Proof of Completeness

Quentin Nivon and Gwen Salaün

Univ. Grenoble Alpes, CNRS, Grenoble INP, Inria, LIG, F-38000 Grenoble France

It was stated in Proposition 1 that the patterns applied to an abstract graph G and a task T are complete, i.e., there does not exist any abstract graph G' in which T has been moved and which has not been generated by applying the patterns. It is worth recalling that this approach applies a succession of steps in sequence, during which exactly one task is moved from one position to another, while the other tasks remain at their original position (i.e., no dependency between tasks is generated/removed except for the task being moved). With this assumption, let us prove the completeness of the proposed patterns.

Proof (Proof of Completeness). Let $G=(S_N,S_E)$ be an abstract graph, $n_i=(S_T,S_G)$ any node of S_N, n_f the first node of G, n_l the last node of G, and T the task to move. According to the definition of abstract graph, the following operations can be applied to G in order to insert T in it without adding/removing eventual dependencies between other tasks of the process:

- T can be inserted in S_T as a task of n_i. Semantically, this means that T will be put in parallel of all the remaining tasks of S_T, and all the sub-graphs of S_G. By definition of the semantics of the abstract graphs, it is equivalent to putting task T inside the set of tasks of a new abstract node having for sub-graph n. More formally, let $n_{new}=(\{T\},\{n_i,\emptyset)\}$, we have that $(S_T\cup\{T\},S_G)\Longleftrightarrow$

  $$\begin{cases}(S_N\setminus\{n_i\}\cup\{n_{new}\},S_E) \text{ if } n_i=n_f \wedge n_i=n_l\\ (S_N\setminus\{n_i\}\cup\{n_{new}\},S_E\setminus\{n_i\rightarrow n_{i+1}\}\cup\{n_{new}\rightarrow n_{i+1}\}) \text{ if } n_i=n_f \wedge n_i\neq n_l\\ (S_N\setminus\{n_i\}\cup\{n_{new}\},S_E\setminus\{n_{i-1}\rightarrow n_i\}\cup\{n_{i-1}\rightarrow n_{new}\}) \text{ if } n_i\neq n_f \wedge n_i=n_l\\ (S_N\setminus\{n_i\}\cup\{n_{new}\},S_E\setminus\{n_{i-1}\rightarrow n_i,n_i\rightarrow n_{i+1}\}\cup\{n_{i-1}\rightarrow n_{new},n_{new}\rightarrow n_{i+1}\}) \text{ o.}\end{cases}$$

  which is actually the definition of Pattern 2 (i) in the case where the combination of nodes used contains only one node. Such an insertion can also be done for any sub-graph of n, which is covered by the recursive call of Pattern 2 (ii). Thus we have proved that $\forall n\in S_N$, inserting T in the set of a tasks of n is semantically equivalent to applying (a fraction of) Pattern 2.

- T can be put in a new node $n_{new}=(\{T\},\emptyset)$ which is put side by side with G as sub-graphs of a new abstract node. By definition of the semantics of the abstract graphs, it is equivalent to putting G inside the set of sub-graphs of n_{new}. More formally, we have that $(\emptyset,\{G,n_{new}\})\Longleftrightarrow(\{T\},\{G\})$, which is actually the definition of Pattern 2 (i) in the case where the combination of nodes used contains all the nodes of G. Such an insertion can also be done for any sub-graph

of n, which is covered by the recursive call of Pattern 2 (ii). Thus we have proved that putting n_{new} and G as sub-graphs of an empty node is semantically equivalent to applying (a fraction of) Pattern 2.

- T can be put in a new node $n_{new} = (\{T\}, \emptyset)$ which is in parallel of an ordered combination of nodes of G. This is directly Pattern 2 (i) when the combination of nodes used contains more than one node and less than $|S_N|$ nodes. It can be put similarly in all the sub-graphs of the nodes of G, which is covered by Pattern 2 (ii).

- T can be put in a new node $n_{new} = (\{T\}, \emptyset)$ which is in sequence of any node of G. This is directly Pattern 1 (i), (ii) and (iii). It can be put similarly in all the sub-graphs of the nodes of G, which is covered by Pattern 1 (vi).

- T can be put before or after any task T' of S_T. To do so, we have to remove T' from S_T, and to put both T and T' in two empty abstract nodes connected to each other. This fresh abstract graph then has to be put in S_G. This is directly Pattern 2 (iv) and (v). It can be put similarly in all the sub-graphs of the nodes of G, which is covered by Pattern 1 (vi).

- T can be put before or after any combination of tasks (T_1, ..., T_m) and sub-graphs (G_1, ..., G_n) of n_i. To do so, each task (T_1, ..., T_m) must be removed from S_T and put in a new abstract node, along with the sub-graphs (G_1, ..., G_n). Then, T is put in a new abstract node which is connected to the previously generated abstract node. This abstract graph is finally added to S_G. This is directly Pattern 3 (i) and (ii). It can be put similarly in all the sub-graphs of the nodes of G, which is covered by Pattern 3 (iii).

Other operations such as putting T in parallel of any combination of nodes, or in parallel of tasks and partial sub-graphs are necessarily adding or removing dependencies between the other tasks of the process, which we do not want. Consequently, this proof shows that all the possible positions of T can be generated by (or have an equivalence with) the proposed patterns.

□