ICFP Programming Contest
2009
Problem Specification

University of Kansas

June 26, 2009

Version	Date	Description
1.0	June 26, 2009 13:00:16 CDT	Initial Release

1 Introduction

At 13:00:16, on June 29th, 1995, the shuttle docked with the Mir for the first time. In order to prepare for future international missions, you are to help program a satellite to clean up the skies of debris. There are three training missions, then *Operation Clear Skies*. Good luck!

1.1 Contest structure

Contestants will implement the virtual machine described in section 2. The contestants will be supplied a series of binaries, each of which simulates a particular satellite maneuver task, as described in section 6. These binaries represent a sequence of problems in increasing difficulty, where the user will be required to program a collection of satellites to perform a specific task. The motion of the satellites is dictated by the equations described in section 4.

The contestants control the simulation using actuators, reacting to the state of the simulation as exposed through sensors. Sensors and actuators interact with the supplied problem binary through a port-mapped I/O interface.

Contestants are responsible for using the sensor data provided by the simulator, and producing inputs to the actuators to control the behaviors of the various bodies. Contestant are free to use whatever language desired to develop this logic as long as it interacts with the virtual machine through the exposed input and output ports. Each solution for a given problem will be submitted as a trace of the solution's actuator inputs. A complete description of the solution submission procedure is described in section 3.

2 Orbit Virtual Machine Specification

The contest consists of a series of tasks involving maneuvering satellites around earth. A simulator for each problem has been compiled to a binary that can be executed on a simple virtual machine, called the Orbit virtual machine. The contestants are responsible for implementing the Orbit VM, according to the ISA specification found in this section. Executables are encoded in a binary format that the contestants must decode. Moreover, solutions to the problems must be submitted as execution traces, also encoded in a binary format. Both the executable format and the solution submission format are described below.

The execution model for the contest integrates the Orbit virtual machine with a controller program that interacts with the VM through a port-mapped interface. Typically, the controller will write to the VM's input ports, then start the VM execution. The VM will execute instructions, starting at address 0x0 and continuing on to the last address. After the machine has executed the entire address space, which corresponds to one simulation time step, the controller may then read the VM's output ports, generate new input port values, and then repeat the process.

2.1 Orbit Virtual Machine Specification

The Orbit virtual machine contains a data memory of 64-bit double-precision floating point values and a disjoint instruction memory consisting of 32-bit instructions. Both the data and the instruction memory are indexed by 14-bit addresses. Doubles are represented as 64-bit IEEE 764 double-precision floating point numbers (1 sign bit, 11 exponent bits, 52 significant bits).

The address of the current instruction is stored in a 14-bit program counter register. This register is incremented after the execution of each instruction. The Orbit virtual machine contains no instructions for manipulating the program counter register.

The virtual machine contains a special 1-bit status register. This register is written by a compare-to-zero instruction, and read by a control-join instruction.

The machine interacts with peripherals through a port-mapped I/O interface. The machine contains a 14-bit address space of doubles for input ports along with a disjoint 14-bit address space of doubles for output ports. Values are read from input ports using an `Input` instruction, and written to output ports with a `Output` instruction.

Instructions are segmented into two categories: D-type instructions require two operands, and S-type instructions require one operand. D-type instructions have the form `Op` r_1 r_2, where r_1 and r_2 are data memory addresses. Similarly, S-type instructions have the form `Op` r_1, where r_1 is a data memory address. In general, the execution of an instruction at address i_{dest} will read operand values at the addresses indicated from the data memory, execute the operator on those values, and then store the result in the data memory address r_{dest} corresponding to the instruction memory address i_{dest}. Consequently, the destination address

Instruction	OPCODE	Semantics	Note
Add r_1 r_2	0x1	$r_d \leftarrow$ mem[r_1] + mem[r_2]	
Sub r_1 r_2	0x2	$r_d \leftarrow$ mem[r_1] - mem[r_2]	
Mult r_1 r_2	0x3	$r_d \leftarrow$ mem[r_1] * mem[r_2]	
Div r_1 r_2	0x4	if mem[r_2] = 0.0	
		then $r_d \leftarrow$ 0.0	
		else $r_d \leftarrow$ mem[r_1] / mem[r_2]	
Output r_1 r_2	0x5	port[r_1] $\leftarrow$ mem[r_2]	
Phi r_1 r_2	0x6	if status = '1'	
		then $r_d \leftarrow$mem[r_1]	
		else $r_d \leftarrow$mem[r_2]	

Table 1: D-Type Instructions

is omitted in the instruction encoding, as it is implied by the address of the instruction[1].

2.2 D-Type Instructions

The VM's D-Type instructions are two-argument instructions. Each instruction has an implicit destination address; this address is the same as the address of the instruction. The instruction is encoded as a 4-bit op-code and two 14-bit source registers. The op-code occupies bits 31-28, address r_1 occupies bits 27-14, and address r_2 occupies bits 13-0, as shown in the following table. The op-code encoding and semantics of each instruction is shown in table 1.

31 28	27 14	13 0
OP	r_1	r_2

2.3 S-Type Instructions

As with the D-type instructions, each S-type instruction has an implicit destination address which corresponds to the instruction address. Each S-type instruction, however, has a single address argument. In the encoding of an S-type instruction, bits 31-28 contain the value 0x0. Bits 27-24 contain the S-type instruction op-code. Bits 23-14 are reserved for instruction-specific immediate parameters. Finally, bits 13-0 contain the source operand address. The diagram below represents this encoding graphically. Op-code encodings and instruction semantics are defined in table 2.

31 28	27 24	23 14	13 0
0x0	OP	IMM	r_1

[1]There are two exceptions to this destination address mode: the compare-to-zero and the output instructions. Both are described below.

Instruction	OPCODE	Semantics	Note
Noop	0x0	$r_d \leftarrow \text{mem}[r_d]$	
Cmpz *imm* r_1	0x1	status$\leftarrow mem[r_1]$ *op* 0.0	[a]
Sqrt r_1	0x2	$r_d \leftarrow \lvert\sqrt{mem[r_1]}\rvert$	
Copy r_1	0x3	$r_d \leftarrow \text{mem}[r_1]$	
Input r_1	0x4	$r_d \leftarrow \text{port}[r_1]$	

[a]The operator is indicated by the immediate mode argument.

Table 2: S-Type Instructions

The Cmpz (compare to zero) instruction uses a 4-bit immediate operand, occupying bits 23-20 of the instruction, to indicate the comparison to be performed. The result of this operator is stored in the boolean status register. Each operator compares the input operand with zero. Table 3 shows the comparison op-codes along with the associated encoding and operators.

OPCODE	Encoding	Operation
LTZ	0x0	$<$
LEZ	0x1	$\leq$
EQZ	0x2	$=$
GEZ	0x3	$\geq$
GTZ	0x4	$>$

Table 3: Comparison Operations

2.4 Orbit Executable Format

The Orbit executable format stores the values for the instruction memory and the initial contents of the data memory. These values are stored in the executable format in sequential order, starting with the values for address zero. Instructions are stored as 32-bit words, encoded according to the format described in sections 2.2 and 2.3. Data memory values are stored as 64-bit double precision floating point values. Both instruction and data values are stored in the binary format in little-endian byte order.

The pairing of a 32-bit instruction and a 64-bit data value constitutes a 96-bit frame in the executable file. The order of the instruction and data values varies, depending on the frame address. For frames corresponding to even addresses, the first eight bytes of the frame contain to the double data value, while the next four bytes constitute the instruction value. Conversely, for frames corresponding to odd addresses, the first four bytes contain the instruction value and then next eight bytes contain the data value.

An Orbit binary may not contain values for the entire address range. If an executable is smaller than the complete address range for the Orbit VM,

the loader should assume that addresses for the instruction and data memories beyond those supplied by the binary contain the `Noop` instruction and the value `0.0`, respectively.

3 Solution Submission

The contest consists of a sequence of problems, each of which consists of a collection of scenarios. Associated with each problem is a binary image, in the format described in section 2.4. Contestants will write a program that will produce actuator inputs on the virtual machine's input ports to control the body represented in the binary to accomplish the defined task. Upon completing the task, the binary will present a score for that scenario which takes into account the scoring criteria as defined by the problem description. The contestants will upload a scenario solution binary that represents the series of actuator inputs used to arrive at their solution.

This scenario solution binary contains the following information:

- The team's identification number, available from the contest website.
- The scenario identification number, also available from the contest website.
- The sequence of actuator inputs used to solve the task.

This data is encoded in the scenario solution binary format as described below. The file format consists of a header, which includes a file magic number, the team identifier, and the scenario identifier. This is followed by a series of simulation frames. Table 4 shows this format graphically.

byte	value
0	0xCAFEBABE
4	TeamID
8	ScenarioID
12	TimeStep_0
16	Count (e.g. k)
20	Addr_i
24	Value_i
28	
	$\vdots$
20+k*12	TimeStep_1
24+k*12	Count
	$\vdots$

Table 4: Submission File Format

Bytes 0-3 of the file header should contain the value 0xCAFEBABE. Bytes 4-7 contain the team identification number. Bytes 8-11 contain the scenario number. Each of these value are 32-bit unsigned integers. All values, regardless of bit size, are stored in little-endian byte order.

The file header is followed by a sequence of simulation frames. These frames are used to identify the actuator inputs for a given time step, and must appear in strictly ascending time step order. To minimize the space needed to represent these values, the simulation frame should only contain values for ports which differ from the previous time step. If an input port's value is unchanged by the contestant's control program, then that port's value should be excluded from the simulation frame.

The simulation frame contains a frame header followed by a collection of port value mappings. The frame header contains the time step of the frame, followed by a count of port value mappings for that frame. Both the time step and count are represented as 32-bit unsigned integers, stored in little-endian byte order. Following the frame header are the collection of port value mappings.

A port value mapping consists of an input port address followed by a corresponding value. The port address is stored as a 32-bit unsigned integer in little-endian byte order. Since the 32-bit address representation is larger than the 14-bits of the virtual machine's port address space, the port address is zero-padded, such that bits 31-14 in the port value mapping address is all zeros, with bits 13-0 containing the actual port address.The port address is followed by the assigned value, stored as a 64-bit double precision floating point value, in little-endian byte order.

The final frame of a solution submission should contain a header with the time step immediately after the step in which the simulation binary reported a score. A valid solution must have a final time step less than 3 Million seconds. Moreover, the final frame should contain no port mappings, but simply indicate it is the final frame by setting the count for that frame to zero.

4 Physics

Each body has a position in space. The position s of a body at a time t is denoted s_t. A body also has a velocity v_t, and an acceleration a_t. The body's position at the next time step $t + 1$, denoted s_{t+1}, is a function of the current position, the velocity, the acceleration, and the amount of time between steps, Δt. The physics equations use SI units: Kg for mass, meters for distance, Newtons for force, m/s for velocity, and m/s^2 for acceleration.

$$s_{t+1} = s_t + v_t \cdot \Delta t + \frac{1}{2} a_t \cdot \Delta t^2 \quad (1)$$

When a force is applied to a body, it will undergo an acceleration, which is a function of the mass and the force, as related by the equation.

$$F = m \cdot a \tag{2}$$

Gravity produces an attractive force, F_g between two bodies. This force is a function of the masses of the two bodies, the distance between the two, r, and a universal gravitational constant G.

$$F_g = \frac{Gm_1m_2}{r^2} \tag{3}$$

$$G = 6.67428 \cdot 10^{-11} \tag{4}$$

5 Simulation Properties

The simulation binaries implement a discrete approximation of the physics equations defined in section 4. This approximation is a result of quantization in both space and time. Moreover, the simulation contains a collection of simplifying assumptions.

1. The mass of each satellite is negligible, and the resulting force upon the earth is zero.
2. The gravitational force between two satellites is negligible, and is disregarded.
3. The distance between two bodies is calculated from the centers of the bodies.
4. The simulation proceeds with a constant Δt of 1 second.
5. The mass of the earth, m_e, is $6.0 \cdot 10^{24}$ Kg.
6. The radius of the earth is $6.357 \cdot 10^6$ meters. A satellite that passes within this radius is assumed to collide with the earth.
7. Position, velocity, and acceleration are restricted to two dimensions.
8. Velocity and acceleration are constant for the duration of the time step.

The distance between two bodies with positions s_1 and s_2 is calculated by decomposing those positions into x and y coordinates, denoted s_x and s_y, as defined:

$$r = \sqrt{(s_{x_1} - s_{x_2})^2 + (s_{y_1} - s_{y_2})^2} \tag{5}$$

The velocity of a body at time $t+1$ is a function of the body's velocity at time t, any acceleration from thrusting, and acceleration from gravity. Gravitational

acceleration g_t and g_{t+1} are defined by equation 3. The acceleration due to thruster firing results in a change in velocity, ΔV, over the duration of the time step Δt.

$$g_t = \frac{Gm_e}{r^2} \tag{6}$$

$$s_{t+1} = s_t + (v_t \cdot \Delta t) + \frac{1}{2} \cdot (g_t + \Delta V) \cdot \Delta t^2 \tag{7}$$

$$v_{t+1} = v_t + (\Delta V + \frac{g_t + g_{t+1}}{2}) \cdot \Delta t \tag{8}$$

6 Problems

The contest consists of four problems. Each problem requires the contestants to write a program that will control a satellite to accomplish a given task. For example, the goal of the first problem is to move a satellite from one circular orbit into a different circular orbit. Furthermore, for each problem, there are a series of scenarios. Each scenario represents an instance of the problem with the satellites in a different starting configuration. Each problem has an associated problem binary, available for download on the contest website. The contestants select a specific scenario for a given problem by assigning a configuration value to the virtual machine input port at address `0x3E80`. Table 5 shows the scenario configuration numbers for the various problems.

Problem	Configuration Range
Hohmann	1001-1004
Meet & Greet	2001-2004
Eccentric	3001-3004
Clear Skies	4001-4004

Table 5: Scenario Configuration Numbers

For all problems, the contestants control the satellite by providing input to a thruster actuator. The control program can observe and react to the environment using a variety of sensors. The actuator inputs and sensor outputs are mapped to specific virtual machine input and output ports, respectively. The port mapping is defined in the specification for each problem.

The actuator input port mappings for all four problems are shown in table 6. The configuration port is described above. The ΔV input represents an actuator input which will result in a change in velocity over the next time step of the simulation, expressed in meters/sec. This value is a vector, and the ports correspond to the x and y components of that vector.

Input port address	Actuator
0x2	ΔV_x
0x3	ΔV_y
0x3E80	Configuration

Table 6: Actuator input ports

6.1 Hohmann

The Hohmann problem requires the controller to transfer the satellite, initially in a circular orbit around earth, into a different circular orbit. The simulator exposes environment sensor data, as shown in table 7.

The Score sensor is the score calculated (according to the formulae in section 7) by the simulation binary. This score will be 0.0 initially, and will only change when the task has been completed successfully. The Fuel Remaining sensor represents a fuel gauge, measured in units of $|\Delta V|$. At time step 0, the output of this sensor represents a full fuel tank. This output will decrease as the thrust actuator is employed by the controller program.

The position of the controlled satellite relative to earth is decomposed into vector components s_x and s_y. The radius of the target orbit is exposed by the final sensor.

Output port address	Sensor
0x0	Score
0x1	Fuel Remaining
0x2	s_x relative to earth
0x3	s_y relative to earth
0x4	Target orbit radius

Table 7: Hohmann Problem sensor output ports

6.2 Meet and Greet

In Meet and Greet, the contestant must move a satellite in a circular orbit about earth to meet with a second satellite, also in a circular orbit. The target satellite will maintain its orbit throughout the simulation. To successfully complete this challenge, the position of the controlled satellite must remain within 1km of the target for 900 consecutive seconds.

The sensors associated with the Meet and Greet problem binary are shown in table 8. These are the same as the Hohmann problem, with the addition of two ports that give the vector to the target satellite from the controlled satellite.

6.3 Eccentric Meet and Greet

This problem generalizes the Meet and Greet problem. The task is still to move the satellite from one orbit to meet with a target satellite. However, the orbits

Output port address	Sensor
0x0	Score
0x1	Fuel Remaining
0x2	s_x relative to earth
0x3	s_y relative to earth
0x4	s_x relative to target satellite
0x5	s_y relative to target satellite

Table 8: Meet and Greet sensor output ports

of both satellites can be arbitrary ellipses. As with the Meet and Greet problem, the completion criteria requires the controlled satellite to remain within 1km of the target for 900 consecutive seconds. The sensor outputs for this problem are the same as for the Meet and Greet problem, as shown table 8. Again, the actuator inputs are those shown in table 6

6.4 Operation Clear Skies

The challenge of the Operation Clear Skies problem is to control the satellite so that it visits a collection of twelve target satellites, each of which are in an elliptical orbit around earth. The binary for Operation Clear Skies will be released 24 hours after the start of the contest, following the conclusion of the lightning round.

To successfully visit a target satellite, the controlled satellite must pass within 1km of the satellite. There is no requirement that the satellite maintain a position within this radius for any longer than one time step. Note that it may be possible to cross paths with the target satellite without being within the maximum radius at a given time step. This will not count as successfully visiting the satellite.

Because the Operation Clear Skies problem may require a large number of orbit transfer maneuvers, there will be a special refueling station in orbit. Whenever the controlled satellite passes within the visiting radius of the refueling station, the amount of fuel required to fill the controlled satellite's tank will be transferred from the station to the controlled satellite.

The sensor output port mappings for the Operation Clear Skies problem are shown in table 9. The sensors for this problem include a relative position vector between the controlled satellite and each target satellite, as well as the fueling station. There are twelve target satellites total, labeled $\text{target}_0 \ldots \text{target}_{11}$, in addition to the fueling station. In the sensor port mapping, each target satellite has a sensor output which indicates whether that satellite has been successfully visited. This output will be `1.0` if the satellite has been visited, `0.0` otherwise.

Output port address	Sensor
0x0	Score
0x1	Fuel Remaining
0x2	s_x relative to earth
0x3	s_y relative to earth
0x4	s_x relative to fueling station
0x5	s_y relative to fueling station
0x6	Fuel remaining on fuel station
0x7	s_x relative to target_0
0x8	s_y relative to target_0
0x9	target_0 successfully collected
0xA	s_x relative to target_1
0xB	s_y relative to target_1
0xC	target_1 successfully collected
	$\vdots$
$3 \cdot k$ + 0x7	s_x relative to target_k
$3 \cdot k$ + 0x8	s_y relative to target_k
$3 \cdot k$ + 0x9	target_k successfully collected

Table 9: Operation Clear Skies sensor output ports

7 Scoring Criteria

The scoring for the contest will be on a total points basis. The scoring formula for each problem is described below. For each problem scenario successfully completed, the scoring formula will be used to calculate the score for that scenario. The scenario scores will be summed to form a problem score. The problem scores will, in turn, be summed to calculate a contest score. In the case of a tie, the team with the earliest submission *for all scenarios* will win the tiebreaker.

In the event that the controlled satellite collides with the earth or the controller produces an actuator input that is consumes more than the amount of fuel remaining in the satellite, the score for that scenario will be -1.0.

For the Hohmann, Meet and Greet, and Eccentric Meet and Greet, the score for a scenario will be calculated using the following formula. The unscaled score is the sum of the points for completing the task, the ratio of the fuel remaining over the fuel at the start of the scenario, and a time bonus. The fuel bonus is 45 points times the percentage of fuel remaining. The time bonus is 30 points, minus a time penalty that is calculated as the base-two logarithm of the number of seconds taken to complete the task divided by 1000. This score is multiplied by scaling factor. The factor is 1.0 for the Hohmann problem, 2.0 for the Meet and Greet problem, and 4.0 for the Eccentric Meet and Greet problem.

$$unscaled \quad = \quad 25 + 45 \cdot (f_{remaining}/f_{start}) + (30 - lg\frac{t}{1000}) \tag{9}$$

Scoring for Operation Clear Skies utilizes a separate formula, shown below. For each target satellite, a countdown timer is initialized to $2 \cdot 10^6$. This countdown timer for each satellite is decremented by one every time step, until that satellite has been visited, at which point the countdown for the satellite ends, and a score t_i is recorded for that satellite. After $2 \cdot 10^6$ seconds, the scores for all satellites summed and divided by $24 \cdot 10^6$. This factor is multiplied by 75 to get the total time component of the score. Additionally, the score incorporates a fuel factor, which is calculated similarly to that of the first three problems, as $f_{remaining}/f_{start}$. However, this ratio is of the total fuel, both stored on the satellite as well as that in the fueling station. The fueling factor is multiplied by 25. Finally, the sum of the time and fueling factors are multiplied by 8.0.

$$t_i = 2 \cdot 10^6 - time_i \tag{10}$$

$$score_t = \frac{\sum_{i=0}^{11} t_i}{24 \cdot 10^6} \tag{11}$$

$$unscaled = 75 \cdot score_t + 25 \cdot (f_{remaining}/f_{start}) \tag{12}$$

$$score = 8.0 \cdot unscaled \tag{13}$$

8 Evaluation Procedure

During the 72-hour duration of the competition, contestants will be able to upload problem solutions. For each problem, the contestant will upload a trace for each scenario associated with the problem. Moreover, the contestants will upload an archive which contains the complete source of the all problem solutions. Contestants can re-upload solutions. For the overall competition, **the organizers will base the team score solely on most recently uploaded solution**.

For the lightning round competition, the organizers will base scoring on the latest solutions within the 24-hour lightning round window. Uploads after the 24-hour time limit will not invalidate lightning round solutions, but will not be considered for the lightning round. Thus, a team can submit a solution for the lightning round, and then submit an improved solution for the overall competition.

Immediately upon uploading, the solution will be marked *unverified*. The organizers will then execute the scenario binary on a reference virtual machine, with the supplied trace, to verify that the submitted solution's completes the task and to compute the score. The solution will then be marked *verified*. Throughout the competition, the organizers may post preliminary scores based on verified solutions. However, these scores are not to be treated as final results until the solutions have been validated, as described below.

Solutions for the top ten teams will be validated, after the contest concludes, by generating a new collection of scenarios for each problem. The team's submitted source code for the problem solutions will be be executed on these new scenarios to verify that solutions submitted are general solutions to the problem

and that those solutions utilize only the sensor output exposed as described in the problem description. The rankings of the contestants in the validation round will be used to determine the final contest placement. To be eligible for consideration in the validation round, the contestants must agree to assist the contest organizers in compiling and executing the submitted problem solutions.

Appendix: Hohmann Transfer

The Hohmann transfer is a fuel efficient way to move between two circular orbits. The Hohmann transfer orbit is a elliptical orbit in which the semi-major axis (axis of largest diameter) is equal to the diameter of the larger circular orbit. In Figure 1, this is orbit 3. The transfer requires two engine impulses: one to enter the elliptical orbit and one to enter the target orbit.

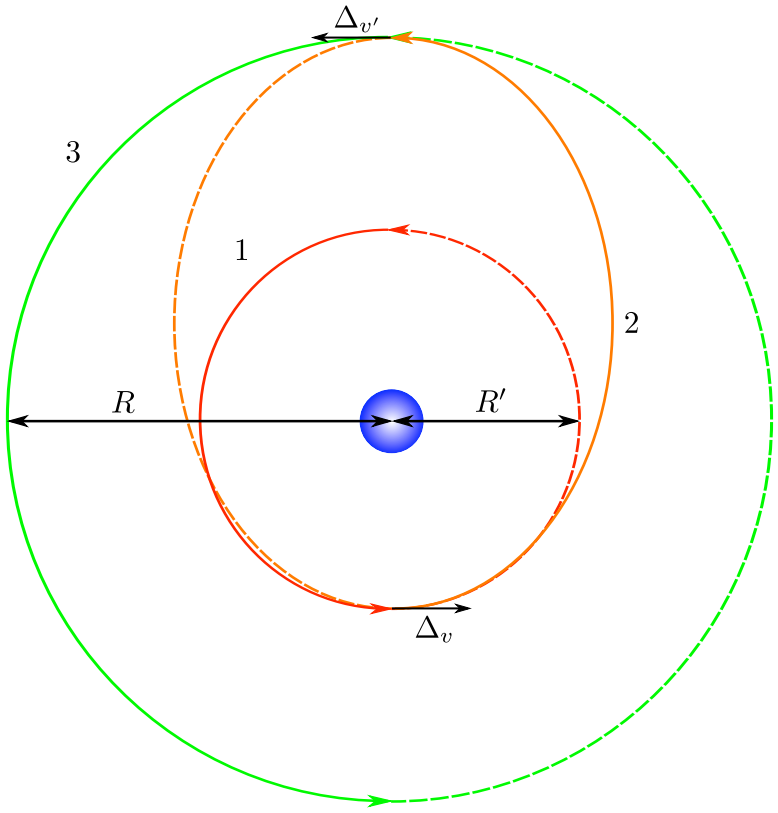

Figure 1: Hohmann Transfer Orbit

Figure 1 illustrates a Hohmann transfer to a larger orbit. The solid lines represent the path a spacecraft would take from the inner orbit (1) to the outer orbit (3). Δ_v corresponds to the change in velocity as a result of the first engine

impulse. This impulse must accelerate the ship in the direction it is currently traveling (i.e. tangent to the orbit). The impulse will place the ship into an elliptical orbit (2). The ship will travel half of the elliptical orbit at which point it will arrive at radius R' and require another engine impulse to accelerate it into the larger circular orbit (3).

Note that in the example above, the Hohmann transfer is used to move to a higher altitude orbit. However, the Hohmann transfer can be used to move to a lower altitude orbit as well. In that case, the thrust applied at Δ_v and $\Delta_{v'}$ will be negative (i.e. in the opposite direction the ship is traveling) to slow the ship rather than accelerate it.

The mathematical equations for Δ_v and $\Delta_{v'}$ are beyond the scope of this document but are discussed in the Wikipedia article here.