1 Implementing PDR/IC3 for Linear Integer Arithmetic: Project Proposal.

- Akshay Naik, Devendra Reddy

1.1 Description of the PDR Algorithm

1.1.1 Propagating Phase

Note: Frame 0, $F_0 = I$, set of initial states.

```
> Propagate(F_k):   //Builds F_(k+1)
>   F_(k+1) = P
>   forall c in clauses(F_k):
>     if UNSAT(F_k && T && !c'): //(F_k && T => c')
>       F_(k+1) = F_(k+1) && c
>   if F_(k+1) = F_k:
>     Terminate "P is valid in system."
>   return
```

1.1.2 Blocking Phase

On adding the current frame, F_k to the trace, we have to compute and discharge its proof obligations. If SAT($F_k \wedge \mathrm{T} \wedge \neg\ P'$), then $\exists$ a cube $s \in F_k$, corresponding to the proof obligation $< s, k >$. All such cubes, s need to be blocked by calling the following procedure with $< s, k >$. Optionally, one may optimize the query by adding $\neg s$.

Block() proceeds in a depth first fashion by eliminating the obligation and 'replacing'(by checking) it with its own obligation on previous frames. It does this by first finding the latest frame, F_{j+1} in which s can be blocked directly, and blocks s in all frames ranging from F_1 to F_{j+1}. At this point we're still left with the proof obligation $< s, f >$. This is eliminated by generating $< t, f-1 >$ for all t in the pre-image of s.

Note: $t \in \mathrm{cubes}(\exists x'(F_{f-1} \wedge \neg s \wedge T \wedge \neg s'))$ `= cubes(Pre-image of s' in F_(f-1))`

```
>Block(<s, f>):
>  isBlocked = False
>  forall j from f to 0:

>    //RI check (Optimization 2)
>    if UNSAT(F_j && !s && T && s'): //(F_j && !s && T => !s')
>      //generalize(): Remove all literals from s, that aren't in unsat_core(?)
>      sg = generalize(s)
```

```
>      //Strengthen F_(j+1) and all preceeding frames(except F_0) with !sg.
>      //Since successful RI check on F_j implies, s is not 1-step reachable from F_j.
>      r = j+1 if j < k else j
>      //Above check to avoid adding !sg to F_(k+1 ) when it's not been constructed yet.
>      //this maintains clausal inclusion property.
>      forall i from r down to 1:
>        F_i = F_i && !sg
>      isBlocked = True
>      break
>
>  if not isBlocked:
>    Terminate "P does not hold in transition system."
>  if j == f: //<s,f> obligation has been eliminated.
>    return
>  if j < f:
>    forall t in cubes(Pre-image of s in F_(f-1)):
>      Block(<t, f-1>)
>    return
```

1.1.3 Final Algorithm

Note: $s \in \text{cubes}(\exists x'(F_k \wedge T \wedge \neg P'))$ `= cubes(Pre-image of !P' in F_k)`

```
> PDR(k):
>   if UNSAT(F_k && T && !P'): //(F_k && T => P'), Inductive check.
>     Propagate(F_k)
>     PDR(k+1)
>   else:
>     forall s in cubes(Pre-image of !P' in F_k):
>       Block(<s, k>)   //s is a 1-step CTI
>     PDR(k)
>
> PDR(0)
```

1.2 Worked Example

1.3 Implementation

We intend to implement the entire algorithm in **Python** and are heavily leaning towards using **Z3** for SMT queries. We picked Z3 because we have prior experience with it and it supports unsat core generation and quantifier elimination for linear integer arithmetic. We are yet to investigate if it can assist in the generalization of cubes.

Storing frames: As an array of lists of tracked Z3 SMT formulas; each new frame is initialized to P.

At the level of pseudo-code there seems to be no need to store the proof obligations directly as in the depth first execution they are implicitly stored on the call stack.

1.4 Project Deliverables