C Formal Verification with Unix Communication and Concurrency

D. N. Hoover
Odyssey Research Associates, Ithaca NY

Abstract

This talk reports the results of a NASA SBIR project in which we developed CSP-Ariel, a verification system for C programs which use Unix system calls for concurrent programming, interprocess communication, and file input and output. This project builds on ORA's Ariel C verification system by using the system of Hoare's book *Communicating Sequential Processes* to model concurrency and communication. The system runs in ORA's Clio theorem proving environment. We outline how we use CSP to model Unix concurrency, and sketch the CSP semantics of a simple concurrent program. We discuss plans for further development of CSP-Ariel.

C Formal Verification with
Unix communication and concurrency
(NASA SBIR)

Aim: Verification system for

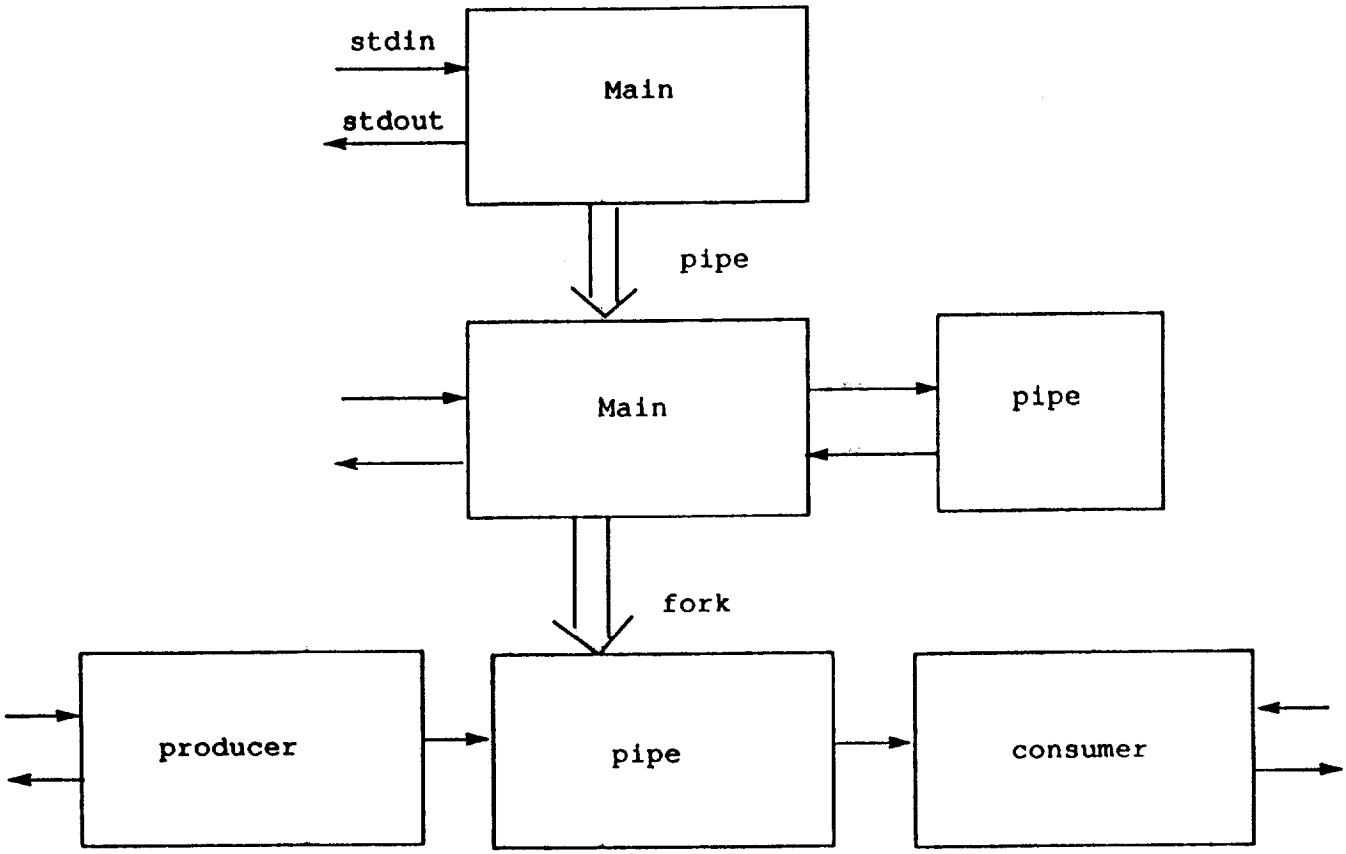
- C programs
- Unix system calls
- concurrent programming (`fork`, `wait`, `exit`, `pipe`)
- file and device i/o (`read`, `write`, `open`, `close`).

A.

Example program.

```
void producer();
void consumer();
int pipedes[2];

void main()
{
 int id;

 if (pipe(pipedes) == -1) return;

 id = fork();
 if (id == -1) return;
 if (id == 0) consumer();
 else producer();

 return;
}

void producer()
{
 char c;
 int status;

 while (read(0, &c, 1) != 0) /* 0 = standard input filedes */
       write(pipedes[1], &c, 1);
 close(pipedes[1]);
 exit(wait(&status));
}

void consumer()
{
 char c;

 close(pipedes[1]); /* so that pipe read will fail when producer
                       closes its write end of pipe */
 while ( read(pipedes[0], &c, 1) != 0)
       write(1, &c, 1);  /* 1 = standard output filedes */
 exit(0);
}
```

Example Program Schematic



Technical Approach

- C semantics via Ariel operational semantics (pre-existing)
- Unix communication and concurrency semantics via Hoare's CSP

CSP (Communicating Sequential Processes)

- See Hoare's book, *Communicating Sequential Processes.*
- An algebraic language for describing systems of processes with synchronous communication.
- Objects of the language are *processes* and *events.*
- Processes resemble state machines, events the input alphabet. Deterministic and nondeterministic processes.
- Processes participate in events and are transformed by them.
- Synchronous communication by participation in shared events.

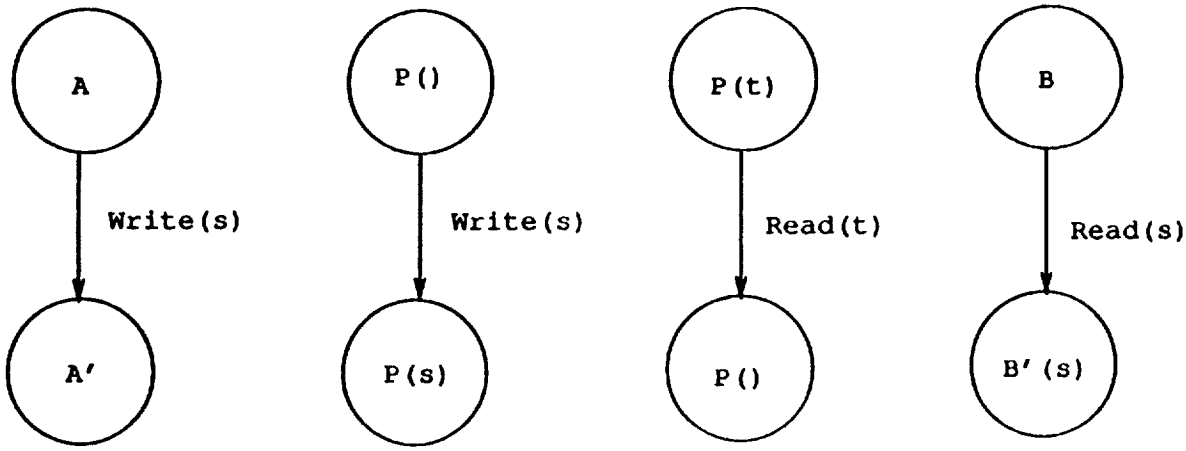
Unix modeling

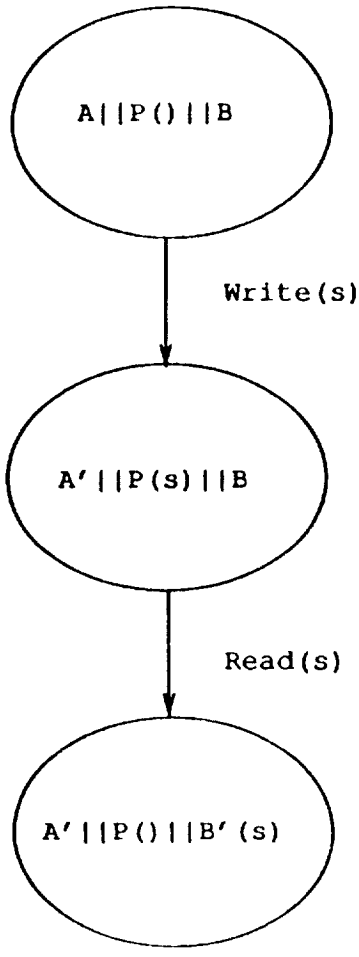
- Unix processes, files, pipes, and certain system tables are modeled as deterministic CSP-processes.

- Forking, pipe creation, file opening and closing, I/O, waiting, and exiting are modeled as events.

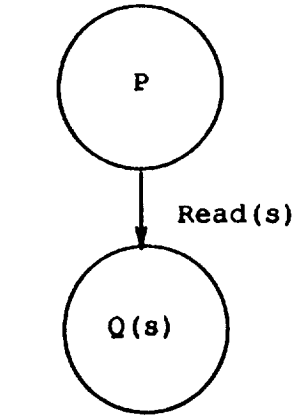
Example: Asynchonous pipe communication

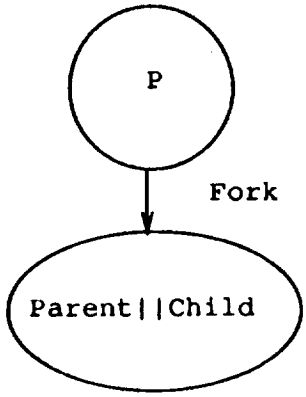
Sending process A, pipe P, receiving process B.

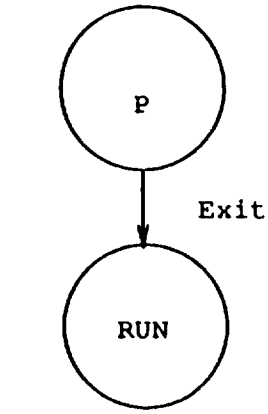




Processes transformed by events



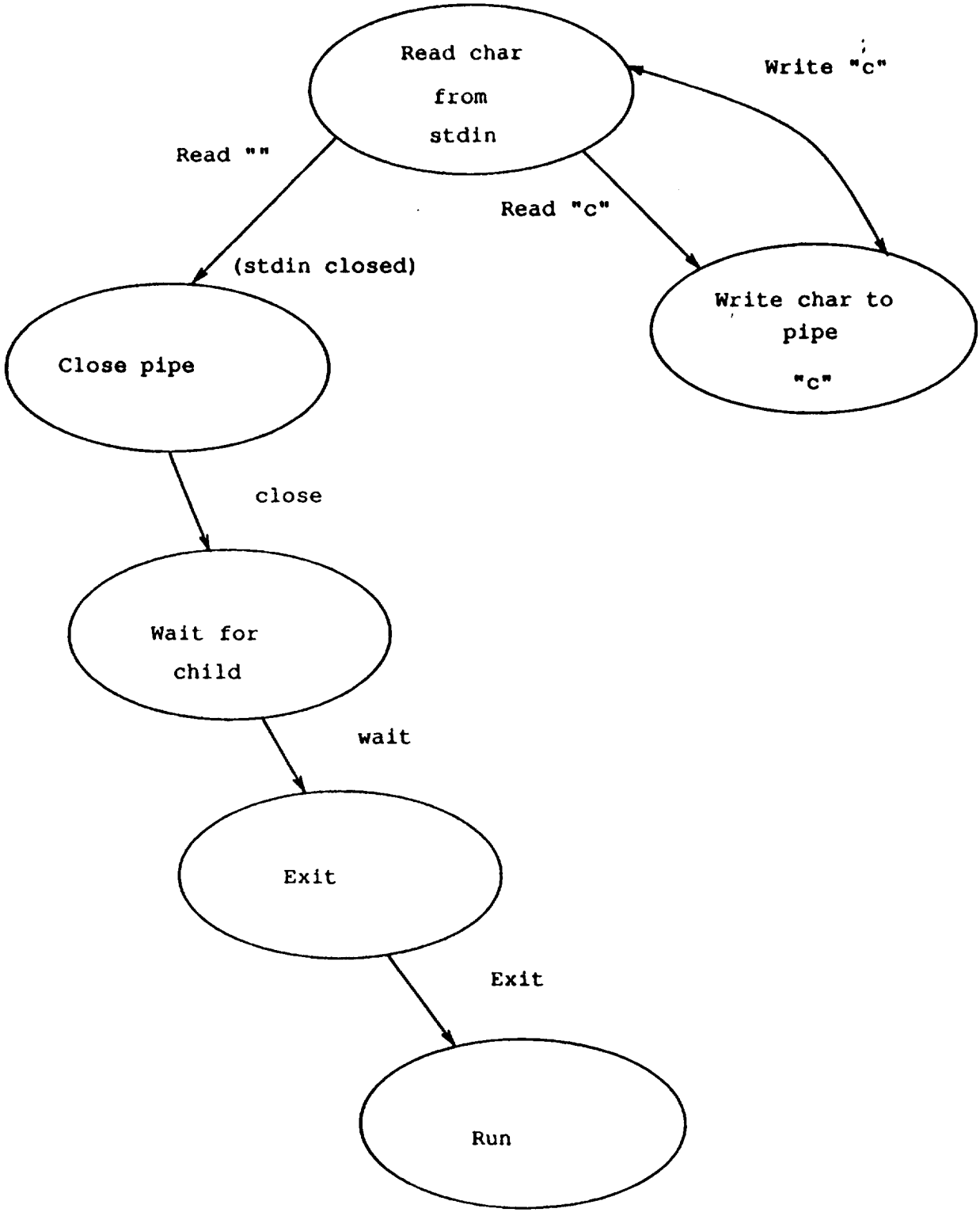




Verification method

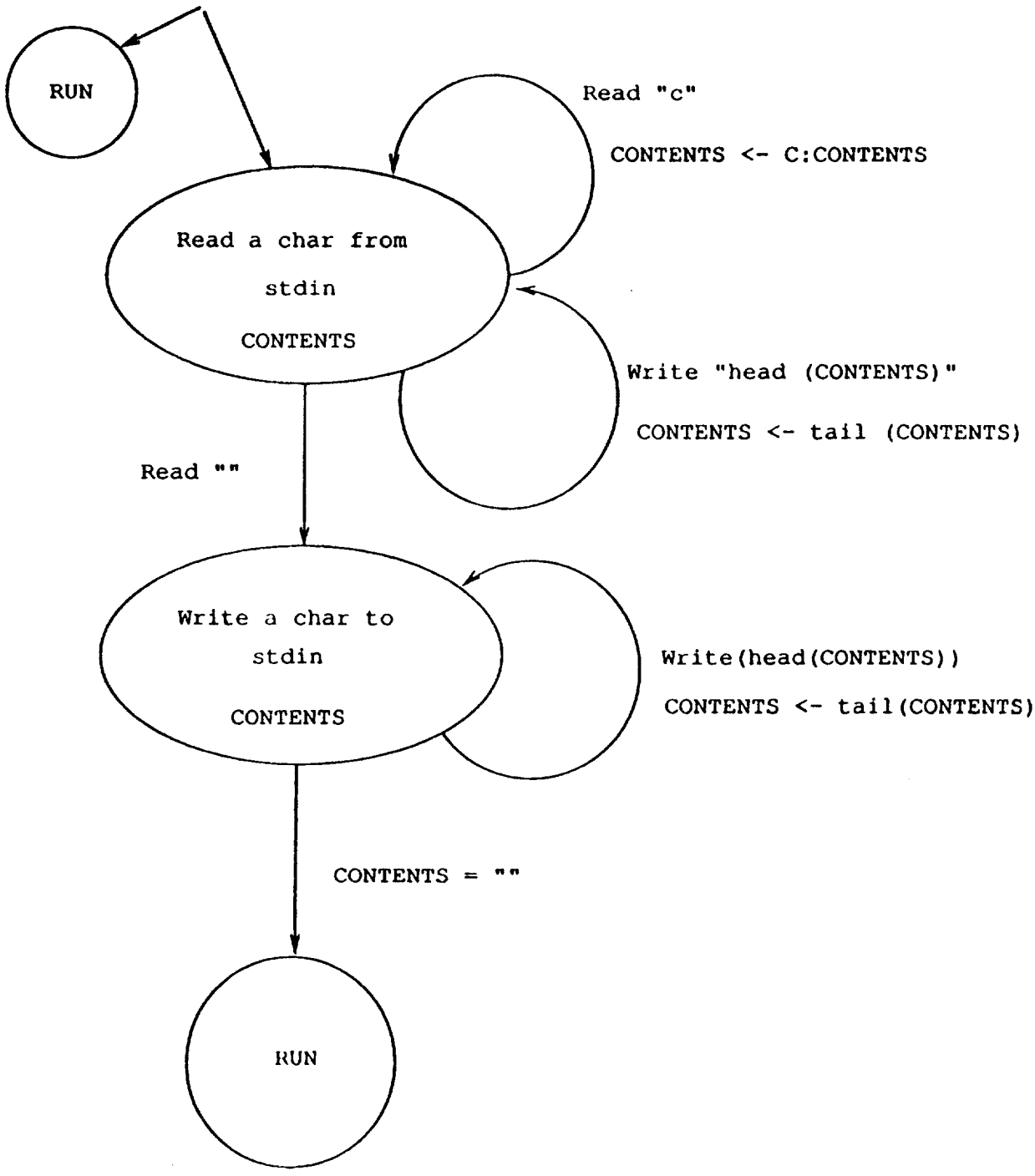
- C program given
- Ariel front end generates Caliban expression for abstract syntax tree of program.
- Ariel C semantics plus Unix system call semantics define denotation of a C program and associated files inside operating system as a CSP process.
- Internal operations of systems of processes hidden by CSP concealment operation.
- We reason about the resulting CSP process in Clio. Main tools are induction on traces (event sequences) of processes, and algebraic laws of CSP. Clio is a very general theorem prover, and we are not limited in the kinds of properties we can prove about processes.

Producer as a CSP process



Hiding events:

Overall process with non-I/O events hidden.



CSP-Ariel Development Plan

- C semantics via Ariel symbolic interpreter (existing)
- Unix communication and concurrency semantics via deterministic CSP (initial work completed).
- Extensions to support network communication planned (sockets).
- Nondeterministic CSP and event concealment for specification and modularity (planned)
- Graphic specification support using Romulus interface (planned)

Clio, Caliban, and, Ariel

- Ariel is a semantic verification system for a subset of C, written in Caliban and the Clio metalanguage. Floating point, overflow support via asymptotic correctness.

- Caliban is a lazy, purely functional language based on recursive equations and pattern matching.

- Clio is a higher-order logic theorem prover. Caliban is its term definition language. Clio's main proof methods are induction on Caliban definitions, term rewriting, and case splitting.