This spec models the behavior of a build graph execution worker.

It assumes that there is a build queue, where nodes are consumed from the front to the back, and have been inserted in topological sorting (so the first node to be consumed has no dependencies).

Each build graph node in the queue can be in one of the following states :
- "waiting", or waiting o be picked up by a build worker
- "building", meaning it is currently being built by a worker
- "built", meaning it was built correctly by a worker
- "cached", meaning the worker identified a cache-hit
- "errored", meaning the worker found a problem with this build node

This tiny state machine allows workers to pull work from a queue, and check if the dependencies of their current job are met or not. If they are, they will proceed to build it, if they aren't, then they need to wait.

Actually building a node may be met with a Cache hit. This wasn't strictly necessary to model since the result is a built node nonetheless.

In this spec we don't quite care how the build graph is built, so we'll explore both the case when some node checks for dependencies and doesn't. This should be enough to figure out how the actual worker will behave with a real build graph. Abstraction for the win.

At the end of the execution, the queue should be emptied, and all nodes should be built or cached OR the process should have been aborted with some *errored* nodes.

```
29 ──────── MODULE Crane_BuildGraphExecution ────────
30 EXTENDS Naturals, Integers, Sequences

32 CONSTANT Nodes
33 CONSTANT Workers
34 ASSUME NodesInRange ≜ Nodes ∈ Nat ∧ Nodes ≥ 1
35 ASSUME WorkersInRange ≜ Workers ∈ Nat ∧ Workers ≥ 1

38   --algorithm build_graph_execution

40 variables
41     abort = FALSE,
42     queue = Nodes,
43     nodes = [n ∈ 1 .. Nodes ↦ "waiting"],
44     work_done = {}
45 ;

47 define
48     TypeInvariant ≜ queue ∈ Nat

50     Statuses ≜ {nodes[n] : n ∈ 1 .. Nodes}
51     AllBuiltOrCached ≜ Statuses ⊆ {"built", "cached"}
52     SomeErrored ≜ "errored" ∈ Statuses

54     EventuallyQueueIsConsumed ≜ ◇□(abort ∨ (¬abort ∧ queue = 0))
55     NoWorkIsDoneTwice ≜ ◇□(abort ∨ (¬abort ∧ work_done = 1 .. Nodes))
56     EitherWeAbortOrThereAreNoErrors ≜ ◇□( (abort ∧ SomeErrored) ∨ (¬abort ∧ AllBuiltOrCached))
```

```
57  end define ;

59  fair process Worker ∈ 1 .. Workers
60  variables
61      current_job = −1
62  ,
63  begin
64      Loop:
65          if abort ∨ queue = 0 then
66              goto Done ;
67           else

69          current_job := queue ;
70          nodes[current_job] := "building" ;
71          queue := queue − 1 ;
72          goto Work ;
73       end if ;

75      Work:
76          when current_job ≠ −1 ;

78           either CheckOnDependencies:
79                  if current_job < Nodes ∧ (nodes[current_job + 1] = "built" ∨ nodes[current_job + 1] = "cached")
80                    goto BuildNode ;
81                   else
82                       WaitForDependency:
83                        while ¬abort ∧ current_job < Nodes ∧ nodes[current_job + 1] = "building" do
84                            skip ;
85                        end while ;
86                        if ¬abort then goto BuildNode ;
87                         else goto Done ;
88                        end  if ;
89                    end if ;
90            or BuildNode:
91                  either    WorkSucceeds:
92                      nodes[current_job] := "built" ;
93                      work_done := work_done ∪ {current_job} ;
94                      goto Loop ;
95                  or CacheHit:
96                       nodes[current_job] := "cached" ;
97                       work_done := work_done ∪ {current_job} ;
98                       goto Loop ;
99                  or BuildError:
100                       nodes[current_job] := "errored" ;
101                      abort := TRUE ;
102                      goto Loop ;
```

```
103             end either ;
104     end either ;
105 end process ;

107 end algorithm ;
108  BEGIN TRANSLATION − the hash of the PCal code: PCal-3054d4d62f832d2509536c57dc70d748
109 VARIABLES abort, queue, nodes, work_done, pc

111  define statement
112 TypeInvariant ≜ queue ∈ Nat

114 Statuses ≜ {nodes[n] : n ∈ 1 .. Nodes}
115 AllBuiltOrCached ≜ Statuses ⊆ {"built", "cached"}
116 SomeErrored ≜ "errored" ∈ Statuses

118 EventuallyQueueIsConsumed ≜ ◇□(abort ∨ (¬abort ∧ queue = 0))
119 NoWorkIsDoneTwice ≜ ◇□(abort ∨ (¬abort ∧ work_done = 1 .. Nodes))
120 EitherWeAbortOrThereAreNoErrors ≜ ◇□( (abort ∧ SomeErrored) ∨ (¬abort ∧ AllBuiltOrCached))

122 VARIABLE current_job

124 vars ≜ ⟨abort, queue, nodes, work_done, pc, current_job⟩

126 ProcSet ≜ (1 .. Workers)

128 Init ≜  Global variables
129         ∧ abort = FALSE
130         ∧ queue = Nodes
131         ∧ nodes = [n ∈ 1 .. Nodes ↦ "waiting"]
132         ∧ work_done = {}
133          Process Worker
134         ∧ current_job = [self ∈ 1 .. Workers ↦ −1]
135         ∧ pc = [self ∈ ProcSet ↦ "Loop"]

137 Loop(self) ≜ ∧ pc[self] = "Loop"
138                ∧ IF abort ∨ queue = 0
139                     THEN ∧ pc′ = [pc EXCEPT ![self] = "Done"]
140                          ∧ UNCHANGED ⟨queue, nodes, current_job⟩
141                     ELSE ∧ current_job′ = [current_job EXCEPT ![self] = queue]
142                          ∧ nodes′ = [nodes EXCEPT ![current_job′[self]] = "building"]
143                          ∧ queue′ = queue − 1
144                          ∧ pc′ = [pc EXCEPT ![self] = "Work"]
145                ∧ UNCHANGED ⟨abort, work_done⟩

147 Work(self) ≜ ∧ pc[self] = "Work"
148               ∧ current_job[self] ≠ −1
149               ∧ ∨ ∧ pc′ = [pc EXCEPT ![self] = "CheckOnDependencies"]
150                 ∨ ∧ pc′ = [pc EXCEPT ![self] = "BuildNode"]
```

```
151                 ∧ UNCHANGED ⟨abort, queue, nodes, work_done, current_job⟩

153 CheckOnDependencies(self) ≜ ∧ pc[self] = "CheckOnDependencies"
154                             ∧ IF current_job[self] < Nodes ∧ (nodes[current_job[self] + 1] = "built" ∨ nod
155                                   THEN ∧ pc' = [pc EXCEPT ![self] = "BuildNode"]
156                                   ELSE ∧ pc' = [pc EXCEPT ![self] = "WaitForDependency"]
157                             ∧ UNCHANGED ⟨abort, queue, nodes, work_done,
158                                          current_job⟩

160 WaitForDependency(self) ≜ ∧ pc[self] = "WaitForDependency"
161                           ∧ IF ¬abort ∧ current_job[self] < Nodes ∧ nodes[current_job[self] + 1] = "buildi
162                                THEN ∧ TRUE
163                                     ∧ pc' = [pc EXCEPT ![self] = "WaitForDependency"]
164                                ELSE ∧ IF ¬abort
165                                          THEN ∧ pc' = [pc EXCEPT ![self] = "BuildNode"]
166                                          ELSE ∧ pc' = [pc EXCEPT ![self] = "Done"]
167                           ∧ UNCHANGED ⟨abort, queue, nodes, work_done,
168                                        current_job⟩

170 BuildNode(self) ≜ ∧ pc[self] = "BuildNode"
171                   ∧ ∨ ∧ pc' = [pc EXCEPT ![self] = "WorkSucceeds"]
172                     ∨ ∧ pc' = [pc EXCEPT ![self] = "CacheHit"]
173                     ∨ ∧ pc' = [pc EXCEPT ![self] = "BuildError"]
174                   ∧ UNCHANGED ⟨abort, queue, nodes, work_done, current_job⟩

176 WorkSucceeds(self) ≜ ∧ pc[self] = "WorkSucceeds"
177                      ∧ nodes' = [nodes EXCEPT ![current_job[self]] = "built"]
178                      ∧ work_done' = (work_done ∪ {current_job[self]})
179                      ∧ pc' = [pc EXCEPT ![self] = "Loop"]
180                      ∧ UNCHANGED ⟨abort, queue, current_job⟩

182 CacheHit(self) ≜ ∧ pc[self] = "CacheHit"
183                  ∧ nodes' = [nodes EXCEPT ![current_job[self]] = "cached"]
184                  ∧ work_done' = (work_done ∪ {current_job[self]})
185                  ∧ pc' = [pc EXCEPT ![self] = "Loop"]
186                  ∧ UNCHANGED ⟨abort, queue, current_job⟩

188 BuildError(self) ≜ ∧ pc[self] = "BuildError"
189                    ∧ nodes' = [nodes EXCEPT ![current_job[self]] = "errored"]
190                    ∧ abort' = TRUE
191                    ∧ pc' = [pc EXCEPT ![self] = "Loop"]
192                    ∧ UNCHANGED ⟨queue, work_done, current_job⟩

194 Worker(self) ≜ Loop(self) ∨ Work(self) ∨ CheckOnDependencies(self)
195               ∨ WaitForDependency(self) ∨ BuildNode(self)
196               ∨ WorkSucceeds(self) ∨ CacheHit(self)
197               ∨ BuildError(self)
```

199 Allow infinite stuttering to prevent deadlock on termination.

200 $Terminating \triangleq \land \forall self \in ProcSet : pc[self] = \text{“Done”}$

201 $\land$ UNCHANGED $vars$

203 $Next \triangleq (\exists self \in 1 \,..\, Workers : Worker(self))$

204 $\lor Terminating$

206 $Spec \triangleq \land Init \land \Box[Next]_{vars}$

207 $\land \forall self \in 1 \,..\, Workers : \mathrm{WF}_{vars}(Worker(self))$

209 $Termination \triangleq \Diamond(\forall self \in ProcSet : pc[self] = \text{“Done”})$

211 END *TRANSLATION* − the hash of the generated TLA code (remove to silence divergence warnings): TLA-c70b83f43df290b

213

* Modification History
* Last modified *Wed Sep* 09 19:32:14 *CEST* 2020 by *ostera*
* Created *Wed Sep* 09 12:07:17 *CEST* 2020 by *ostera*