

CSF Simulation Specifications and Implementation Details

March 13, 2017

1 Specifications

CSF provides easy-to-use general-purpose simulation interface.

A complete simulation consists of definitions of *objects*, *watches*, *rules* and optionally *assertions*.

A simulation will use data from the system, applying rules to generate new data, and log the specified variables. (see below for details).

1.1 Predefined variables and constants

Predefined values are all globally accessible.

variable	meaning	vector?	comment
t	time	N	
g	gravitational acceleration	Y	constant
G	gravitational constant	N	constant
c	speed of light	N	constant
e	base of natural logarithm	N	constant
π (pi)	circles' circumference / diameter	N	constant

Constant value may be override. Constant means the value *should* not change during simulation.

RF must never be actively updated since it has no storage, rather, it should be computed every time referenced.

1.2 Mathematical representations

Expressions with elementary operations (addition, subtraction, multiplication, division, and powering), trigonometric and inverse trigonometric functions, and logarithm must be implemented.

Vectors in expressions are handled as in mathematics, but may have different syntax with scalars and are not required to be mixed with scalar expressions.

1.3 Identifier

Identifier is anything used to distinguish objects. Identifier must be unique to any object within the same type. Identifier may be either input by user or assigned internally by an implementation. One object may have multiple identifiers. Objects may be referenced by identifier if the given identifier compares equal to that of one object.

1.4 Coordinate system

CSF by default uses cartesian system, and provides support for temporary local system, which can be spherical, cylindrical or cartesian. Temporary system may be used to locate new objects, but cannot be used in simulation.

1.5 Object

Object refers to any physical object involved in a physical process.

User specified variable are unconditionally watched. (see [Watch](#))

1.6 Rule

Rules are used to provide information of interaction and define how interactions are applied to *objects*.

Rules can be thought as extended form of forces **or** constraints physically and routines programmatically.

Rules can be conditional or not and be applicable to a set of choice of *object* combinations or all combinations. Conditional *rule* may be associated with *rule* pack. A *rule* is conditional if it is only applicable when a predicate is met, and the predicate is not relevant to identity of an *object*. Conditional block will have all variables defined if they appeared new, no matter if they're explicitly stated in all part of the pack. An implementation must at least provide syntax for adding *rule* for specific *object* with arbitrary number of *objects* referenced either by identifier or by number.

Each *rule* has an attribute. Attribute are typically a short letter-only string used for sorting. Specially the attribute 'F' is reserved for computing predefined variable RF. Attribute of conditional blocks applies to all blocks that defines new variable.

The simulation may skip the rest if all needed information are obtained before reaching the end (see [Watch](#)). Implementation shall expect the system to be overdetermined, and should halt rather than silently solve the overdetermined system.

Rules are resolved in the order of user input. Implmentation are entitled to choose either to update variables in place or not. Implementation may simplify the resolving procedure if proved equivalent *mathematically*.

1.7 Assertion

Assertions are used to verify some condition or predicate through the simulation process. The simulator shall halt if an assertion fails. Assertions shall be verified immediately when simulator comes across it. If an assertion fails due to insufficient variable defined, simulator shall still report the error. [Note: Simulator shall report the error either statically or dynamically, yet is encouraged to do it as early as possible.]

1.8 Watch

Watch defines what variable are of interest in simulation. Watched variables are kept throughout the simulation process and will be written to output.

Watch specification is ordered.

1.9 Simulation

Simulation proceeds by increment time t by user defined step. Implementation may choose to use a different step no greater than the user set one.

Implementation shall at least support MKS, CGS and FPS unit system, and appropriate conversions.

The accuracy shall be at least 10^{-6} (that is, the error is less than 10^{-6}) each step with respect to your selected unit system absolutely **or** relatively. [*Example:* 1×10^{-7} may compare equal to 5×10^{-7} and 10^7 may compare equal to $10^7 + 1$.] For implementation with binary floating-point numbers, additional rounding error must be considered.

1.10 Standard library

1.10.1 `mechanics/mass_point`

This file defines common dynamic object with default values.

1.10.2 `mechanics/rules`

This file defines several common passive forces and law of real mass point motion.

1. Normal force applies \mathbf{F}_N if $\mathbf{pos}_i - \mathbf{pos}_j = \mathbf{0}$ and $\mathbf{v}_i \times \mathbf{v}_j = \mathbf{0}$ where the subscript i and j refers to distinct *objects*. For tangent plane defined by $\mathbf{p} = r\mathbf{u} + s\mathbf{w} + \mathbf{p}_0$, the effect is that some \mathbf{F}_N satisfying $\mathbf{F}_N \times (\mathbf{u} \times \mathbf{w}) = \mathbf{0}$ [*Note:* that is, $\mathbf{F}_N = k\mathbf{u} \times \mathbf{w}$] is applied so that $(\mathbf{a}_i - \mathbf{a}_j) \cdot (\mathbf{u} \times \mathbf{w}) = \mathbf{0}$, and \mathbf{F}_N is either defined to the resolved value or $\mathbf{0}$ if not applicable.
2. Collision solver is triggered if $\mathbf{pos}_i - \mathbf{pos}_j = \mathbf{0}$ where the subscript i and j refers to distinct *objects*. The total momentum of the pair is conserved, with coefficient of restitution e defined by material.
3. Static friction \mathbf{f} applies if $\mathbf{v}_i - \mathbf{v}_j = \mathbf{0}$ and $\mathbf{F}_N \neq \mathbf{0}$. For the plane defined above and the effect is that to the extent $\mathbf{f} \leq \mu_s \mathbf{F}_N$ and $\mathbf{f} \cdot (\mathbf{a} \times \mathbf{b}) = \mathbf{0}$, \mathbf{f} is applied and hereby defined so that $\mathbf{v}_i = \mathbf{v}_j$, otherwise \mathbf{f} is undefined.
4. Dynamic friction \mathbf{f} applies if $\mathbf{v}_i - \mathbf{v}_j \neq \mathbf{0}$ and $\mathbf{F}_N \neq \mathbf{0}$ and the effect is $\mathbf{f} = -\frac{\mu \|\mathbf{F}_N\|}{\|\mathbf{v}_i - \mathbf{v}_j\|} \cdot (\mathbf{v}_i - \mathbf{v}_j)$.
5. Newton's second law of motion, $\mathbf{R}\mathbf{F} = m\mathbf{a}$.
6. Velocity is integration of acceleration w.r.t time, $\dot{\mathbf{v}} = \mathbf{a}$
7. Displacement is integration of velocity w.r.t time, $\dot{\mathbf{pos}} = \mathbf{v}$

2 Implementation

2.1 General

CSF is implemented in static fashion.



The implementation is comprised of a compiler and an interpreter, and source file is firstly processed to produce CSF bytecode and then interpreted by the interpreter.

2.2 Source format and syntax

Source files usually have `.csf` extension. Complete source files consist of exactly three parts, namely objects, rules and start directive, one after another. A source file may contain exactly one start directive which must be the last non blank line. One line may contain exactly one statement. The source file is newline-sensitive and indent-sensitive. Specially everything after `#` till newline are treated as comment.

2.2.1 Accessing watched variables of another object

Watched variables of other *objects* can be accessed by adding `$n`, where `n` is a nonnegative integer. `$1`, `$2` ... point to other *objects* arranged in ascending order of definition. The implementation will provide arguments to the largest number given as sorted permutation. [Note: Explicit state of `$0` is unnecessary and not recommended.]

2.2.2 Units and literals

The default unit system is the SI system. Default system is applied to numbers when a specific dimension is needed and user has not provided one. User can set default system with compiler option.

User may use explicit unit literals anywhere when a dimension is needed. This number will be converted to the default system automatically by the compiler.

2.2.3 Protected names

There are a few names that are used in the implementation or have special meaning or are reserved for future use, so that they must not be used as identifiers.

None	True	False	global	using
object	rule	const	scalar	vector
if	else	assert	start	step
watch	def	Inf	NaN	Pi

Beside those listed above, identifiers shall not clash with names of specification mandated global variables.

2.2.4 Expressions and vectors in expressions

Expressions resembles they are in mathematical context but indices on functions like $\sin^2()$ are not yet accepted.

Symbols are treated as scalars unless declared to be vectors or otherwise specified. The declaration may either occur in *object* definition and *rule* definition. Vector addition and subtraction can be

done in the way it was, but vector plus or minus scalar will be rejected. Vectors have different syntax on multiplication and some other operations.

Vector scalar multiplication is handled via method `.mul`. The method `.dot` and `.cross` are for inner product and for cross product, respectively. Vector also provides `.mag` for magnitude, `.angle` for angle between two vectors, `.unit` for obtaining unit vector, `.proj` to obtain projection over another vector. Users are generally encouraged to use these functions for better performance.

Vectors are immutable, which means once defined, it cannot be changed. However, they could be reassigned. All the methods provided with vector return constant.

2.2.5 Using directive

Using directive `"using" <filename>`

Using directives are used to import data from other file, which is done with plain textual replacement. Using directives may appear anywhere in a source file.

2.2.6 Object section

Object section may only contain *object* definitions.

Object definition `"object" <name> ":" "vector"? <property> "=" <value>`

Object must be defined before first reference. Values used to initialize *objects* must be known at compile time. [Note: Indirect compile time constant are accepted.]

2.2.7 Rule section

Vector declarations, *rule* and conditional *rule* definitions and assertions may appear in this section.

Rule definition `"rule" ":" <eqn>`

Vector declaration `"vector" <name>`

Conditional rule definition `"rule" ":" "if" <condition> ":" <eqn>+`

Assertion `"assert" <condition>`

2.2.8 Start directive

Start directive

`"inc" <var> ["till" <time>] ["by" <time>] ["watch" <var>(","<var>)*]`

2.3 Object

Object is dynamically defined in compilation. The fields of the object shall all be initialized, or will be write as 0 bytes.

Objects are stored in an array of size 128.

2.4 Rule

Rules are resolved by substituting computed variable into the pending one and the resolving order.

Conditional *rules* are packed into a block, which are interpreted as a single function that takes the union of all variables used in any part of the block, including condition.

The simulator will abort if some *rule* cannot be resolved at the time simulator comes across it or one *rule* has all variables defined. In parallelized environment, the simulator shall behave the same as unparallelized.

2.4.1 Determining the variable(s) to solve for

When the simulator come across a *rule*, it has to determine the variable(s) to solve for.

The simulator follows the following precedence rules:

1. unwatched temporary variables
2. watched variables
3. global variables
4. constant values (for comparison only, results in an error)

[*Example:* For a *rule* $F = -2 * v$, F will be updated since v has lower precedence.][*Note:* temporary variables shall not have the same name with global variables.]

When there are multiple variables with same precedence, the order of presence or definition is used.

[*Example:* For *rule* $\mathbf{pos} = \mathbf{v}$, under default watches, it would use \mathbf{v} to update \mathbf{pos} , because \mathbf{v} is *watched* before \mathbf{pos} .]

The exact order is determined as a part of compilation.

2.5 Interpreter

THIS PAGE INTENTIONALLY LEFT BLANK