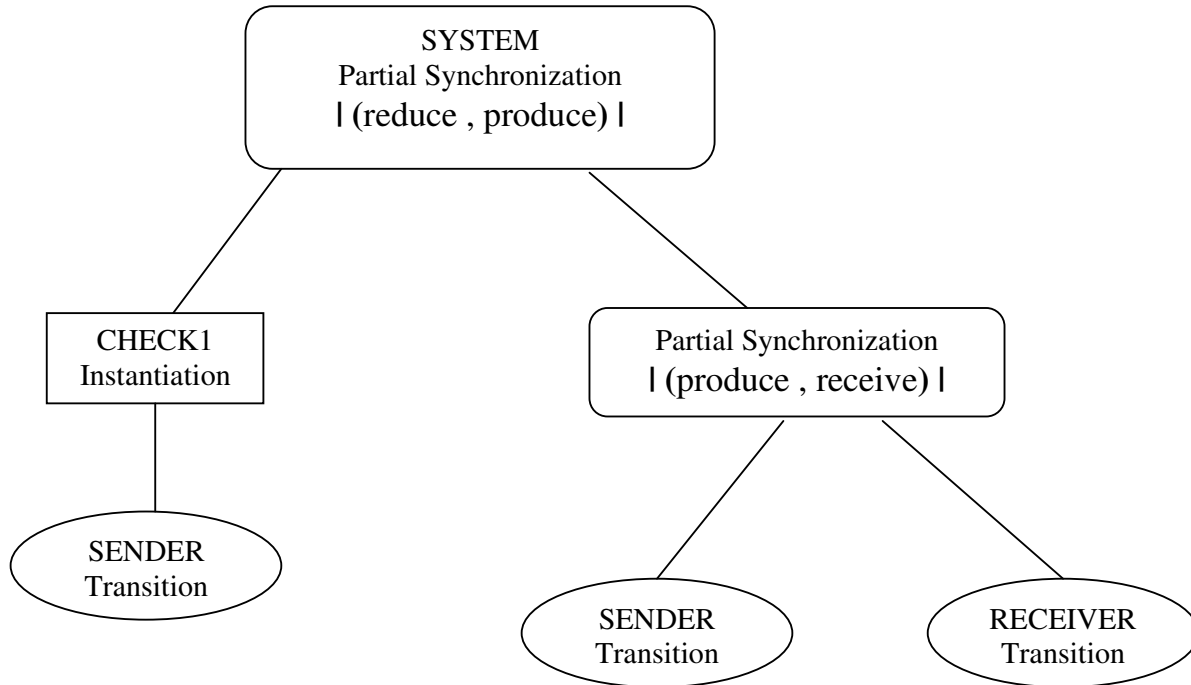


# Core2Step Translation

The example used in this description is following:



Sender contains transition produce.

Check1 renames transition produce -> reduce in module Sender below it.

Receiver contains transition receive.

The full code of the core program that this tree describes is in file named "Combination2" in the folder Combination2 under the examples directory.

Description of the tree data structure above:

The tree contains pointer to the root node.

Each node in the tree contains the following fields:

- `modulenode_type` Type – one of five possible types of modules : Transition, Instantiation, Full/Partial Synchronous and Asynchronous
- `modulenode_type` `Prev_type` – type of node before the handling of synchronization
- `ModuleNode*` `Left` – the node on the left
- `ModuleNode*` `Right` – the node on the right
  
- `Module*` `Module` - actual module associated with the node
- `PairSet*` `Sync` - the set of pairs of names of transitions that are synchronized (in case of partial or full synchronization)
- `ActualParameterSet*` `Params` - parameters of the module instantiation (if one contained in node)
- `TransitionSet*` `ChangedTrans` - set of transitions that originally were standalone in module and now are synchronized
- `TransitionSet*` `SyncTrans` - set of transitions that are synchronized in the module
  
- `PairSet*` `ChangedNames` - Pair set that holds the name changes following the synchronization handling  
Left part of the pair is the synchronization pair that affected the change of name  
Right part of the pair is the new name given to each transition that appears in synchronization.
- `PairSet*` `CurrChanges` - Pair set that holds the name changes that are made during the handling of the current synchronous node. Used for the full synchronization and consequent pairs partial synchronization when changes are to be applied on the original names of the transitions from before beginning of handling the node.
- `RenameSet*` `Renames` - Rename set that hold the renames for the transitions in case the node contains module that is of transition type. In case the node is of the other type, contains the renames for the pairs that are synchronized to allow additional synchronization on the upper level.

Description of the process of the translation:

- 1) Create tree as the one on the top of the description. Two Sender modules in the tree are different and unrelated.
- 2) Perform in-order transition through the tree and each time node of type Full or Partial Synchronous combination is met do:
  - If the node is of type Full Synchronization then collect all transition names from left and right sub-trees and create set of pairs of transition names (a, b) such that transition a is in one of the modules in the left sub-tree and transition b is in one of the modules in the right sub-tree of the node.
  - If the type of the node is Partial Synchronization then for each pair in synchronization collected transition names from left and right subtree that apps the pair in synchronization (more than one transition can be associated with single name, if the transition with this name was already synchronized with other transition on lower level of tree) and then cartesian product of left and right names is made.In the example :
  1. For partial synchronization between SENDER and RECEIVER transition from left subtree is “produce” and from right “receive”
  2. For partial synchronization in the root, transition from the left subtree is reduce and from right subtree transition “produce\_receive”

Use the sets of pairs of names created above in the following procedure:

For each pair in the set do:

- 1.1) Create new name for the joint transition from the names of the transitions in the pair.
- 1.2) For each transition name in the pair find the node that contains the module that has transition with this name in its transition set.
- 1.3) Create copy of each transition that was found in 2 and give it the name from 1. Save the change in the node->CurrChanges set (for case of full synchronization or partial synchronization of type |(a,b),(a,c)| where a appears in more than one pair.

After all pairs in set are handled, go through the set of node transitions in all nodes in the subtree of the current node and leave only those transitions that don't appear in ChangedTrans set (e.g. are not synchronized). Afterward, apply changes from CurrChanges on the set of synchronized transitions of the module and set ChangedNames.

- 3) After all synchronizations were handled, perform one more tree traversal, during

- which for each node with module inside transfer transitions from SyncTrans (synchronized transitions) to the transition set of the module.
- 4) If the root node of the tree is not Transition Module, then create new root module with updated dependencies according to changes made in 1 and 2.

## **Translation Notes:**

- 1) Handling of non-deterministically initialized variables in all modules is performed by using the following transformation to STeP:
- Go through the tree, and for each node that contains module do:
- 1.1) Collect all non-deterministically initialized variables into the set of variables.
  - 1.2) Add new boolean variable “hide\_var” with initial value “true” to the module.
  - 1.3) Update enable fields of all transitions to expression : “old\_enable  $\wedge$  !hide\_var”
  - 1.4) Add new transition to module. The transition will initialize all variables that were initialized non-deterministically in core by using relation field and update “hide\_var” variable to “false”.
- 2) Limitations of the current version:
- Non-deterministic initialization of global variables is not supported
  - Multiple transition synchronization is not supported : synchronization of type  $l(a,(b,c))l$ .
- To overcome the problem, the writer of the CDL program should use the following syntax:
- Instead of  $A\ l(a,(b,c))\ (B\ \parallel\ C)$   
 Create module D that instantiates  $B\ \parallel\ C$  and use the following syntax:
- $A\ l(a,b\_c)\ D[(b,c)\rightarrow b\_c]$