BIP Examples

Wang Qiang

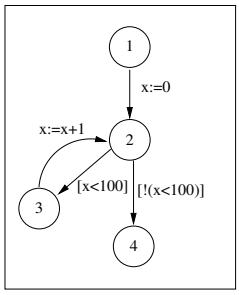
March 24, 2014

1 Abstract interpretation makes simple

Consider the following program:

```
1: x:=0;
2: while(x < 1000)
   {
3:         x:=x+1;
   }
4: assert (x==1000);
```

We will not analysis this version of source code directly, but build a control flow graph (CFG) or a program graph (called by Prof.Patrick Cousot), on which the analysis will be carried out. The control flow graph of this program is the following:



Intuitively, the essence of safety analysis of programs is to collect all information of possible program executions. In our case, the information is the valuation of program variable (e.g. x) in each program location (e.g. 2). So the naive way to find these valuations is to explicitly compute the values following the control flow graph. However, due to the presence of loops, this computation is usually too expensive and non-terminating. This approach can be generalized and formalized as a forward analysis.

We use $L = \{l_0, ..., l_n\}$ to denote the set of program control locations, where l_0 is set to be the initial location. We also define a map $\mu = L \mapsto FOL$, which assigns a first order formula to each program location. Intuitively, this formula characterises the information on the corresponding location (e.g. the set of valuations of variables). Program analysis is to find this map, which can be solved by the following procedure.

Procedure : Forward analysis
Input:
Output:

$\mu(l_0) = \top$; $\mu(l_i) = \bot \ \forall i \neq 0$
$WL := \{l_0\}$;
while $WL \neq \emptyset$ do:
  $l_i = pop(WL)$;
  foreach $l_j \in succ(l_i)$ do:
    (* $l_i \xrightarrow{e(i,j)} l_j$ *)
    $f = sp(\mu(l_i), e(i,j))$;
    if $f \not\Rightarrow \mu(l_j)$:
      $\mu(l_j) := \mu(l_j) \vee f$;
      $WL := push(WL, l_j)$;

If you change the image domain of map μ, we can obtain analysis of different precise. This is the basic idea of abstract interpretation. Instead of using formulae of first order logic, we can in practice use interval domain to obtain better efficiency, but with less preciseness. Note that we can view this analysis as a under-approximation approach, because we start from the bottom (i.e. $\mu(l_i) = \bot \ \forall i \neq 0$), and progressively to compute the invariants of each locations. Another formalization is also possible, in which we build a system of equations characterizing the effects of post operations (see Prof.Patrick Cousot's paper). In this case, we can obtain the same map by computing the fixed point of the equation system.

$\mu(l_0) = \top$
$\mu(l_1) = \bigvee_{e(k,1)} sp(\mu(l_k), e(k,1))$
...
$\mu(l_n) = \bigvee_{e(j,n)} sp(\mu(l_j), e(j,n))$

Notation $e(k,1), k \in [1,n]$ represents the action (i.e. guard or assignment) labelled on transition from l_k to l_1. The update of each location should take into account every possible predecessor.

This form of analysis tries to first collect as much information as possible, and then check if these information can prove or disprove our property. The drawback of this approach is that the computation is almost "blind" in the sense that we never know if the information being collected is useful for our goal. So in most cases we may waste our computation by collecting information irrelevant to our goal.

We can use this procedure to compute the approximate invariant of BIP atomic components by using different abstract domain (interval domain, polyhedra domain, and predicate domain).

2 Property guided analysis

In this section we will see another general form of safety analysis, which is from top and called "lazy" in the sense that it only collect the information in response to the failure of achieving the goal. So we can also say it is property guided analysis.

3 A proposal of model checking BIP

In brief, when system designers build a system model, it is necessary to check or prove their design respects the desired requirements, or simply make sure their design is free from deadlocks. The approach we take to achieve this goal is the algorithmic verification, like what DFinder does. However, besides this automation feature, there are some others that would be useful in practice from the perspective of engineers.

One of them is the capability of producing counterexamples. When the automatic verification detects a potential problem of the design, it should report a sufficient detailed "witness" of this problem. One kind of witnesses is a "bad" state that violating the requirements. This is what DFinder produces. However, this single state reveals very limited information of the design, so it is not sufficient detailed and not useful in practice. A good alternative is the execution trace violating the desired requirements. The trace reveals the exact executions of the system from an initial state to a "bad" state. So we require that our approach should produce such a trace when requirements are violated.

Another feature is controllability, which means in some cases we can control the process of verification. Due to the fundamental problem of state explosion, there is no general solution to systems of arbitrary size. So in case the size of our system grows, and the verification runs for several hours without any useful information, with current approaches we can either stops the running or wait for a loner time, in both cases we do not get what we want. The reason is that when we start the verification, it is out of control.

In general, there are two ways to abstract the atomic components:

1. Source-to-source abstraction by using Boolean abstraction. After abstraction, we get an abstract BIP model, on which we can do other transformations (e.g. BIP-to-NuSMV); This is described in my semester project. The data transfer is dealt with in an eager way by propagating predicates around components.

2. Abstract interpretation. In this way, we can generate the approximate invariant directly. The data transfer between different components can be dealt with lazily.

4 DFinder's story

4.1 Overview of DFinder approach

Input: $S =< \gamma(B_1, ..., B_n), Init >$
Output: S is deadlock-free or has a set of potential deadlocks.

1: Find an invariant Φ of S.
2: Compute DIS for $\gamma(B_1, ..., B_n)$.
3: If $\Phi \wedge DIS = false$ then return "S is deadlock-free" else go to 4 or 6.
4: Find Φ' an invariant of S.
5: $\Phi := \Phi \wedge \Phi'$ go to 3.
6: return the get of the solutions that satisfy $\Phi \wedge DIS$.

Several problems restrict the application of DFinder in practice, despite the data transfer problem. The way DFinder tries to prove a property (an invariant)

is to compute an approximate invariant of the system (in a compositional manner), and then prove the property is implied by the computed invariant. The methodology is eager in the sense that it tries to obtain as much information as possible regardless the properties to be checked, and does not know if the information obtained is enough or not to prove the desired property. A better strategy is to have an approach which only obtains the information necessary for proving the property.

5 Leader election algorithm

For $n \in N$, n processes $P_1, ..., P_n$ are located in a ring topology where each process is connected by an unidirectional channel to its neighbour is a clockwise manner. To distinguish the processes, each process is assigned a unique identifier $id \in 1, ..., n$. The aim is to elect the process with the highest identifier as the leader within the ring.

Let's take n=3 for example. The models for each process and its channel interface are given below. The property we want to verify is that only one process can stay at location $S4$ after each round.

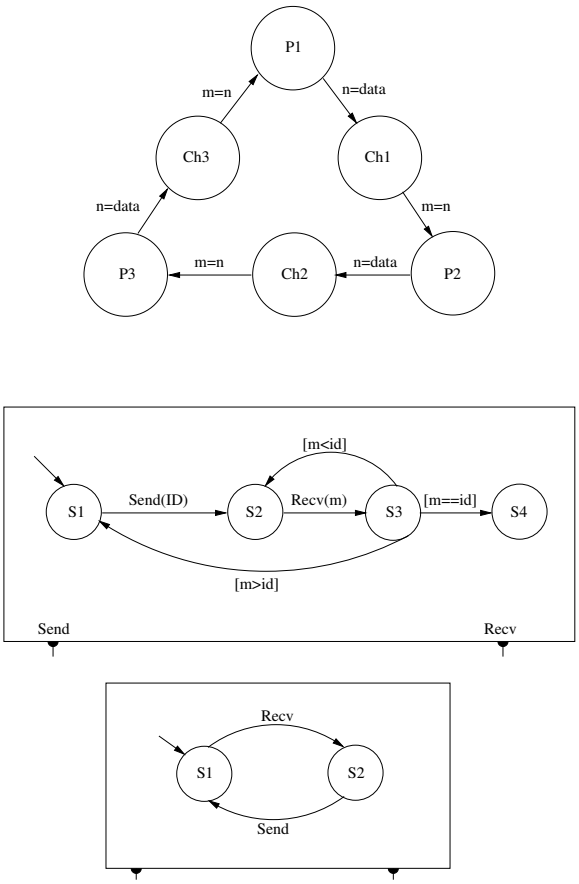


Figure 1: process and its channel interface

6 Quorum protocol

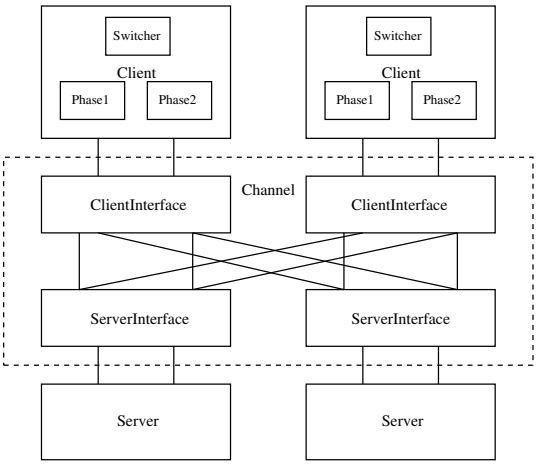
• Client c broadcasts its proposal v to all servers and waits for accept messages.

• When a server receives a proposal v from a client c, if it has not accept any message, it sends an accept message *accept(v)* to c; otherwise it sends *accept(v')* to c.

• If a client receives two different accept messages, it switches to Backup phase (Paxos) with his own proposal.

• If a client receives the same accept messages *accept(v)* from all the servers, then it decides v.

• If timer t_c expires, client then switches to Backup phase with value v' if he has such any response from servers.



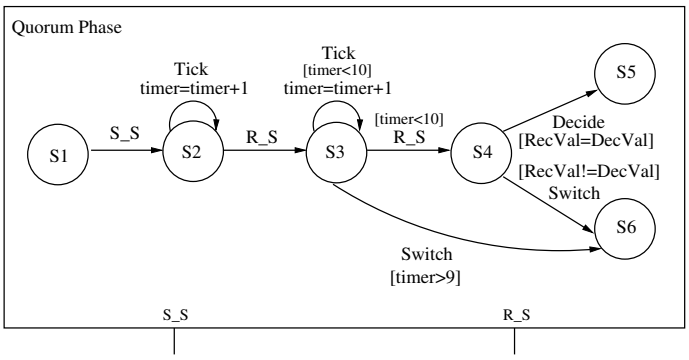


Figure 2: Client model

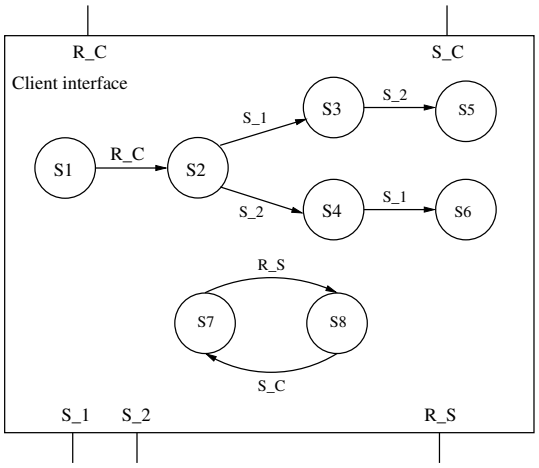


Figure 3: Client interface model

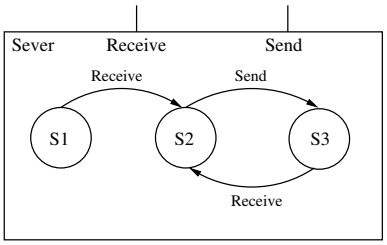


Figure 4: Server model

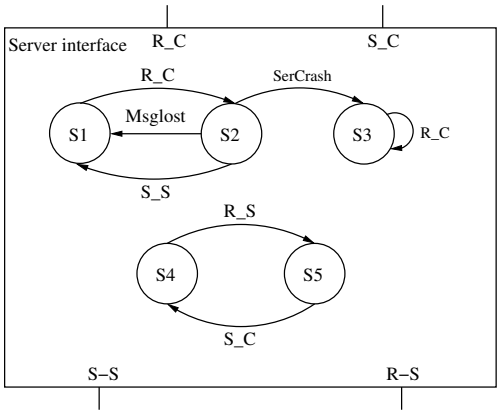


Figure 5: Server interface model

The properties we want to check are followings:

I1 If some client decides then all clients that switch, either before or after, do so with the same value.

I2 If some client decides then all clients that decide do so with the same value.

7 Dining cryptographers

8 Temperature control system

This system controls the coolant temperature in a reactor tank by moving two independent control rods . The goal is to maintain the coolant between the minimal temperature Θ_m and maximal temperature Θ_M. When the temperature reaches its maximal value Θ_M, the tank must be refrigerated with one of the two rods. The temperature rises at a rate υ_r and decreases at rate υ_d at the corresponding places. A rod can be moved again only if T time units have elapsed since the end of its previous movement. If the temperature of the coolant cannot decrease because this is no available rod, a complete shutdown is required.

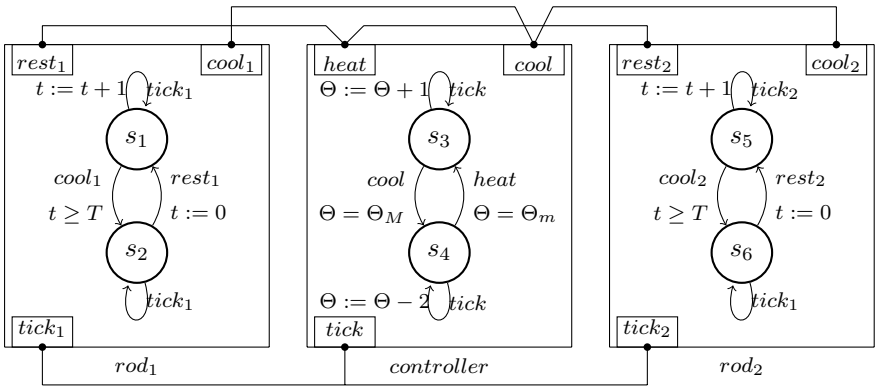


Figure 6: Temperature Control System

9 Bakery algorithm

We consider 2-process Bakery protocol whose BIP model is depicted in the following. Two components B_1 and B_2 model two processes and they are identical up to the renaming of ports and locations. B_1 has three locations: s_1 if it is in idle section; s_2 if it is waiting to enter its critical section; and s_3 if it is in its critical section. It has a variable x_1 ranging over the natural numbers and representing the "tokens" of the process, 3 loop transitions labeled by a allowing accessing the value of "token", three transitions w, e, r for moving between the locations. Initially in location l_1, B_1 can take w_1 to move to the waiting location s_2. From s_2, B_1 can enter the critical location s_3 by e_1 transition. The return to s_1 by transition r_1 resets the "token" x_1 to 0.

Two processes B_1, B_2 communicate with each other by a set of interactions $\gamma = \{a_1w_2, a_1e_2, a_2w_1, a_2e_1, r_1, r_2\}$ with the corresponding guards and functions illustrated in the model. An important property for Bakery system is mutual exclusion: both components can not be at the critical locations at the same time, i.e. $P = \neg(s_3 \vee s_6)$.

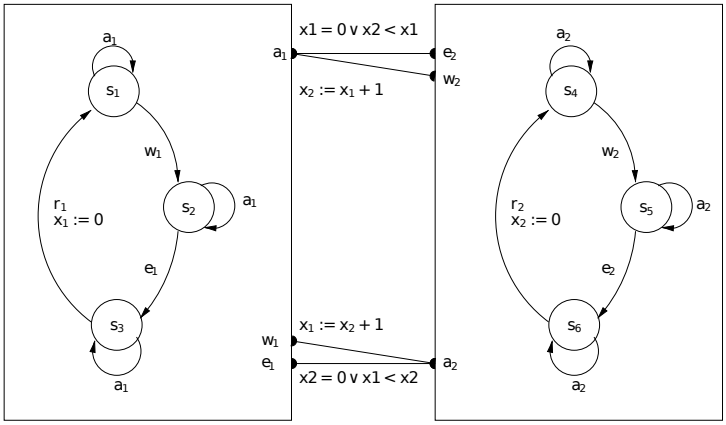


Figure 7: Bakery protocol

There is a problem of abstracting interactions. In this example for instance, the interaction a_1e_2 is guided by condition $x_1 = 0 \vee x_2 < x_1$, and the atomic predicate $x_2 < x_1$ refers to variables in both components, which prevents us from applying abstraction in a component-wise manner.

10 Paxos algorithm

The Paxos algorithm operates in the following two phases:

Phase 1.

a A proposer selects a proposal number n and sends a prepare request with number n to a majority of acceptors;

b If an acceptor receives a prepare request with number n greater than that of any prepare request to which it has already responded, then it responds to the request with a promise not to accept any more proposals numbered less than n and with the highest-numbered proposal (if any) that it has accepted.

Phase 2.

a If the proposer receives a response to its prepare requests (numbered n) from a majority of acceptors, then it sends an accept request to each of those acceptors for a proposal numbered n with a value v, where v is the value of the highest-numbered proposal among the responses, or is any value if the responses reported no proposals.

b If an acceptor receives an accept request for a proposal numbered n, it accepts the proposal unless it has already responded to a prepare request having a number greater than n.

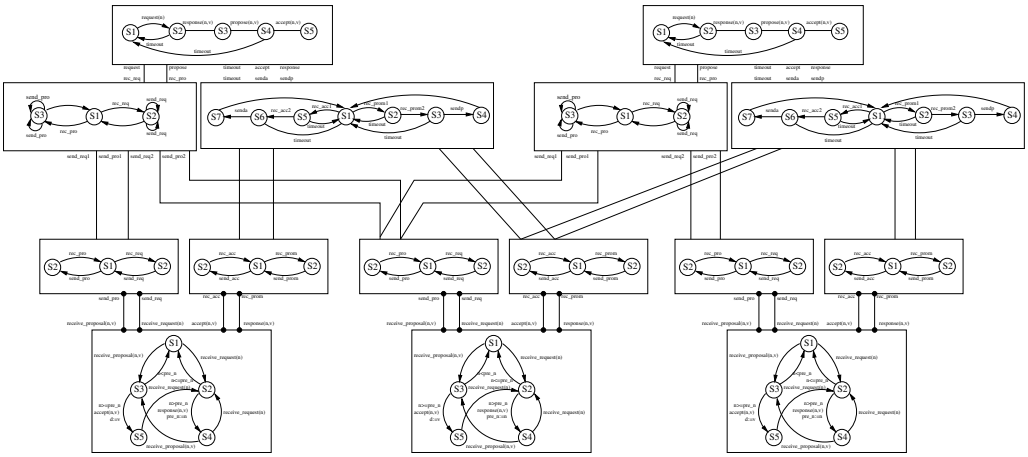


Figure 8: Paxos

11 Simple Voting Protocol

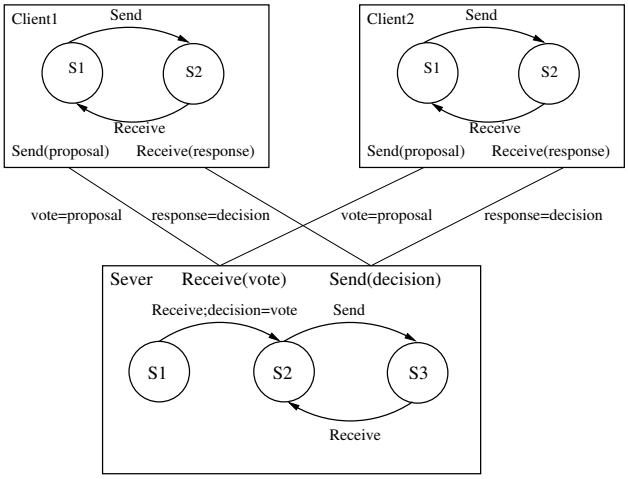


Figure 9: Simple voting protocol

In general, if the domain of proposal is the set of integers, then this system can not be verified directly using model checking techniques. Assuming we want to verify the property that if client one receives value 1, then client two will definitely receive the same value. Then we will use predicates $p1 \triangleq (response1 == 1)$ and $p2 \triangleq (response2 == 1)$ to abstract our system,

where $response1, response2$ correspond to the response in client 1 and client 2 respectively. Then the abstract client model becomes the following:

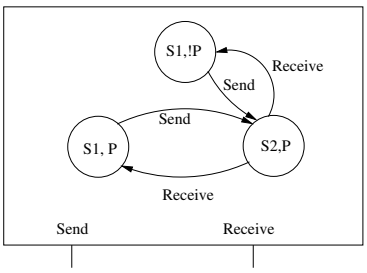


Figure 10: Abstract Client Model

Note that after abstraction, the original variable *proposal* and *response* are removed from the component, and thus the original data transfer is removed. For this reason, proper abstraction of server model is needed, and the predicates should be inferred by taking into account both the abstraction of other models and the data transfer function. For this example, predicate $p3 \triangleq (decision == 1)$ can be easily inferred.

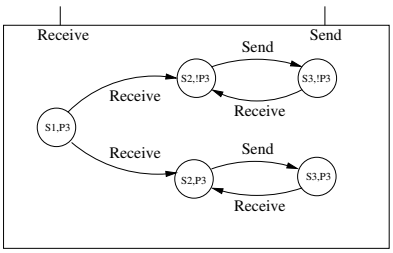


Figure 11: Server Client Model

Then we get an abstract system as following:

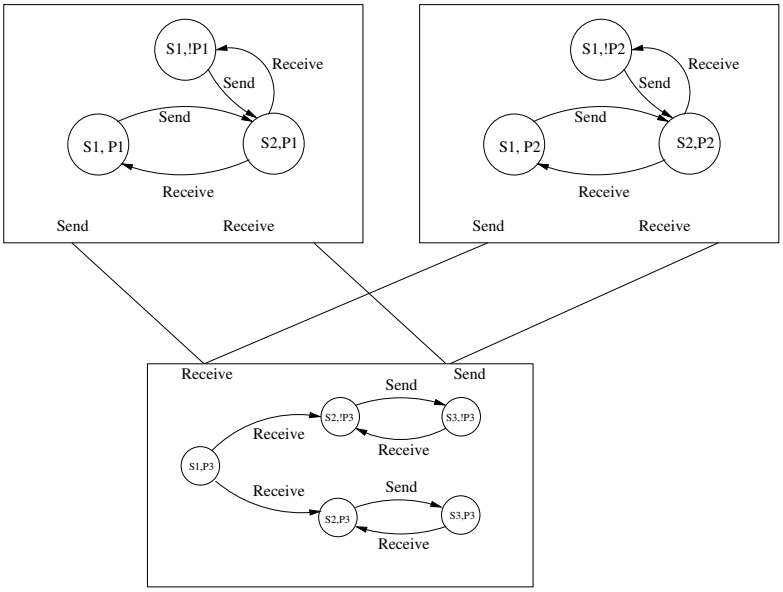


Figure 12: Abstract System Model

Note that in this abstract model, we remove the original data transfer. And we get a system which simulates the original one. However, this abstraction is too coarse in the sense that it introduces too many spurious behaviours. And the property we care is not satisfied in this abstract model. For instance the trace $((s_1, p_1), (s_1, p_2), (s_1, p_3)) - ((s_2, p_1), (s_1, p_2), (s_2, !p_3)) - ((s_1, !p_1), (s_1, p_2), (s_3, !p_3)) - ((s_1, !p_1), (s_2, p_2), (s_2, !p_3)) - ((s_1, !p_1), (s_1, p_2), (s_3, !p_3))$. The reason of having such a spurious trace is the absence of data transfer. And it can be resolved by introducing the following data transfer $p_1 = p_3$ or $p_2 = p_3$ in the connector. Then this property can be verified.

We explain the basic idea of abstracting BIP models in a component-wise manner. However, there are two important techniques which should be pointed out: the technique of inferring predicates and the technique of abstracting data transfer.

Another question is how to decompose a predicate involving variables in different components into several predicates which only contain variables in the same component. This is required in two phases: abstracting the data transfer and inferring predicates from properties.

Suppose we have a BIP model B which is built by composing several atomic components $\{B_i\}_{i \in I}$ with interaction model Γ, in order to get an approximation of B, theoretically we have two possible ways: we build the global system model first and then apply the abstraction; or we abstract each atomic component (also interaction model) first and then build the composition. However, in practise the first way is impossible to implement since the global system is infinite or very complicate to build. Naturally we would like to do the abstraction following the second way.

Definition (Minimal abstraction) Defined as the abstraction in which composition is done first.

Definition (Local abstraction) Defined as the abstraction in which abstraction is done first.

Prove that under some conditions they coincide with each other.

Suppose we already designed an abstraction for each atomic component B_i, which is characterized by a Galois connection (α_i, γ_i), then the abstraction for the composite component B is defined pointwise as $(\alpha_1 \times \alpha_2 ... \times \alpha_n, \gamma_1 \times \alpha_2 ... \times \gamma_n)$.