de nós lidos do modelo de entrada, se algum desses nós for do tipo StartEvent.class temos então o ponto de partida.

Lógica Aplicada

A partir do Start Event é aplicado uma lógica algorítmica da seguinte forma: a API disponibiliza os nós posteriores ou anteriores à qualquer nó disposto em sua estrutura. Agora que o ponto de partida está definido, o algoritmo trabalha de maneira recursiva buscando os nós posteriores a cada nó encontrado na estrutura a fim de combinar com a sequência de eventos do caso que está sendo tratado.

Contudo, alguns intitulados nós pelo framework, na realidade são Gateways, Tasks ou ainda Eventos da notação BPMN, onde cada um deles precisa de um tratamento específico, por alterar o fluxo de execução.
Cada um desses nós gera uma nova chamada recursiva com a lista de nós posteriores a ele. Caso o uma dessas chamadas recursivas resulte em um único nó posterior de End Event (Evento Final) e a lista de eventos do caso que está sendo analisado já tenha sido completamente percorrida, o caso é considerado válido.

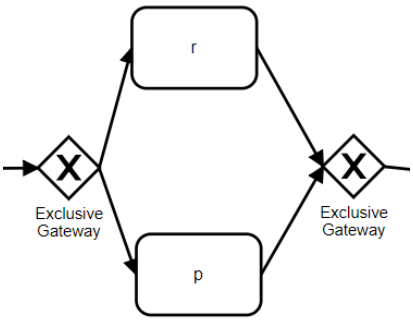
Tasks



São a base de comparação para a lista de eventos à ser validada. Caso o tarefa seja a mesma que está no topo da lista de eventos, o evento é removido da lista e tarefa gera uma nova chamada recursiva com seus nós posteriores.

```
for(Iterator<FlowNode> node = currentNodes.iterator(); node.hasNext();) {
    FlowNode currentNode = node.next();

    if(currentNode.getName().equals(currentLog)) {
        System.out.println("> " + currentNode.getName());
        countRemoveNode++;
```

Exclusive Gateway



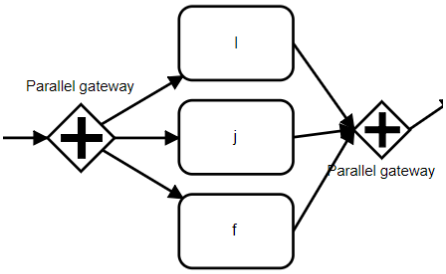
Só pode conter uma saída correta, ou seja o fluxo segue apenas um caminho.

```
if(currentNode.getClass() == ExclusiveGatewayImpl.class) {
    while(iterator.hasNext()) {
        SequenceFlow sequence = iterator.next();
        FlowNode newCurrentNode = sequence.getTarget();
```

Para tanto, é tratado cada saída posterior ao gateway exclusivo de forma que é salvo o estado atual dos nós e o fluxo segue a primeira saída encontrada, caso for a correta, ou seja associa com o evento atual do caso que está sendo validado, a execução segue normal. Caso contrário é retornado a execução e a próxima saída é avaliada da mesma forma. Se nenhuma das saídas se relacionarem com o fluxo que está sendo validado, o caso analisado é um não válido.

```
            List<FlowNode> newCurrentNodesCopy = new ArrayList<>(newCurrentNodes);

            if (!newCurrentNodes.contains(newCurrentNode)) {
                newCurrentNodes.add(newCurrentNode);
            }

            for(int i = (currentNodes.indexOf(currentNode) + 1); i < currentNodes.size(); i++) {
                if(!newCurrentNodes.contains(currentNodes.get(i))) {
                    newCurrentNodes.add(currentNodes.get(i));
                }
            }

            if (checkBpmnLogRecursive(modelBpmn, logEvents, newCurrentNodes)) {
                return true;
            } else {
                newCurrentNodes = new ArrayList<>(newCurrentNodesCopy);
                logEvents = new ArrayList<>(logEventsCopy);
            }
        }
        return false;
    }
```

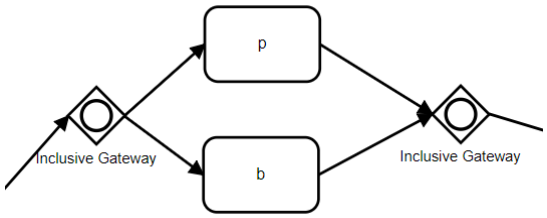
Parallel Gateway



Neste caso, todas as saídas formam uma execução paralela. É avaliado todos os nós posteriores e eles necessariamente precisam ser os próximos eventos do caso que está sendo validado.

```
            if(currentNode.getClass() == ParallelGatewayImpl.class) {
                if(isActiveNode(currentNode, currentNodes)) {
                    while (iterator.hasNext()) {
                        SequenceFlow sequence = iterator.next();
                        FlowNode newCurrentNode = sequence.getTarget();
                        if (!newCurrentNodes.contains(newCurrentNode)) {
                            newCurrentNodes.add(newCurrentNode);
                        }
                    }
                }
            }
```

Inclusive Gateway



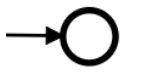
Para esse caso, onde pode conter apenas uma saída válida bem como todos os nós posteriores serem válidos, é aplicado uma regra de combinação com as saídas a fim de testar toda e qualquer possibilidade disponível.

```
if(currentNode.getClass() == InclusiveGatewayImpl.class) {
    if(isActiveNode(currentNode, currentNodes)) {
        List<FlowNode> newInclusiveNodes = new ArrayList<>();

        Collection<SequenceFlow> outgoingInclusive = currentNode.getOutgoing();
        Iterator<SequenceFlow> iteratorInclusive = outgoingInclusive.iterator();
        while (iteratorInclusive.hasNext()) {
            SequenceFlow sequence = iteratorInclusive.next();
            FlowNode newInclusiveNode = sequence.getTarget();
            if (!newInclusiveNodes.contains(newInclusiveNode)) {
                newInclusiveNodes.add(newInclusiveNode);
            }
        }

        InclusiveComb comb = new InclusiveComb(newInclusiveNodes, 0);
```

Basicamente é salvo o estado atual do nós e executado cada possível combinação dos nós posteriores, caso nenhuma dessas combinações resulte na sequência de eventos esperado, o caso é invalidado.

```
InclusiveComb comb = new InclusiveComb(newInclusiveNodes, 0);
while ( comb.hasNext() ) {
        List<FlowNode> listCombInclusiveNodes = new ArrayList<>();
        listCombInclusiveNodes = comb.next() ;

        List<FlowNode> newCurrentNodesCopy = new ArrayList<>(newCurrentNodes);

    for ( FlowNode e : listCombInclusiveNodes ) {
        newCurrentNodes.add(e);
    }

    for(int i = (currentNodes.indexOf(currentNode) + 1); i < currentNodes.size(); i++) {
            if(!newCurrentNodes.contains(currentNodes.get(i))) {
            newCurrentNodes.add(currentNodes.get(i));
        }
    }

    if (checkBpmnLogRecursive(modelBpmn, logEvents, newCurrentNodes)) {
        return true;
    } else {
        newCurrentNodes = new ArrayList<>(newCurrentNodesCopy);
        logEvents = new ArrayList<>(logEventsCopy);
    }
}
return false;
```

End Node



Evento Final, onde se à lista de eventos que está sendo analisada já tenha sido completamente percorrida, o caso é considerado válido.

BPMN - Elementos Suportados

Start Event - Evento inicial dá início ao fluxo de execução.
End Event - Evento final dá fim ao fluxo de execução.
Gateways - Exclusive, Parallel e Inclusive são suportados.
Tasks - Tarefas são suportadas.

Limitações

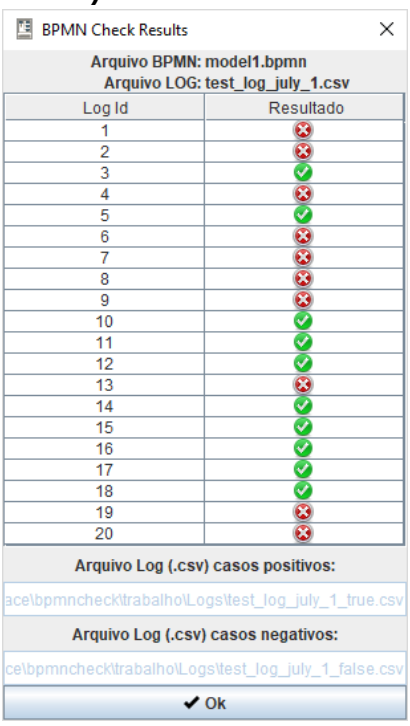
No decorrer da implementação foi apresentado um problema nos nós Parallel, bem como para os casos em que a combinação de nós posteriores ao Inclusive Gateway é maior que um. Nesses casos, onde existem saídas paralelas para um nó, é esperado que o fluxo dessas duas ou mais saídas em algum momento se encontrem novamente (em último caso essa união irá ocorrer para o evento final). Porém, até esse reencontro, os fluxos podem ter

tamanhos diferentes, onde um dos fluxos pode chegar ao ponto de encontro primeiro, ao chegar ao ponto de encontro primeiro é preciso esperar até que o outro fluxo (podendo ser mais de um) chegue a esse mesmo ponto para se dar continuidade na execução do processo.
Para tentar solucionar esse problemas, foi criado um conceito de nó ativo e implementado a seguinte a solução: quando um nó tem mais de um nó antecessor a ele, ou seja é um ponto de encontro de fluxos, é verificado se seus antecessores e consequentemente os precedentes aos antecessores (verificação em 2 níveis apenas) são nós ativos. Para ser um nó ativo, essa verificação não pode retornar nenhum nó que ainda esteja na lista de eventos do caso em análise, ou seja se algum nó antecessor ao encontro de fluxos ainda está na lista de eventos a serem verificados, é porque os fluxos têm tamanhos diferentes e é preciso aguardar. Contudo a limitação fica em ser verificado em apenas dois níveis antecessores, isso porque, existem casos com mais de dois níveis, onde é gerado um loop infinito.

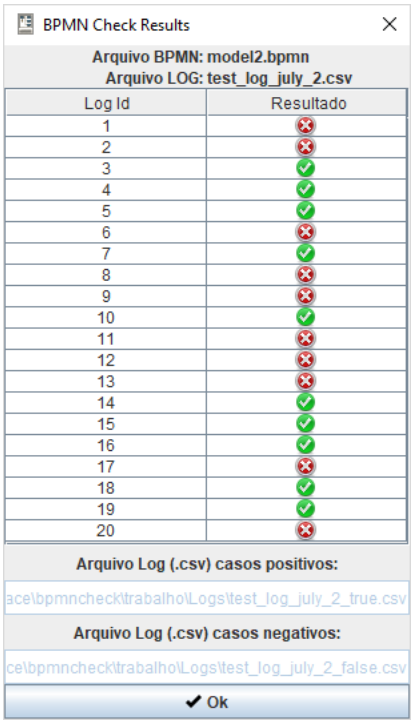
Caso de Estudo Aplicado

Foram disponibilizados 10 modelos BPMN (numerados de 1 a 10) e 10 arquivos Log (numerados de 1 a 10), um modelo para seu respectivo arquivo Log. Onde cada Log contém 20 casos a serem analisados. Entre esses 10 Logs, 4 deles foram analisados manualmente, sendo eles o 2, 3, 6 e 7, a fim de confrontar os resultados e comprovar a eficiência da ferramenta. Para esses casos analisados manualmente a ferramenta apresentou 100% de corretude. Segue resultados dos 10 modelos.

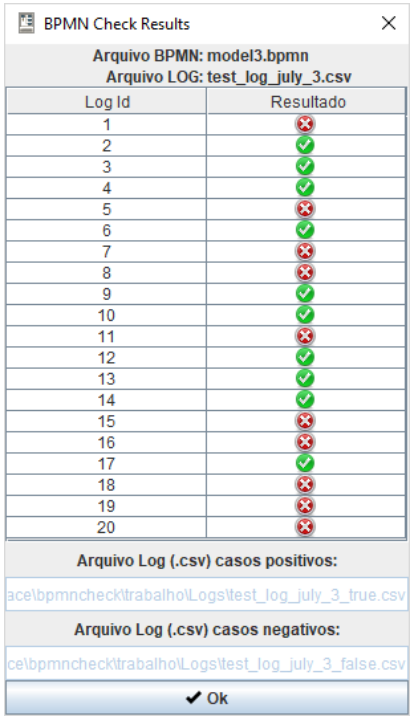
Model 1) 10 válidos e 10 inválidos



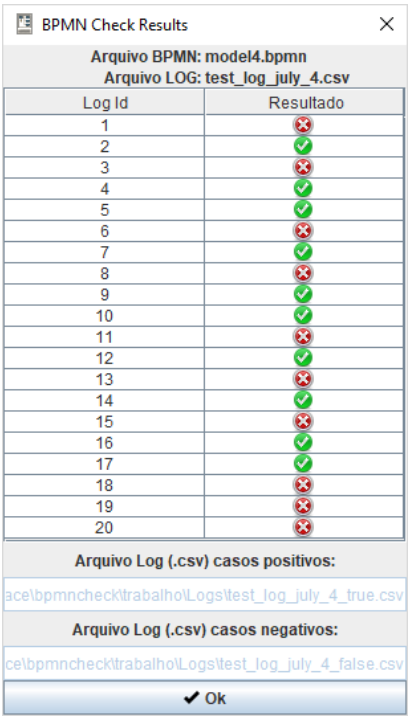
Model 2) 10 válidos e 10 inválidos



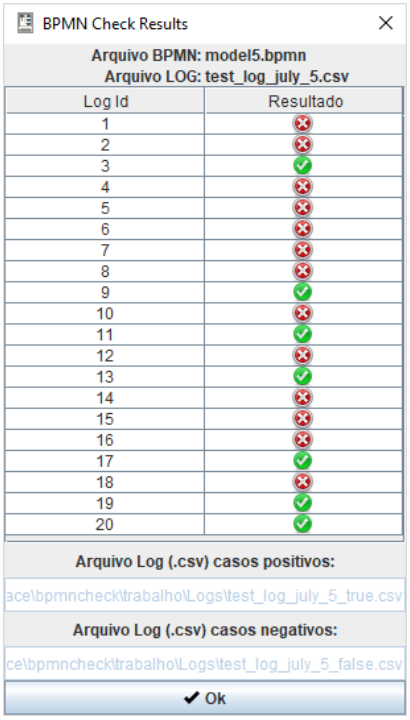
Model 3) 10 válidos e 10 inválidos



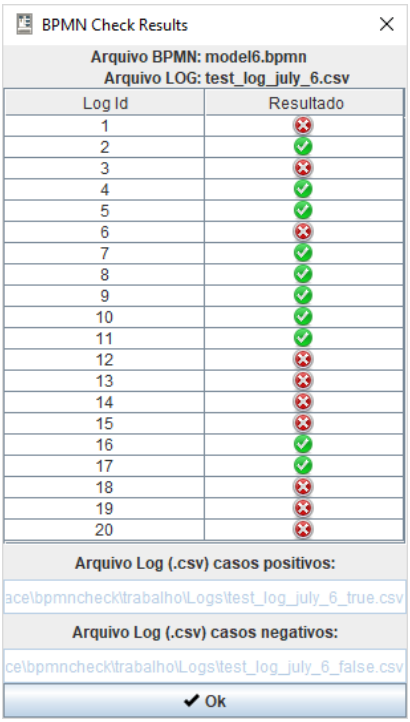
Model 4) 10 válidos e 10 inválidos



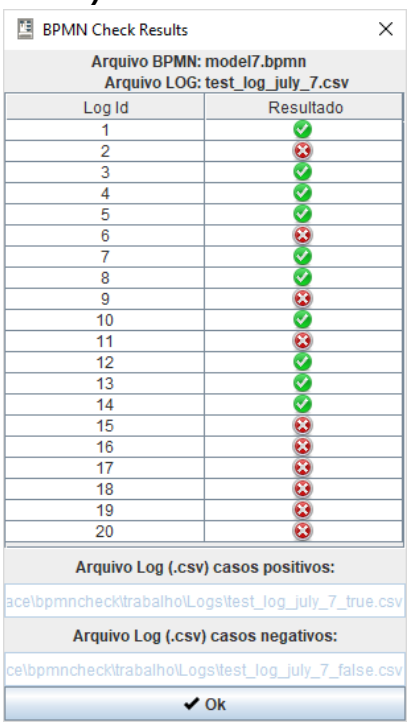
Model 5) 7 válidos e 13 inválidos



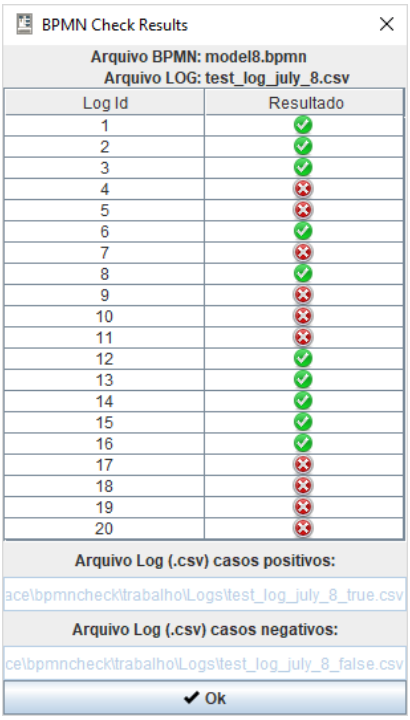
Model 6) 10 válidos e 10 inválidos



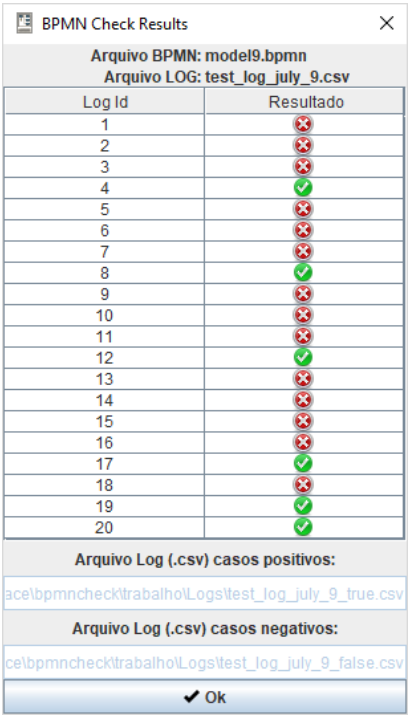
Model 7) 10 válidos e 10 inválidos



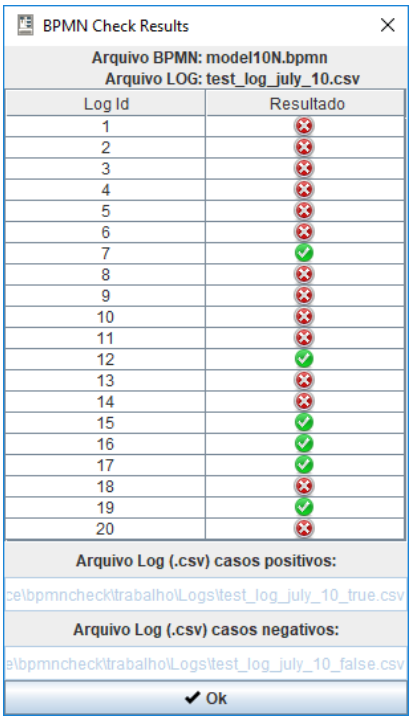
Model 8) 10 válidos e 10 inválidos



Model 9) 6 válidos e 14 inválidos



Model 10) 6 válidos e 14 inválidos



Contudo, por se tratar de um estudo controlado, mesmo sem saber quais casos, já era conhecido que cada arquivo Log deveria apresentar 10 casos válidos e 10 inválidos, o que não aconteceu com os modelos 5, 9 e 10, devido às restrições mencionadas.