ParCor

Amin M. Alipour
malipour@mtu.edu

Department of Computer Science
Michigan Technological University

October 29, 2010
Outline

Goal

Initial Specification of the input language

Initial Specification of the input language

Initial Specification of the input language

ParCor Architecture

process.h

node.h

rgaux.h

symanal.h, Tool.cpp
Main Goal
Building a tool to

- enumerate the state space of a concurrent program (generation), and
- to prune the state space to conform a certain property (correction).

Requirements

- Building a specification language to specify the program.
- Using the input in specification language to generate state space
- Implementation of correction algorithm in the state space.
Initial Specification of the input language

\[
\begin{align*}
\text{processBody} & \quad ::= \quad (\text{statementBlock;}^+ \\
\text{statementBlock} & \quad ::= \quad (\text{statement;}^+ \\
\text{statement} & \quad ::= \quad (\text{atomicStatement} \mid \text{nonAtomicStatement})^+ \\
\text{atomicStatement} & \quad ::= \quad \langle\text{sequentialIf}^+\rangle\langle\text{assignment}^+\rangle\langle\text{parallelIf}^+\rangle \\
\text{nonAtomicStatement} & \quad ::= \quad \text{sequentialIf}^+\text{assignment}^+\text{parallelIf}^+ \\
\text{sequentialIf} & \quad ::= \quad \text{If} \quad (\text{condition}) \quad \text{Then} \quad \text{statement} \\
\text{parallelIf} & \quad ::= \quad \text{[If} \quad (\text{condition}) \quad \text{Then} \quad \text{statement} \\
\text{assignment} & \quad ::= \quad v := expr, \text{ where } v \in V_p \text{ and } expr \text{ is an expression in terms of variables in } V_p.
\end{align*}
\]
Implemented Specification of the input language

Example 1- Barrier Synchronization

boundary (0,2);
char pc[10]=0;
process 10 barrier
{
  while(1)
  {
    pc[_pid]=0;
    pc[_pid]=1;
    pc[_pid]=2;
  }
}
Implemented Specification of the input language

Example 2- Single Bridge

boundary(0,2);
char car[2]=0;
char capacity=2;
char occupied=0;
process 2 bridge
{
char choice=0;
while(1)
{ atomic{if(capacity>occupied)
{ car[_pid]=car[_pid]+1; occupied=occupied+1;}}
if(car[_pid]>0)
{ ::{ ::(1) then {choice=1;}
    ::(1) then {choice=0;}
}}
if(choice==1|| capacity==car[_pid])
    atomic{ car[_pid]=car[_pid]-1; occupied=occupied-1; choice=0;}}
ParCor Architecture

Program Specification → Parser

Simulation Driver (Tool.cpp) → process.h

Revision Library (rgeux.h) → g++

RG generator and corrector → Set of Minterms for each statement
$ ./parser prog.spec produces process.h which defines the data structure for the state and corresponding functions for state exploration.

**Data structure**
This data structure includes a PC for every thread, global data and a copy of local data for each local data. Example for the bridge example (reminder: it has two threads):

typedef struct {int pc[2];} programcontrol;
typedef struct {
    char choice_1[2];
    char occupied_0;
    char capacity_0;
    char car_0[2];} state;
typedef struct programstate {programcontrol *p;
    char res;
    state* varstate;
    struct programstate* next;
} programstate;
process.h- functions

process.h provides the following functions

- `programstate *init()`, which return an initial state.
- `string toString(state *oldstate)`, prints the program state.
- `int processnum()`, returns the process number.
- `programstate* process(int processno)` returns the next state upon execution of next statement in thread processno.
- `int equalp(programstate *pstate1,programstate *pstate2)` checks the equality of pstate1 and pstate1.
- `void restore(programstate *progstate)` load progstate in local variables.
- `programstate *takesnapshot(int n)` return snapshot of values after execution of one atomic step of thread n based on loaded values.
An snippet of programstate* process(int processno)

if(pc[processno]==1)
{
    if(capacity_0>occupied_0)
        nextpc= 2;
    else nextpc= -2;return takesnapshot(processno);
}

if(pc[processno]==2)
{
    car_0[processno]=car_0[processno]+1; occupied_0=occupied_0+1;
    nextpc= 3;return takesnapshot(processno);
}
if(pc[processno]==-2)
{
    nextpc= 3;return takesnapshot(processno);
}
node.h

rgaux.h simply defines the data structure required for reachability graph.

typedef struct node
{
    bool visited;
    bool deleted;
    bool safetyboundary;
    bool visited2;
    char rank;
    programstate* content;
    struct edge *in,*out;
    char indegree,outdegree;
    bool operator==(const node n1) const
    {
        return (equalp((*this).content,n1.content)&&equal((*this).content->varstate,n1.content->varstate));
    }
} node;
typedef struct edge
{
    struct edge *next;
    node *content;
    char process;
    char trans_id;
    bool isvalid;
} edge;

typedef struct nodelist
{
    struct nodelist* next;
    node *content;
}nodelist;
rgaux.h simply implements the data structure required for reachability graph and the corresponding functions.

```c
struct nlist_item
{node *content;
 struct nlist_item* next;};
struct nlist_item *node_first=NULL;
int totalnodes=0;
int totaledges=0;
int deleted_nodes=0;
int deleted_edges=0;
void remove(node *src, node *dest,int process);
void removeAll(node *src, node *dest);
void showdata(node* cur);
```
symanal.h, tool.cpp

symanal.h defines the safety and progress property. (This part has to be done manually.) Tool.cpp simply explores the RG and then apply the correction algorithms which described in the technical report.