A Verified Design of a Fault-Tolerant Clock Synchronization Circuit: Preliminary Investigations

Paul S. Miner

March 1992
Abstract

Schneider [1] demonstrates that many fault-tolerant clock synchronization algorithms can be represented as refinements of a single proven correct paradigm. Shankar [2] provides a mechanical proof (using EHDM [3]) that Schneider's schema achieves Byzantine fault-tolerant clock synchronization provided that eleven constraints are satisfied. Some of the constraints are assumptions about physical properties of the system and can not be established formally. Proofs are given (in EHDM) that the fault-tolerant midpoint convergence function satisfies three of these constraints. This paper presents a hardware design, implementing the fault-tolerant midpoint function, which will be shown to satisfy the remaining constraints. The synchronization circuit will recover completely from transient faults provided the maximum fault assumption is not violated. The initialization protocol for the circuit also provides a recovery mechanism from total system failure caused by correlated transient faults.
## Contents

1 Introduction .............................................. 1

2 Description of the Reliable Computing Platform .......... 2

3 Clock Definitions ........................................... 4
   3.1 Shankar's Notation ..................................... 5
   3.2 Shankar's Conditions ................................... 6

4 Fault-Tolerant Midpoint as an Instance of Schneider's Schema ... 8
   4.1 Translation Invariance .................................. 9
   4.2 Precision Enhancement .................................. 9
   4.3 Accuracy Preservation .................................. 12
   4.4 EHDM Proofs of Convergence Properties .............. 13

5 Proposed Verification ....................................... 14
   5.1 Informal Description .................................... 14
   5.2 Correctness Criteria .................................... 17

6 Transient Recovery .......................................... 17
   6.1 Single Fault Scenario ................................... 18
   6.2 General Case ........................................... 18
   6.3 Comparison with Other Approaches .................... 18

7 Initial Synchronization ...................................... 19
   7.1 Mechanisms for Initialization ......................... 19
   7.2 Comparison to Other Approaches ..................... 20

8 Concluding Remarks ......................................... 20

A Proof Summary ............................................... 22

B \LaTeX{} printed EHDM Modules ............................... 23

C Proof Chain Status .......................................... 79
   C.1 Translation Invariance .................................. 79
   C.2 Precision Enhancement .................................. 81
   C.3 Accuracy Preservation .................................. 88
1 Introduction

NASA Langley Research Center is currently involved in the development of a formally verified Reliable Computing Platform (RCP) for real-time digital flight control systems [4, 5, 6]. An often quoted requirement for critical systems employed for civil air transport is a probability of catastrophic failure less than $10^{-9}$ for a 10 hour flight [7]. Since failure rates for digital devices are on the order of $10^{-6}$ per hour [8], hardware redundancy is required to achieve the desired level of reliability. While there are many ways of incorporating redundant hardware, the approach taken in the RCP is the use of identical redundant channels with exact match voting (see [4, 5] and [6]).

A critical function in a fault-tolerant system is that of synchronizing the clocks of the redundant computing elements. The clocks must be synchronized in order to provide coordinated action among the redundant sites. Although perfect synchronization is not possible, clocks can be synchronized within a small skew. The purpose of this work is to provide a mechanically verified design of a fault-tolerant clock synchronization circuit.

The fault-tolerant clock synchronization circuit is intended to be part of a verified hardware base for the RCP. The primary intent of the RCP is to provide a verified fault-tolerant system which is proven to recover from a bounded number of transient faults. The current model of the system assumes (among other things) that the clocks are synchronized within a bounded skew [5]. It is crucial that the clock synchronization circuitry also be able to recover from transient faults. Originally, Lamport and Melliar-Smith's Interactive Convergence Algorithm (ICA) [9] was to be the basis for the clock synchronization hardware, the primary reason being the existence of a mechanical proof that the algorithm is correct [10]. However, modifications to ICA to achieve transient fault recovery are unnecessarily complicated. The fault-tolerant midpoint algorithm of [11] is more readily adapted to transient recovery.

The synchronization circuit is designed to tolerate arbitrarily malicious permanent, intermittent and transient hardware faults. A fault is defined as a physical perturbation altering the function implemented by a physical device. Intermittent faults are permanent physical faults which do not constantly alter the function of a device (e.g. a loose wire). A transient fault is a one shot short duration physical perturbation of a device (e.g. caused by a cosmic ray or other electromagnetic effect). Once the source of the fault is removed, the device can function correctly.

Most proofs of fault-tolerant clock synchronization algorithms are by
induction on the number of synchronization intervals. Usually, the base case of the induction, the initial skew, is assumed. The descriptions in [1, 2, 9, 10] all assume initial synchronization with no mention of how it is achieved. Others, including [11, 12, 13] and [14] address the issue of initial synchronization and give descriptions of how it is achieved in varying degrees of detail. In proving an implementation correct, the details of initial synchronization cannot be ignored. If the initialization scheme is robust enough, it can also serve as a recovery mechanism from multiple correlated transient failures (as is noted in [14]).

Schneider [1] demonstrates that many fault-tolerant clock synchronization algorithms can be represented as refinements of a single proven correct paradigm. Shankar [2] provides a mechanical proof (using EHDM [3]) that Schneider's schema achieves Byzantine fault-tolerant clock synchronization, provided that eleven constraints are satisfied. Some of the constraints are assumptions about physical properties of the system and can not be established formally. This paper proposes a hardware solution to the clock synchronization problem which will be shown to satisfy the remaining constraints.

This paper discusses preliminary results in the verification of the design. The fault-tolerant midpoint function is formally proven (in EHDM) to satisfy the properties of translation invariance, precision enhancement, and accuracy preservation. A register transfer level design is presented which implements the synchronization algorithm. An argument for transient recovery from a single fault is presented and issues relating to the more general case are raised. Finally, the approach for achieving initial synchronization is discussed. The notation used here is from Shankar [2].

2 Description of the Reliable Computing Platform

This section summarizes the key details of the Reliable Computing Platform to establish a context for the clock synchronization circuit. It is included here for completeness. The material in this section is paraphrased from Butler and DiVito [5]. The interested reader should consult [5] for more detailed information.

\footnote{These properties will be defined in the section describing the fault-tolerant midpoint convergence function.}
Figure 1: Hierarchical Specification of the Reliable Computing Platform.

NASA-Langley is currently involved in the development of a formally verified Reliable Computing Platform for real-time control [4, 5]. A primary goal is to provide a fault-tolerant computing base that appears to the application programmer as a single ultra-reliable computer. To achieve this, it is necessary to conceal implementation details of the system. Some characteristics of the system are as follows [5]:

- "the system is non-reconfigurable"
- the system is frame-synchronous
- the scheduling is static, non-preemptive
- internal voting is used to recover the state of a processor affected by a transient fault"

A hierarchy of models is introduced which provides different levels of abstraction (figure 1, taken from [5]). The top level is the view presented to the applications programmer, i.e. an ultra-reliable uniprocessor system. The details of fault-tolerance are introduced in the lower levels. The next two levels, replicated synchronous and distributed synchronous, introduce the redundancy and voting required for fault-tolerance, but assume perfectly synchronized clocks and an interactive consistency network for reliable distribution of single source data. The fourth level, distributed asynchronous, weakens the assumption of perfect synchrony to one where the clocks are synchronized to within a bounded skew. The details of the hardware/software implementation have yet to be worked out. An abstract view of the assumed
The hardware architecture is given in Figure 2 (from [5]). The clock synchronization circuit presented here is intended to serve as part of the verified hardware base at the lowest level of the hierarchy.

3 Clock Definitions

This section introduces the notation and assumptions used in Shankar's proof and is largely taken from sections 2.1 and 2.2 of [2]. The conditions enumerated here provide the formal specification for the clock synchronization circuit.
The reading of p's physical clock at real time t.

The reading of p's virtual clock at time t. This is the logical time used by the system.

The reading of p's i'th interval clock at real time t (Only sensible if \( t_i \leq t \)).

The real time that processor p begins the i'th synchronization interval.

Cumulative adjustment to p's physical clock up to and including \( t^i_p \).

An array of clock readings (local to p) such that \( \Theta^i_p(q) \) is p's reading of q's clock at \( t^i_p \).

Convergence function executed by p to establish correct \( V^iC(p(t^i_p)) \).

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( PC_p(t) )</td>
<td>The reading of p's physical clock at real time t.</td>
</tr>
<tr>
<td>( VC_p(t) )</td>
<td>The reading of p's virtual clock at time t. This is the logical time used by the system.</td>
</tr>
<tr>
<td>( IC^i_p(t) )</td>
<td>The reading of p's i'th interval clock at real time t (Only sensible if ( t^i \leq t )).</td>
</tr>
<tr>
<td>( t^i_p )</td>
<td>The real time that processor p begins the i'th synchronization interval.</td>
</tr>
<tr>
<td>( adj^i_p )</td>
<td>Cumulative adjustment to p's physical clock up to and including ( t^i_p ).</td>
</tr>
<tr>
<td>( \Theta^i_p )</td>
<td>An array of clock readings (local to p) such that ( \Theta^i_p(q) ) is p's reading of q's clock at ( t^i_p ).</td>
</tr>
<tr>
<td>( cfu(p, \Theta^i_p) )</td>
<td>Convergence function executed by p to establish correct ( PC^i_p(t^i_p) ).</td>
</tr>
</tbody>
</table>

Table 1: Clock Notation

3.1 Shankar's Notation

In general, clocks will be represented by different abstractions. Each redundant clock will incorporate a physical oscillator which marks passage of time. Each oscillator will drift with respect to real time by a small amount. Physical clocks derived from these oscillators will similarly drift with respect to each other. The purpose of a clock synchronization algorithm is to make periodic adjustments to local (virtual) clocks to keep redundant clocks within a bounded skew of each other. This periodic adjustment makes analysis difficult, so an interval clock abstraction is used in the proofs. This interval clock is indexed by the number of elapsed intervals since the beginning of the protocol. An interval corresponds to the elapsed time between adjustments to the virtual clock. The proof that synchronization is maintained is by induction on intervals.

Table 1 introduces the notation for the key elements required for a verified clock synchronization algorithm. Shankar outlines the following set of relationships between these values,

\[
\begin{align*}
adj^i_p + 1 &= cfu(p, \Theta^i+1) - PC^i_p(t^i+1) \\
adj^i_p &= 0 \\
IC^i_p(t) &= PC^i_p(t) + adj^i_p
\end{align*}
\]
VC_p(t) = IC_p(t), for t_p^i \leq t < t_p^{i+1}

presuming the presence of PC and VC, with an abstraction for IC used in the proofs. The following can be simply derived.

\[
\begin{align*}
V'C_p(t_p^{i+1}) &= IC_p^{i+1}(t_p^{i+1}) = cfn(p, \Theta_p^{i+1}) \\
V'C_p(t) &= cfn(p, \Theta_p^{i+1}) + PC_p(t) - PC_p(t_p^{i+1})
\end{align*}
\]

Using these equations and the eleven conditions outlined in the next section, Shankar mechanically verified Schneider’s paradigm. Some of the conditions will need to be modified in order to reason about transient recovery. It will then be necessary to rerun the EHDM proofs of the main theorem of [2] (below).

Any implementation which satisfies the constraints in Shankar’s report will provide the following guarantee.

**Theorem 1 (bounded skew)** For any two clocks p and q that are non-faulty at time t,

\[
|VC_p(t) - VC_q(t)| \leq \delta
\]

That is, the difference in time observed by two non-faulty clocks is bounded by a small amount. This gives the leverage needed to reliably build a fault-tolerant system. The next section enumerates the conditions to be met to guarantee this result.

### 3.2 Shankar’s Conditions

The first condition is initial skew, \(\delta_S\), which is a bound on the difference between good clocks at the beginning of the protocol.

<table>
<thead>
<tr>
<th>Condition 1 (initial skew)</th>
<th>For nonfaulty processors p and q</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(</td>
</tr>
</tbody>
</table>

The rate at which a good clock can drift from real-time is bounded by a small constant \(\rho\).²

²Notice that in this formulation a good clock must have been good continually since time 0. This condition will need to be modified in order to reason about recovery from transient faults.
Condition 2 (bounded drift) There is a nonnegative constant $\rho$ such that if clock $p$ is nonfaulty at time $s$, $s \geq t$, then

$$(1 - \rho)(s - t) \leq PC_p(s) - PC_p(t) \leq (1 + \rho)(s - t)$$

Shankar notes the following corollary to bounded drift which limits the amount two good clocks can drift with respect to each other during interval from $t$ to $s$.

$$|PC_p(s) - PC_q(s)| \leq |PC_p(t) - PC_q(t)| + 2\rho(s - t)$$

The next four conditions describe some constraints upon the synchronization interval as related to initial conditions of the protocol.

Condition 3 (bounded interval) For nonfaulty clock $p$

$$0 < r_{\text{min}} \leq t_{p}^{i+1} - t_{p}^{i} \leq r_{\text{max}}$$

Condition 4 (bounded delay) For nonfaulty clocks $p$ and $q$

$$|t_{q}^{i} - t_{p}^{i}| \leq \beta$$

Condition 5 (initial synchronization) For nonfaulty clock $p$

$$t_{p}^{0} = 0$$

Since we do not want process $q$ to start its $(i+1)$th clock before process $p$ starts its $i$th we state a nonoverlap condition

Condition 6 (nonoverlap)

$$\beta \leq r_{\text{min}}$$

This, with bounded interval and bounded delay, ensures that for good clocks $p$ and $q$, $t_{p}^{i} \leq t_{q}^{i+1}$.

All clock synchronization protocols require each process to obtain an estimate of the clock values for other processes within the system. Error in this estimate can be bounded, but not eliminated.
**Condition 7** (reading error) For nonfaulty clocks $p$ and $q$

$$|IC_q^{t(t+1)} - \Theta_p^{t+1}(q)| \leq \Lambda$$

There is bound to the number of faults which can be tolerated

**Condition 8** (bounded faults) At any time $t$, the number of faulty processes is at most $F$.

For the purpose of the algorithm presented here, we will assume that the number of clocks, $N$, satisfies the inequality $N \geq 3F + 1$.

Synchronization algorithms execute a convergence function $cnf(p, \theta)$ which must satisfy the conditions of translation invariance, precision enhancement, and accuracy preservation irrespective of the physical constraints on the system. Shankar mechanically proves that Lamport and Melliar-Smith's Interactive Convergence function [9] satisfies these three conditions. The next section defines these conditions in the context of the fault-tolerant midpoint function used by Welch and Lynch [11].

## 4 Fault-Tolerant Midpoint as an Instance of Schneider's Schema

The convergence function for the implementation described here is the fault-tolerant midpoint used by Welch and Lynch in [11]. The function consists of discarding the $F$ largest and $F$ smallest clock readings, and then determining the midpoint of the range of the remaining readings. Its formal definition is

$$cnf_{\text{MID}}(p, \theta) = \frac{\theta(F+1) + \theta(N-F)}{2}$$

where $\theta(m)$ returns the $m$th largest element in $\theta$. This formulation of the convergence function is different from that used in [11]. A proof of equality between the two formulations is not needed since it is shown that this formulation satisfies the properties required by Schneider's paradigm.

---

3This condition will need to be changed to "the number of processes not working . . .", where working will be a predicate analogous to the one used in [4, 5]. This is necessary for reasoning about recovery from transient failures.
This section presents informal proofs that \( cfnMID(p, \theta) \) satisfies the desired properties. The EHDM proofs are presented in the appendix and assume that there is a deterministic sorting algorithm which arranges the array of clock readings. This assumption will need to be discharged when the implementation is verified.

The properties presented in this section are applicable for any clock synchronization protocol which employs the fault-tolerant midpoint convergence function. All that will be required for a verified implementation is a proof that the function is correctly implemented and proofs that the other conditions have been satisfied.

4.1 Translation Invariance

*Translation invariance* states that the value obtained by adding \( x \) to the result of the convergence function should be the same as adding \( x \) to each of the clock readings used in evaluating the convergence function.

**Condition 9 (translation invariance)** *For any function \( \theta \) mapping clocks to clock values,*

\[
\text{cfn}(p, (\lambda n : \theta(n) + x)) = \text{cfn}(p, \theta) + x
\]

*Translation invariance* is evident by noticing that for all \( m \):

\[
(\lambda l : \theta(l) + x)_{(m)} = \theta_{(m)} + x
\]

and

\[
\frac{\theta_{(F+1)} + \theta_{(N-F)} + x}{2} = \frac{\theta_{(F+1)} + \theta_{(N-F)}}{2} + x
\]

4.2 Precision Enhancement

*Precision enhancement* is a formalization of the concept that, after executing the convergence function, the values of interest should be closer together.
Condition 10 (precision enhancement) Given any subset $C$ of the $N$ clocks with $|C| \geq N - F$, and clocks $p$ and $q$ in $C$, then for any readings $\gamma$ and $\theta$ satisfying the conditions

1. for any $l$ in $C$, $|\gamma(l) - \theta(l)| \leq x$
2. for any $l, m$ in $C$, $|\gamma(l) - \gamma(m)| \leq y$
3. for any $l, m$ in $C$, $|\theta(l) - \theta(m)| \leq y$

there is a bound $\pi(x, y)$ such that

$$|\text{cfn}(p, \gamma) - \text{cfn}(q, \theta)| \leq \pi(x, y)$$

Theorem 2 Precision Enhancement is satisfied for $\text{cfn}_{MID}(p, \theta)$ if

$$\pi(x, y) = \frac{y}{2} + x$$

One characteristic of $\text{cfn}_{MID}(p, \theta)$ is that it is possible for it to use readings from faulty clocks. If this occurs, we know that such readings are bounded by readings from good clocks. The next few lemmas establish this fact. To prove these lemmas it was necessary to develop a pigeon hole principle.

Lemma 1 (Pigeon Hole Principle) If $N$ is the number of clocks in the system, and $C_1$ and $C_2$ are subsets of these $N$ clocks,

$$|C_1| + |C_2| \geq N + k \supset |C_1 \cap C_2| \geq k$$

This principle greatly simplifies the existence proofs required to establish the next two lemmas. First, we establish that the values used in computing the convergence function are bounded by readings from good clocks.

Lemma 2 Given any subset $C$ of the $N$ clocks with $|C| \geq N - F$ and any reading $\theta$, there exists a $p, q \in C$ such that,

$$\theta(p) \geq \theta(F+1) \text{ (and } \theta(N-F) \geq \theta(q))$$
Proof: By definition, $|\{p : \theta(p) \geq \theta(F+1)\}| \geq F+1$ (similarly, $|\{q : \theta(N-F) \geq \theta(q)\}| \geq F+1$). The conclusion follows immediately from the pigeon hole principle.

Now we introduce a lemma that allows us to relate values from two different readings to the same good clock.

Lemma 3 Given any subset $C$ of the $N$ clocks with $|C| \geq N - F$ and readings $\theta$ and $\gamma$, there exists a $p \in C$ such that,

$$\theta(p) \geq \theta(N-F) \text{ and } \gamma(F+1) \geq \gamma(p).$$

Proof: Recalling that $N \geq 3F+1$, we can apply the pigeon hole principle twice. First to establish that $|\{p : \theta(p) \geq \theta(N-F)\} \cap C| \geq F+1$, and then to establish the conclusion.

A immediate consequence of the preceding lemma is that the readings used in computing $c_{\mu_{MID}}(p, \theta)$ bound a reading from a good clock.

The next lemma introduces a useful fact for bounding the difference between good clock values from different readings.

Lemma 4 Given any subset $C$ of the $N$ clocks, and clock readings $\theta$ and $\gamma$ such that for any $l$ in $C$, the bound $|\theta(l) - \gamma(l)| \leq x$ holds. for all $p, q \in C$.

$$\theta(p) \geq \theta(q) \land \gamma(q) \geq \gamma(p) \Rightarrow |\theta(p) - \gamma(q)| \leq x$$

Proof: By cases,

- If $\theta(p) \geq \gamma(q)$, then $|\theta(p) - \gamma(q)| \leq |\theta(p) - \gamma(p)| \leq x$
- If $\theta(p) \leq \gamma(q)$, then $|\theta(p) - \gamma(q)| \leq |\theta(q) - \gamma(q)| \leq x$

This enables us to establish the following lemma.

Lemma 5 Given any subset $C$ of the $N$ clocks, and clock readings $\theta$ and $\gamma$ such that for any $l$ in $C$, the bound $|\theta(l) - \gamma(l)| \leq x$ holds, there exist $p, q \in C$ such that,

$$\theta(p) \geq \theta(F+1),$$

$$\gamma(q) \geq \gamma(F+1), \text{ and}$$

$$|\theta(p) - \gamma(q)| \leq x.$$
Proof: We know from lemma 2 that there are $p_1, q_1 \in C$ that satisfy the first two conjuncts of the conclusion. There are three cases to consider:

- If $\gamma(p_1) > \gamma(q_1)$, let $p = q = p_1$.
- If $\theta(q_1) > \theta(p_1)$, let $p = q = q_1$.
- Otherwise, we have satisfied the hypotheses for lemma 4, so we let $p = p_1$ and $q = q_1$.

We are now able to establish precision enhancement for $cfn_{MID}(p, \gamma)$ (Theorem 2).

Proof: Without loss of generality, assume $cfn_{MID}(p, \gamma) \geq cfn_{MID}(q, \theta)$. 

\[
|cfn_{MID}(p, \gamma) - cfn_{MID}(q, \theta)| = |\gamma(F+1) + \gamma(N-F) - \theta(F+1) + \theta(N-F)| \\
= \left| \gamma(F+1) + \gamma(N-F) - \frac{\theta(F+1) + \theta(N-F)}{2} \right|
\]

Thus we need to show that

\[
|\gamma(F+1) + \gamma(N-F) - (\theta(F+1) + \theta(N-F))| \leq y + 2x
\]

By choosing good clocks $p, q$ from lemma 5, $p_1$ from lemma 3, and $q_1$ from the right conjunct of lemma 2, we establish

\[
|\gamma(F+1) + \gamma(N-F) - (\theta(F+1) + \theta(N-F))| \\
\leq |\gamma(q) + \gamma(p_1) - \theta(p_1) - \theta(q_1)| \\
= |\gamma(q) + (\theta(q) - \theta(p)) + \gamma(p_1) - \theta(p_1) - \theta(q_1)| \\
\leq |\theta(p) - \theta(q_1)| + |\gamma(q) - \theta(p)| + |\gamma(p_1) - \theta(p_1)| \\
\leq y + 2x \text{ (by hypotheses and lemma 5)}
\]

4.3 Accuracy Preservation

Accuracy preservation formalizes the notion that there should be a bound on the amount of correction applied in any synchronization interval.
### Condition 11 (accuracy preservation)
Given any subset $C$ of the $N$ clocks with $|C| \geq N - F$, and clock readings $\theta$ such that for any $l$ and $m$ in $C$, the bound $|\theta(l) - \theta(m)| \leq x$ holds, there is a bound $\alpha(x)$ such that for any $q$ in $C$:

$$|cfn(p, \theta) - \theta(q)| \leq \alpha(x)$$

### Theorem 3
**Accuracy preservation** is satisfied for $cfn_{MID}(p, \theta)$ if $\alpha(x) = x$.

**Proof:** Begin by selecting $p_1$ and $q_1$ using lemma 2. Clearly, $\theta(p_1) \geq cfn_{MID}(p, \theta)$ and $cfn_{MID}(p, \theta) \geq \theta(q_1)$. There are two cases to consider:

- If $\theta(q) \leq cfn_{MID}(p, \theta)$, then $|cfn_{MID}(p, \theta) - \theta(q)| \leq |\theta(p_1) - \theta(q)| \leq x$.
- If $\theta(q) \geq cfn_{MID}(p, \theta)$, then $|cfn_{MID}(p, \theta) - \theta(q)| \leq |\theta(q_1) - \theta(q)| \leq x$.

### 4.4 EHDM Proofs of Convergence Properties

This section presents the important details of the EHDM proofs that $cfn_{MID}(p, \theta)$ satisfies the convergence properties. In general, the proofs closely follow the presentation given above. The EHDM modules used in this effort are listed in the appendix. One underlying assumption is that $N \geq 3F + 1$. This is a well known requirement for systems to achieve Byzantine fault-tolerance without requiring authentication. Another assumption added for this effort states that the array of clock readings can be sorted. Additionally, a few properties one would expect to be true of a sorted array were assumed. These additional properties used in the EHDM proofs are (from module clocksort):

- **funsort_ax:** Axiom
  
  \begin{align*}
  i \leq j & \land j \leq N \supset \forall(\text{funsort}(\vartheta)(i) \geq \vartheta(\text{funsort}(\vartheta)(j))
  
  \end{align*}

- **funsort_trans_inv:** Axiom
  
  \begin{align*}
  k \leq N & \supset (\forall(\text{funsort}((\lambda q : \vartheta(q) + X))(k)) = \vartheta(\text{funsort}(\vartheta)(k)))
  
  \end{align*}

- **cnt_sort_geq:** Axiom
  
  \begin{align*}
  k \leq N & \supset \text{count}((\lambda p : \vartheta(p) \geq \vartheta(\text{funsort}(\vartheta)(k))), N) \geq k
  
  \end{align*}

- **cnt_sort_leq:** Axiom
  
  \begin{align*}
  k \leq N & \supset \text{count}((\lambda p : \vartheta(\text{funsort}(\vartheta)(k)) \geq \vartheta(p)), N) \geq N - k + 1
  
  \end{align*}
These properties will be proven in the context of the design.

A few of the given modules are taken from Shankar's proofs [2]. These include the arithmetic modules (absmod, multiplication, and division), clock-assumptions, and countmod. With the exception of countmod these modules were unaltered. A number of lemmas were added to (and proven in) module countmod. The most important of these is the aforementioned pigeon hole principle. In addition, lemma count_complement was moved from Shankar's module ica3 to countmod. Shankar's complete proof was re-run after the changes to ensure that nothing was inadvertently destroyed. Future efforts will likely require additional modifications to Shankar's modules.

The induction modules, natinduction and noetherian, were taken from Rushby's transient recovery verification [6]. The standard induction schema was modified to syntactically match that used by Shankar. In addition, a lemma was added for complete induction over the natural numbers. The remaining modules were generated in the course of this verification.

The appendix contains the proof chain analysis for the three properties stated above. The proof for translation invariance is in module mid, precision enhancement is in mid3, and accuracy preservation is in mid4.

5 Proposed Verification

This section describes the proposed verification that the circuit correctly implements the convergence function. First an informal description of the circuit is given, and then the verification plan is discussed. This design assumes that the network of clocks is completely connected.

5.1 Informal Description

As in other synchronization algorithms, this one consists of an infinite sequence of synchronization intervals of duration $\approx R$. For the time being, we will presume the constraints listed above. It is assumed that all good clocks know the index of the current interval (a simple counter is sufficient, provided that all good channels start the counter in the same interval). The major concern is when to begin the next interval. For this we require readings of the other clocks in the system, and a suitable convergence function. As stated above, the selected convergence function is the fault-tolerant midpoint.

In order to execute the convergence function to start the $(i+1)$th interval clock, we need an estimate of the other processes clocks when local time is
\[(i + 1)R \text{ (according to } I C_p^i(t))\]. Our estimate, \(\Theta_p^{i+1}\), of other clocks is

\[\Theta_p^{i+1}(q) = (i + 1)R + (Q - LC_p^i(t_{pq}))\]

where \(t_{pq}\) is the time that \(p\) receives the signal from \(q\), and \(LC\) is a local counter measuring elapsed time since the beginning of the current interval. All clocks participating in the protocol know to send their signal when \(LC_p^i(t) = Q\). The value \((Q - LC_p^i(t_{pq}))\) gives the difference between when the local clock expected the signal and when it observed a signal from \(q\). The reading is taken in such a way, that simply adding the value to the current time gives an estimate of the other processors clocks at that instant (modulo any effects from drift).

If the local processor \(p\) reads its clock at time \(t\) it will receive the pair \((i, LC_p^i(t))\). This reading gives the duration of time since the beginning of the protocol. The correct interpretation is \(VC_p^i(t) = iR + LC_p^i(t)\). Thus the reading of the virtual clock just before \(p\) resets its registers for the \(i\)th interval will be \(iR + cfn_{MID}(p, (\lambda Q, \Theta_p^i(q) - iR))\). Notice that translation invariance allows the computation of the convergence function based solely on \((\lambda Q - LC_p^i(t_{pq}))\).

Figure 3 presents an informal block model of the proposed clock synchronization circuit. The circuit consists of the following components:

- \(N\) pulse recognizers (only one pulse per clock is recognized in any given interval),
- a pulse counter (triggers events based upon pulse arrivals),
- a local counter \(LC\) (measures elapsed time since beginning of current interval),
- an interval counter (contains the index \(i\) of the current interval),
- one adder for computing the value \(-(Q - LC_p^i(t_{pq}))\),
- one register each for storing \(-\theta_{(F+1)}\) and \(-\theta_{(N-F)}\),
- an adder for computing the sum of these two registers, and
- a divide-by-2 component.

The pulses are already sorted by arrival time, so it is natural to use a pulse counter to select the time-stamp of the \((F + 1)\)th and the \((N - F)\)th pulses for the computation of the convergence function. As stated previously, all that is required is the difference between the local and remote clocks. Let \(\theta = (\lambda Q, \Theta_p^{i+1}(q) - (i + 1)R)\). When the \((F + 1)\)th \((N - F)\)th signal is observed, register \(-\theta_{(F+1)}\) \((-\theta_{(N-F)})\) is clocked, saving the value \(-(Q - LC_p^i(t))\). After \(N - F\) signals have been observed, the multiplexer selects the computed
Figure 3: Informal Block Model
convergence function instead of \( Q \). When \( LC'(t) - (-c\mu_{MID}(p, \theta)) = R \) it is time to begin the \( i + 1 \)st interval. To do this, all that is required is to increment \( i \) and reset \( LC \) to 0. The pulse recognizers, multiplexer select and registers are also reset at this time.

5.2 Correctness Criteria

First, the RTL description will be entered in the E\texttt{HDM} specification language, and then \texttt{EHDM} will be used to prove that RTL description correctly implements \( c\mu_{MID}(p, \theta) \). Each block in the informal model will be decomposed into normal hardware components such as registers, arithmetic logic units, multiplexors, and standard logic components. A functional description will be given for each device, and their composition will be shown to implement the fault-tolerant mid-point convergence function. This part of the verification will assume the properties of read error, bounded drift, and initial synchronization. Any assumptions about the convergence function used in the proofs of translation invariance, precision enhancement, or accuracy preservation need to be discharged at this level.

6 Transient Recovery

The argument for transient recovery capabilities hinges upon the following observation:

\textit{As long is there is power to the circuit and no faults are present, the circuit will execute the algorithm.}

Using the fact that the algorithm executes continually, and that pulses can be observed during the entire synchronization interval, we can establish that up to \( F \) transiently affected channels will automatically reintegrate themselves into the set of good channels.

We will break the discussion down into cases: First, the simple case when \( F = 1 \), and then the more general case for \( F > 1 \). Remember that \( N \geq 3F + 1 \). The reason two cases are considered is that only a simple modification to the hardware is required to guarantee reintegration when \( F = 1 \); the more general case require more inventive techniques.
6.1 Single Fault Scenario

The only modification required is that the synchronization signals include the sender's value for \( i \) (the index for the current sync interval). By virtue of the maintenance algorithm the \( N - 1 \) good clocks are synchronized within a bounded skew \( \delta \ll R \). Suppose the recovering clock observes \( N - 1 \) pulses within \( \delta + 2\Lambda \); it will choose two of these good values for computing the convergence function and a simple vote of the received interval indices will restore correct time to a lost process.

There is a possibility that the readings from the good clocks will straddle the frame boundary. The recovering clock will be ignored in the computations of the good channel, and it should adjust its own clock such that in its next interval, it will see all of the good clocks. If the window is symmetric (i.e. \( Q = R/2 \)), it is possible that the recovering channel will compute no correction and will remain unsynchronized. However, if the window is asymmetric, a split at the boundaries will cause a recovering process to compute sufficient correction to push it into a region where it will see all the good clocks in the same interval. Thus, \( Q \) should be selected so that the window is asymmetric (i.e. \( Q \neq R/2 \)).

6.2 General Case

When \( F \geq 2 \) the problem becomes more complicated. As above, if the recovering clock observes \( N - F \) pulses within \( \delta + 2\Lambda \), it will restore its synchrony via the convergence function and a vote of the received interval indices. However, if the good clocks straddle the boundary, the additional faulty clock(s) can prevent any adjustment from being computed on the recovering clock. It is likely that recovery cannot be guaranteed unless a timeout mechanism is added.

6.3 Comparison with Other Approaches

A number of other fault-tolerant clock synchronization protocols allow for restoration of a lost clock. The approach taken here is very similar to that proposed by Welch and Lynch [11]. They propose that when a process awakens, that it observe incoming messages until it can determine which round is underway, and then wait sufficiently long to ensure that it has seen all valid messages in that round. It can then compute the necessary correction to become synchronized. Srikanth and Toueg [12] use a similar approach, modified to the context of their algorithm. Halpern et al. [13] suggest a rather
complicated protocol which requires explicit cooperation of other clocks in the system. It is more appropriate when the number of clocks in the system varies greatly over time. All of these approaches have the common theme, namely, that the joining processor knows that it wants to join. This implies the presence of some diagnostic logic or timeout mechanism which triggers the recovery process. The approach suggested here happens automatically. By virtue of the algorithm’s execution in dedicated hardware, there is no need to awaken a process to participate in the protocol. The main idea is for the recovering process to converge to a state where it will observe all other clocks in the same interval, and then to restore the correct interval counter.

7 Initial Synchronization

If we can get into a state which satisfies the requirements for precision enhancement:

Given any subset $C$ of the $N$ clocks with $|C| \geq N - F$, and clocks $p$ and $q$ in $C$, then for any readings $\gamma$ and $\theta$ satisfying the conditions

1. for any $l$ in $C$, $|\gamma(l) - \theta(l)| \leq x$
2. for any $l, m$ in $C$, $|\gamma(l) - \gamma(m)| \leq y$
3. for any $l, m$ in $C$, $|\theta(l) - \theta(m)| \leq y$

there is a bound $\pi(x, y)$ such that

$|\text{cfn}(p, \gamma) - \text{cfn}(q, \theta)| \leq \pi(x, y)$

where $y = R/2$ and $x$ is the normal value ($\approx 2\Lambda$), the above circuit will converge to within $\delta_2$ in approximately $\log_2(R/2)$ intervals. Byzantine agreement will then be required to establish a consistent interval counter. It will be necessary to ensure that the clocks converge to a state satisfying the above constraints.

7.1 Mechanisms for Initialization

In order to ensure that we reach a state which satisfies the above requirements, it is necessary to identify possible states which violate the above requirements. Such states would happen due to the behavior of clocks prior to the time that enough good clocks are running. In previous cases we knew
we had a set $C$ of good clocks with $|C| \geq N - F$. This means that there were a sufficient number of clock readings to resolve $\theta_{(F+1)}$ and $\theta_{(N-F)}$. This may not be the case during initialization. We need to determine a course of action when we do not observe $N - F$ clocks. Two plausible options are to

1. pretend all clocks are observed to be in perfect synchrony, or

2. pretend that unobserved clocks are observed at the end of the interval (i.e. $(LC_p(t_{pq}) - Q) = (R - Q)$). Compute the correction based upon this value.

Both options will be explored. The first option is simple to implement because no correction is necessary. When $LC = R$, set both $i$ and $LC$ to 0, and reset the circuit for the next interval. To implement the second option, perform the following action when $LC = R$: if fewer than $N - F (F + 1)$ signals are observed, then enable register $-\theta_{(N-F)} (-\theta_{(F+1)})$. This will cause the unobserved readings to be $(R - Q)$ which is equivalent to observing the pulse at the end of an interval of duration $R$.

It will be necessary to define a convergence stair (ala [15]) for scenarios that don’t converge by default.

### 7.2 Comparison to Other Approaches

Most of the comments concerning the approach to transient recovery are applicable here as well. This approach for achieving initial synchronization differs from most methods in that it first synchronizes the interval clocks, and then it decides upon a value for the current interval. Techniques in [11], [12], and [13] all depend upon the good clocks knowing that they wish to initialize. Agreement is reached among the clocks wishing to join, and then the protocol begins. The approach taken here seems closer to that used in [14], however, details of their approach are not given.

### 8 Concluding Remarks

Clock synchronization provides the cornerstone of any fault-tolerant computer architecture. To avoid a single point failure it is imperative that each processor maintain a local clock which is periodically resynchronized with other clocks in a fault-tolerant manner. Due to subtleties involved in reasoning about interactions involving misbehaving components, it is necessary to prove that the clock synchronization function operates correctly. Shankar
[2] provides a mechanical proof (using EHDM [3]) that Schneider's generalized protocol [1] achieves Byzantine fault-tolerant clock synchronization, provided that eleven constraints are satisfied. Shankar's work provides the formal specification of the proposed verified design.

The fault-tolerant midpoint convergence function has been proven (in EHDM) to satisfy the properties of translation invariance, precision enhancement, and accuracy preservation. These proofs are reusable in the verification of any synchronization algorithm which uses the same function. An informal design of a circuit to implement this function has been presented. Future efforts will focus on formalizing this design and proving the additional required properties from it. A register transfer level description of the design will be expressed in the specification language of EHDM, and proven to correctly implement the fault-tolerant midpoint function. Other properties to be proven from the design include bounded interval, bounded delay, initial synchronization, non-overlap, and any assumptions made in establishing the properties of the convergence function. Bounded drift is a physical property of the oscillator and cannot be established formally. The value for drift will be taken from the oscillator's stated performance parameters. It is assumed that the number of faults F is less than N/3, where N denotes the number of clocks in the system. Read error will be assumed in this development, but there is ongoing work at SRI to prove that remote clocks can be read with bounded error. An approach for bounding initial skew will be verified for the single fault scenario and a more general solution will be explored.

In keeping with the spirit of the Reliable Computing Platform, it is imperative that the clock synchronization subsystem provide for recovery from transient faults. This paper has argued that the proposed design will recover from a single transient fault. This argument will be formalized in EHDM using an approach similar to that used by DiVito, Butler, and Caldwell for the RCP [4]. Extensions to accommodate the more general case will be developed, but would likely involve modifications to the design. An interesting feature of this design is that for the single fault case (i.e. 4, 5, or 6 clocks), the properties of transient recovery and initial synchronization occur automatically. The clock system will recover without explicitly recognizing that something is amiss. The system can be augmented to recognize loss of synchrony due to a transient fault, but need not do so for recovery purposes.
A Proof Summary

Notice that the only modules with failed proofs have the suffix _tcc. These modules are automatically generated by EHDM and cannot be altered by the user. When a proof fails the user must prove the type check constraint elsewhere. The proof chain analysis (Appendix C) ensures that these obligations have been discharged.

Proof summaries for modules on using chain of module mid_top

<table>
<thead>
<tr>
<th>Module</th>
<th>Successful proofs</th>
<th>Failures</th>
<th>Errors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module mid4_tcc</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Module mid3_tcc</td>
<td>8</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>Module mid2_tcc</td>
<td>2</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>Module mid_tcc</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Module tcc_mid</td>
<td>9</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Module division_tcc</td>
<td>7</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Module natinduction_tcc</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Module countmod_tcc</td>
<td>3</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>Module ft_mid_assume</td>
<td>no proofs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Module clocksort</td>
<td>no proofs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Module select_defs</td>
<td>6</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Module mid</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Module mid2</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Module mid3</td>
<td>9</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Module noetherian</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Module natinduction</td>
<td>5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Module countmod</td>
<td>30</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Module clockassumptions</td>
<td>9</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Module absmod</td>
<td>15</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Module division</td>
<td>11</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Module multiplication</td>
<td>11</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Module arith</td>
<td>no proofs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Module mid4</td>
<td>9</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Module mid_top</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Totals: 146 successful proofs, 12 failures, 0 errors

Total time: 715 seconds.
B \textsc{LaTeX} printed EHDM Modules

mid_top: Module

Using mid4. countmod_tcc, natinduction_tcc, division_tcc, tcc_mid

Theory

Proof

posint_TCC1.PROOF: Prove posint_TCC1 \{i_1 - 1\}

countmod_TCC4_pr: Prove count_TCC4 from
\hspace{1cm} countsize,
\hspace{1cm} countsize \{i - ( if i > 0 then i - 1 else i end if)\}

countmod_TCC5.pr: Prove count_TCC5 from
\hspace{1cm} countsize,
\hspace{1cm} countsize \{i - ( if i > 0 then i - 1 else i end if)\}

End mid_top
countmod.tcc: Module

Using countmod

Exporting all with countmod

Theory

\( i_1: \text{Var integer} \)
\( \text{ppred: Var function[naturalnumber — boolean]} \)
\( i: \text{Var naturalnumber} \)
\( p: \text{Var naturalnumber} \)
\( k: \text{Var naturalnumber} \)
\( n: \text{Var naturalnumber} \)
\( d_1: \text{Var nk.type} \)
\( nk: \text{Var nk.type} \)
\( nk2: \text{Var nk.type} \)
\( j: \text{Var naturalnumber} \)

\text{posint.TCC1: Formula} (\exists i_1 : i_1 > 0)

\text{count.TCC1: Formula} (i > 0) \supset (i - 1 \geq 0)

\text{count.TCC2: Formula} (\text{ppred}(i - 1)) \land (i > 0) \supset (i - 1 \geq 0)

\text{count.TCC3: Formula} \neg(\text{ppred}(i - 1)) \land (i > 0) \supset (i - 1 \geq 0)

\text{count.TCC4: Formula}
\((\text{ppred}(i - 1)) \land (i > 0) \supset \text{countsize(ppred, i)} > \text{countsize(ppred, i - 1)}\)

\text{count.TCC5: Formula}
\((\neg(\text{ppred}(i - 1))) \land (i > 0) \supset \text{countsize(ppred, i)} > \text{countsize(ppred, i - 1)}\)

Proof

\text{posint.TCC1.PROOF: Prove posint.TCC1}
\text{count.TCC1.PROOF: Prove count.TCC1}
\text{count.TCC2.PROOF: Prove count.TCC2}
count.TCC3.PROOF: Prove count.TCC3

count.TCC4.PROOF: Prove count.TCC4

count.TCC5.PROOF: Prove count.TCC5

End countmod.tcc
natinduction_tcc: Module

Using natinduction

Exporting all with natinduction

Theory

m: Var naturalnumber
n: Var naturalnumber
i: Var naturalnumber
j: Var naturalnumber
d1: Var naturalnumber

ind_m_proof_TCC1: Formula
(in m then n - m else 0 end if ≥ 0)

Proof

ind_m_proof_TCC1.PROOF: Prove ind_m_proof_TCC1

End natinduction_tcc
division_tcc: Module

Using division

Exporting all with division

Theory

\(x\): Var number
\(y\): Var number
\(z\): Var number

mult_div_1_TCC1: Formula \((z \neq 0) \supset (z \neq 0)\)

mult_div_TCC1: Formula \((y \neq 0) \supset (y \neq 0)\)

div_cancel_TCC1: Formula \((x \neq 0) \supset (x \neq 0)\)

ceil_mult_div_TCC1: Formula \((y > 0) \supset (y \neq 0)\)

div_nonnegative_TCC1: Formula \((x \geq 0 \land y > 0) \supset (y \neq 0)\)

div_ineq_TCC1: Formula \((z > 0 \land x \leq y) \supset (z \neq 0)\)

div_minus_1_TCC1: Formula \((y > 0 \land x < 0) \supset (y \neq 0)\)

Proof

mult_div_1_TCC1_PROOF: Prove mult_div_1_TCC1

mult_div_TCC1_PROOF: Prove mult_div_TCC1

div_cancel_TCC1_PROOF: Prove div_cancel_TCC1

ceil_mult_div_TCC1_PROOF: Prove ceil_mult_div_TCC1

div_nonnegative_TCC1_PROOF: Prove div_nonnegative_TCC1

div_ineq_TCC1_PROOF: Prove div_ineq_TCC1

div_minus_1_TCC1_PROOF: Prove div_minus_1_TCC1

End division_tcc
tcc_mid: Module

Using mid.tcc mid2.tcc mid3.tcc mid4.tcc

Theory

Proof

ft_mid_TCC2_PROOF: Prove ft_mid_TCC2 from ft_mid_maxfaults

good_less_NF_TCC1_PROOF: Prove good_less_NF_TCC1 from
    ft_mid_maxfaults

good_less_NF_pr_TCC1_PROOF: Prove good_less_NF_pr_TCC1 from
    ft_mid_maxfaults

good_between_TCC1_PROOF: Prove good_between_TCC1 from
    ft_mid_maxfaults

ft_mid_prec_sym1_TCC2_PROOF: Prove ft_mid_prec_sym1_TCC2 from
    ft_mid_maxfaults

ft_mid_prec_sym1_TCC4_PROOF: Prove ft_mid_prec_sym1_TCC4 from
    ft_mid_maxfaults

mid_gt_imp_sel_gt_TCC2_PROOF: Prove mid_gt_imp_sel_gt_TCC2 from
    ft_mid_maxfaults

ft_mid_prec_sym1_pr_TCC2_PROOF: Prove ft_mid_prec_sym1_pr_TCC2 from
    ft_mid_maxfaults

ft_mid_greater_TCC1_PROOF: Prove ft_mid_greater_TCC1 from
    ft_mid_maxfaults

End tcc_mid
absmod: Module

Using multiplication

Exporting all

Theory

\[.x, y, z, x_1, y_1, z_1, x_2, y_2, z_2: \text{Var number}\]

[\# | 1]: Definition function[number — number] =
( \lambda x : ( \text{if } x < 0 \text{ then } -x \text{ else } x \text{ end if})

abs_main: Lemma \(|x| < z \land (x < z \lor -x < z)\)

abs_leq.0: Lemma \(|x - y| \leq z \land (x - y) \leq z\)

abs.diff: Lemma \(|x - y| < z \land ((x - y) < z \lor (y - x) < z)\)

abs_leq: Lemma \(|x| \leq z \land (x \leq z \lor -x \leq z)\)

abs_bnd: Lemma
\[0 \leq z \land 0 \leq x \land x \leq z \land 0 \leq y \land y \leq z \land |x - y| \leq z\]

abs_1_bnd: Lemma \(|x - y| \leq z \land x \leq y + z\)

abs_2_bnd: Lemma \(|x - y| \leq z \land x \geq y - z\)

abs_3_bnd: Lemma \(x \leq y + z \land x \geq y - z \land |x - y| \leq z\)

abs_drift: Lemma
\[|x - y| \leq z \land |x_1 - x| \leq z_1 \lor |x_1 - y| \leq z + z_1\]

abs_com: Lemma \(|x - y| = |y - x|\)

abs.drift_2: Lemma
\[|x - y| \leq z \land |x_1 - x| \leq z_1 \land |y - y_1| \leq z_2 \land \]
\[|x_1 - y_1| \leq z + z_1 + z_2\]

abs_geq: Lemma \(x \geq y \land y \geq 0 \lor |x| \geq |y|\)

abs_ge0: Lemma \(x \geq 0 \lor |x| = x\)

abs_plus: Lemma \(|x + y| \leq |x| + |y|\)
abs_diff_3: Lemma \(x - y \leq z \land y - x \leq z \supset |x - y| \leq z\)

Proof

abs_plus_pr: Prove abs_plus from
\(|*1| \{x - x + y\}, |*1|, |*1| \{x - y\}\)

abs_diff_3_pr: Prove abs_diff_3 from \(|*1| \{x - x - y\}\)

abs_ge0_proof: Prove abs_ge0 from \(|*1|\)

abs_geq_proof: Prove abs_geq from \(|*1|, |*1| \{x - y\}\)

abs_drift_2_proof: Prove abs_drift_2 from
abs_drift,
abs_drift
\(\{x - y, y - y_1, z - z_2, z_1 - z + z_1\}\),
abs_com \(\{x - y_1\}\)

abs_com_proof: Prove abs_com from
\(|*1| \{x - (x - y)\}, |*1| \{(x - (y - x))\}\)

abs_drift_proof: Prove abs_drift from
abs_1_bnd,
abs_1_bnd \(\{x - x_1, y - x, z - z_1\}\),
abs_2_bnd,
abs_2_bnd \(\{x - x_1, y - x, z - z_1\}\),
abs_3_bnd \(\{x - x_1, z - z + z_1\}\)

abs_3_bnd_proof: Prove abs_3_bnd from \(|*1| \{(x - (y - x))\}\)

abs_main_proof: Prove abs_main from \(|*1|\)

abs_leq_0_proof: Prove abs_leq_0 from \(|*1| \{x - x - y\}\)

abs_diff_proof: Prove abs_diff from \(|*1| \{x - (x - y)\}\)

abs_leq_proof: Prove abs_leq from \(|*1|\)
abs.bnd.proof: Prove abs.bnd from \(| x - (x - y) |\)

abs.1.bnd.proof: Prove abs.1.bnd from \(| x - (x - y) |\)

abs.2.bnd.proof: Prove abs.2.bnd from \(| x - (x - y) |\)

End absmod
multiplication: Module

Exporting all

Theory

\[ x, y, z, x_1, y_1, z_1, x_2, y_2, z_2: \text{Var number} \]
\[ \ast 1 \ast 2: \text{function[number, number number]} = (\lambda x, y: (x \ast y)) \]

\textbf{mult.\ldist}: Lemma \( x \ast (y + z) = x \ast y + x \ast z \)

\textbf{mult.\ldist.\minus}: Lemma \( x \ast (y - z) = x \ast y - x \ast z \)

\textbf{mult.rident}: Lemma \( x \ast 1 = x \)

\textbf{mult.lident}: Lemma \( 1 \ast x = x \)

\textbf{distrib}: Lemma \( (x + y) \ast z = x \ast z + y \ast z \)

\textbf{distrib.\minus}: Lemma \( (x - y) \ast z = x \ast z - y \ast z \)

\textbf{mult.non.neg}: Axiom
\[ ((x \geq 0 \land y \geq 0) \lor (x \leq 0 \land y \leq 0)) \Rightarrow x \ast y \geq 0 \]

\textbf{mult.pos}: Axiom \((x > 0 \land y > 0) \lor (x < 0 \land y < 0)) \Rightarrow x \ast y > 0 \)

\textbf{mult.com}: Lemma \( x \ast y = y \ast x \)

\textbf{pos.product}: Lemma \( x \geq 0 \land y \geq 0 \Rightarrow x \ast y \geq 0 \)

\textbf{mult.leq}: Lemma \( z \geq 0 \land x \geq y \Rightarrow x \ast z \geq y \ast z \)

\textbf{mult.leq.2}: Lemma \( z \geq 0 \land x \geq y \Rightarrow x \ast z \geq z \ast y \)

\textbf{mult.10}: Axiom \( 0 \ast x = 0 \)

\textbf{mult.gt}: Lemma \( z > 0 \land x > y \Rightarrow x \ast z > y \ast z \)

\textbf{Proof}

\textbf{mult.gt.pr}: Prove \textbf{mult.gt} from
- \textbf{mult.pos} \{\( x - x - y, y - z \)}, \textbf{distrib.\minus}
distrib_minus_pr: Prove distrib_minus from
mult_ldistrib_minus \{x - z, y - x, z - y\},
mult_com \{x - x - y, y - z\},
mult_com \{y - z\},
mult_com \{x - y, y - z\}

mult_leq_2_pr: Prove mult_leq_2 from
mult_ldistrib_minus \{x - z, y - x, z - y\},
mult_non_neg \{x - z, y - x - y\}

mult_leq_pr: Prove mult_leq from
distrib_minus, mult_non_neg \{x - x - y, y - z\}

mult_com_pr: Prove mult_com from
*1 **2, *1 **2 \{x - y, y - x\}

pos_product_pr: Prove pos_product from mult_non_neg

mult_rident_proof: Prove mult_rident from *1 **2 \{y - 1\}

mult_lident_proof: Prove mult_lident from
*1 **2 \{x - 1, y - x\}

distrib_proof: Prove distrib from
*1 **2 \{x - x + y, y - z\},
*1 **2 \{y - z\},
*1 **2 \{x - y, y - z\}

mult_ldistrib_proof: Prove mult_ldistrib from
*1 **2 \{y - y + z, x - x\}, *1 **2, *1 **2 \{y - z\}

mult_ldistrib_minus_proof: Prove mult_ldistrib_minus from
*1 **2 \{y - y - z, x - x\}, *1 **2, *1 **2 \{y - z\}

End multiplication
noetherian: Module [dom: Type, <: function[dom, dom — bool]]

Assuming

measure: Var function[dom — nat]
a, b: Var dom

well_founded: Formula
(∃ measure: a < b ⊃ measure(a) < measure(b))

Theory

p. A, B: Var function[dom — bool]  
d, d1, d2: Var dom

general_induction: Axiom
(∀ d1: (∀ d2: d2 < d1 ⊃ p(d2)) ⊃ p(d1)) ⊃ (∀ d: p(d))

d3, d4: Var dom

mod_induction: Theorem
(∀ d3, d4: d4 < d3 ⊃ A(d3) ⊃ A(d4))
∧ (∀ d1: (∀ d2: d2 < d1 ⊃ (A(d1) ∧ B(d2))) ⊃ B(d1))
⊂ (∀ d: A(d) ⊃ B(d))

Proof

mod_proof: Prove
mod_induction {d1 — d1®p1, d3 — d1®p1, d4 — d2}
from general_induction {p — (λ d: A(d) ⊃ B(d))}

End noetherian
select_def: Module

Using arith. countmod. clockassumptions. clocksort

Exporting all with clockassumptions

Theory

process: Type is nat
Clocktime: Type is number
l, m, n, p, q: Var process
ψ: Var function[process — Clocktime]
i, j, k: Var posint
T, X, Y, Z: Var Clocktime
*1(+2): function[function[process — Clocktime]. posint — Clocktime] == ( λ ψ, i : ψ(funsort(ψ)(i)))

select_trans_inv: Lemma
k ≤ N ⊃ ( λ ψ : ψ(q) + X) = ψ(k) + X

select_exists1: Lemma i ≤ N ⊃ ( ∃ p : p < N ∧ ψ(p) = ψ(i))

select_exists2: Lemma p < N ⊃ ( ∃ i : i ≤ N ∧ ψ(p) = ψ(i))

select_ax: Lemma 1 ≤ i ∧ i < k ∧ k ≤ N ⊃ ψ(i) ≥ ψ(k)

count_geq_select: Lemma
k ≤ N ⊃ count(( λ p : ψ(p) ≥ ψ(k)). N) ≥ k

count_leq_select: Lemma
k ≤ N ⊃ count(( λ p : ψ(k) ≥ ψ(p)). N) ≥ N - k + 1

Proof

select_trans_inv_pr: Prove select_trans_inv from
funsort_trans_inv

select_exists1_pr: Prove select_exists1 {p — funsort(ψ)(i)} from funsort_fun_1.1 {j — i}

select_exists2_pr: Prove select_exists2 {i — i@p1} from
funsort_fun_onto

35
select_ax_pr: Prove select_ax from funsort_ax \{i \preceq k, j \preceq k\}

count_leq_select_pr: Prove count_leq_select from cnt_sort_leq

count_geq_select_pr: Prove count_geq_select from cnt_sort_geq

End select_defs
ft_mid_assume: Module

Using clockassumptions

Exporting all with clockassumptions

Theory

ft_mid_maxfaults: Axiom $N \geq 3 \cdot F + 1$

End ft_mid_assume
arith: Module
Using multiplication. division. absmod
Exporting all with multiplication. division. absmod
End arith
clocksort: Module

Using clockassumptions

Exporting all with clockassumptions

Theory

{l,m,n,p,q: Var process
i,j,k: Var posint
X,Y: Var Clocktime
v: Var function[process — Clocktime]
funsort: function[function[process — Clocktime]
— function[posint — process]]

funsort.ax: Axiom
i \leq j \land j \leq N \supset v(funsort(v)(i)) \geq v(funsort(v)(j))

funsort.fun_1_1: Axiom
i \leq N \land j \leq N \land funsort(v)(i) = funsort(v)(j)
\supset i = j \land funsort(v)(i) < N

funsort.fun_onto: Axiom
p < N \supset (\exists i: i \leq N \land funsort(v)(i) = p)

funsort.trans_inv: Axiom
k \leq N \supset (\forall q: v(q) + X)(k) = v(funsort(v)(k))

cnt.sort.geq: Axiom
k \leq N \supset \text{count}((\lambda p: v(p) \geq v(funsort(v)(k))), N) \geq k

cnt.sort.leq: Axiom
k \leq N \supset \text{count}((\lambda p: v(funsort(v)(k)) \geq v(p)), N) \geq N - k + 1

Proof

End clocksort
clockassumptions: Module

Using arith.countmod

Exporting all with countmod.arith

Theory

$N$: nat

$N.0$: Axiom $N > 0$

process: Type is nat
event: Type is nat
time: Type is number
Clocktime: Type is number
$m, n, p, q, p_1, p_2, q_1, q_2, p_3, q_3$: Var process
$i, j, k$: Var event
$x, y, z, r, s, t$: Var time
$X, Y, Z, R, S, T$: Var Clocktime
$\gamma, \theta$: Var function[process — Clocktime]
$\delta, \mu, \rho, r_{min}, r_{max}, \beta, \lambda$: number
$P C_{+1}(x+2), V C_{+1}(x+2)$: function[process.time — Clocktime]
$t_{+2}^{+2}$: function[process.event — time]
$\Theta_{+2}^{+2}$: function[process.event

$— function[process — Clocktime]]$
$IC_{+2}(x+3)$: function[process.event.time — Clocktime]
correct: function[process.time — bool]
em cfn: function[process.function[process — Clocktime]

$— Clocktime]
$\pi$: function[Clocktime, Clocktime — Clocktime]
$\alpha$: function[Clocktime — Clocktime]

delta.0: Axiom $\delta \geq 0$
mu.0: Axiom $\mu \geq 0$
rho.0: Axiom $\rho \geq 0$
rho.1: Axiom $\rho < 1$
rmin.0: Axiom $r_{min} > 0$

40
rmax.0: Axiom \( r_{max} > 0 \)

beta.0: Axiom \( \beta > 0 \)

lamb.0: Axiom \( \Lambda \geq 0 \)

init: Axiom correct\( (p, 0) \supset PC_{p}(0) \geq 0 \land PC_{p}(0) \leq \mu \)

correct.closed: Axiom \( s \geq t \land \) correct\( (p, s) \supset \) correct\( (p, t) \)

rate.1: Axiom
\[
\text{correct}(p, s) \land s \geq t \supset PC_{p}(s) - PC_{p}(t) \leq (s - t) \star (1 + \rho)
\]

rate.2: Axiom
\[
\text{correct}(p, s) \land s \geq t \supset PC_{p}(s) - PC_{p}(t) \geq (s - t) \star (1 - \rho)
\]

rts0: Axiom correct\( (p, t) \land t \leq t_{p}^{i+1} \supset t - t_{p}^{i} \leq r_{max}
\)

rts1: Axiom correct\( (p, t) \land t \geq t_{p}^{i+1} \supset t - t_{p}^{i} \geq r_{min}
\)

rts.0: Lemma correct\( (p, t_{p}^{i+1}) \supset t_{p}^{i+1} - t_{p}^{i} \leq r_{max}
\)

rts.1: Lemma correct\( (p, t_{p}^{i+1}) \supset t_{p}^{i+1} - t_{p}^{i} \geq r_{min}
\)

rts2: Axiom
\[
\text{correct}(p, t) \land t \geq t_{q}^{i} + \beta \land \text{correct}(q, t) \supset t \geq t_{q}^{i}
\]

rts.2: Axiom
\[
\text{correct}(p, t_{p}^{i}) \land \text{correct}(q, t_{q}^{i}) \supset t_{p}^{i} - t_{q}^{i} \leq \beta
\]

synctime.0: Axiom \( t_{p}^{0} = 0 \)

VClock.defn: Axiom
\[
\text{correct}(p, t) \land t \geq t_{p}^{i} \land t \leq t_{p}^{i+1} \supset V C_{p}(t) = IC_{p}^{i}(t)
\]

Adj: function[process, event — Clocktime] =
\[
(\lambda p, i:\n\quad (\text{if } i > 0 \text{ then } cfu(p, \Theta_{p}^{i}) - PC_{p}(t_{p}^{i}) \text{ else } 0 \text{ end if}))
\]

IClock.defn: Axiom correct\( (p, t) \supset IC_{p}^{i}(t) = PC_{p}(t) + \text{Adj}(p, i)\)
Readererror: Axiom

\( \text{correct}(p, t_p^{i+1}) \land \text{correct}(q, t_q^{i+1}) \)
\( \supset |\Theta_p^{i+1} q| - IA_{t_p^{i+1}} | \leq \Lambda \)

\text{translation_invariance: Axiom}

\( X \geq 0 \supset cfn(p, (\lambda p_1 - \text{Clocktime} : \gamma(p_1) + X)) = cfn(p, \gamma) + X \)

\text{ppred: Var function[process — bool]}
\( F: \text{process} \)
\( \text{okay_Readpred: function[process — Clocktime].} \)
\( \text{Clocktime. function[process — bool] — bool} = \)
\( (\lambda \gamma, Y: \text{ppred} : \)
\( (\forall l, m : \text{ppred}(l) \land \text{ppred}(m) \supset |\gamma(l) - \gamma(m)| \leq Y)) \)
\( \text{okay_pairs: function[process — Clocktime].} \)
\( \text{Clocktime. function[process — Clocktime].} \text{Clocktime.} \)
\( \text{function[process — bool] — bool} = \)
\( (\lambda \gamma, \theta, X: \text{ppred} : \)
\( (\forall p_3 : \text{ppred}(p_3) \supset |\gamma(p_3) - \theta(p_3)| \leq X)) \)

\text{N_maxfaults: Axiom} \ F \leq N \)

\text{precision_enhancement_ax: Axiom}
\( \text{count(ppred, } N) \geq N - F \)
\( \land \text{okay_Readpred}(\gamma, Y: \text{ppred}) \)
\( \land \text{okay_Readpred}(\theta, Y: \text{ppred}) \)
\( \land \text{okay_pairs}(\gamma, \theta, X: \text{ppred}) \land \text{ppred}(p) \land \text{ppred}(q) \)
\( \supset |cfn(p, \gamma) - cfn(q, \theta)| \leq \pi(X, Y) \)

\text{correct_count: Axiom} \ \text{count}((\lambda p : \text{correct}(p,t)), N) \geq N - F \)
okay_Reading: function[function[process — Clocktime].
Clocktime. time — bool] =

(λ γ, Y, t:
   ( ∀ p₁, q₁ :
     correct(p₁, t) ∧ correct(q₁, t) ⊃ |γ(p₁) − γ(q₁)| ≤ Y))

okay_Readvars: function[function[process — Clocktime].
function[process — Clocktime].
Clocktime. Clocktime — bool] =

(λ γ, θ, X, t:
   ( ∀ p₃ : correct(p₃, t) ⊃ |γ(p₃) − θ(p₃)| ≤ X))

okay_Readpred_Reading: Lemma
okay_Reading(γ, Y, t)
   ⊃ okay_Readpred(γ, Y, (λ p : correct(p, t)))

okay_pairs_Readvars: Lemma
okay_Readvars(γ, θ, X, t)
   ⊃ okay_pairs(γ, θ, X, (λ p : correct(p, t)))

precision_enhancement: Lemma
okay_Reading(γ, Y, t₁₊₁)
   ∧ okay_Reading(θ, Y, t₁₊₁)
   ∧ okay_Readvars(γ, θ, X, t₁₊₁)
   ∧ correct(p, t₁₊₁) ∧ correct(q, t₁₊₁)
   ⊃ |cfu(p, γ) − cfu(q, θ)| ≤ π(X, Y)

okay_Reading_defn_lr: Lemma
okay_Reading(γ, Y, t)
   ⊃ ( ∀ p₁, q₁ :
     correct(p₁, t) ∧ correct(q₁, t) ⊃ |γ(p₁) − γ(q₁)| ≤ Y)

okay_Reading_defn_rl: Lemma
( ∀ p₁, q₁ :
     correct(p₁, t) ∧ correct(q₁, t) ⊃ |γ(p₁) − γ(q₁)| ≤ Y)
   ⊃ okay_Reading(γ, Y, t)

okay_Readvars_defn_lr: Lemma
okay_Readvars(γ, θ, X, t)
   ⊃ ( ∀ p₃ : correct(p₃, t) ⊃ |γ(p₃) − θ(p₃)| ≤ X)
okay_Readvars_defn_rhl: Lemma
( ∀ p3 : correct(p3, t) ∨ |γ(p3) − θ(p3)| ≤ X )
⇒ okay_Readvars(γ, θ, X, t)

accuracy_preservation_ax: Axiom
okay_Readpred(γ, X, ppred)
∧ count(ppred, N) ≥ N − F ∧ ppred(p) ∧ ppred(q)
⇒ |cfn(p, γ) − γ(q)| ≤ α(X)

Proof
okay_Readvars_defn_rhl.pr: Prove
okay_Readvars_defn_rhl {p3 − p3@P1S, q1 − q1@P1S} from okay_Readings

okay_Readvars_defn_lr.pr: Prove okay_Readvars_defn_lr from okay_Readings {p1 − p1@CS, q1 − q1@CS}

proof
okay_Readvars_defn_rhl.pr: Prove
okay_Readvars_defn_rhl {p3 − p3@P1S} from okay_Readvars

okay_Readvars_defn_lr.pr: Prove okay_Readvars_defn_lr from okay_Readvars {p3 − p3@CS}

precision_enhancement_pr: Prove precision_enhancement from precision_enhancement_ax
{ppred − ( λ q : correct(q, t_p+1))},
okay_Readpred_Reading {t − t_p+1},
okay_Readpred_Reading {t − t_p+1, γ − θ},
okay_pairs_Readvars {t − t_p+1},
correct_count {t − t_p+1}

okay_Readpred_Reading_pr: Prove okay_Readpred_Reading from
okay_Readpred {ppred − ( λ p : correct(p, t))},
okay_Readings {p1 − l@P1S, q1 − m@P1S}

okay_pairs_Readvars.pr: Prove okay_pairs_Readvars from
okay_pairs {ppred − ( λ p : correct(p, t))},
okay_Readvars {p3 − p3@P1S}

rts_0_proof: Prove rts_0 from rts0 {t − t_p+1}

44
rts.1_proof: Prove rts.1 from rts \{ t - t_{p+1} \}

End clockassumptions
countmod: Module

Exporting all

Theory

\[ i_1: \text{Var int} \]
\[ \text{posint: Type from nat with } (\lambda i_1 : i_1 > 0) \]
\[ l, m, n, p, q, p_1, p_2, q_1, q_2, p_3, q_3: \text{Var nat} \]
\[ i, j, k: \text{Var nat} \]
\[ x, y, z, r, s, t: \text{Var number} \]
\[ X, Y, Z: \text{Var number} \]
\[ \text{ppred, ppred1, ppred2: Var function[nat -- bool]} \]
\[ v, \theta, \gamma: \text{Var function[nat -- number]} \]
\[ \text{countsize: function[function[nat -- bool].nat -- nat]} = \]
\[ (\lambda \text{ppred}, i : i) \]
\[ \text{count: Recursive function[function[nat -- bool].nat -- nat]} = \]
\[ (\lambda \text{ppred}, i : \]
\[ \text{then} \]
\[ \text{then} \]
\[ \text{else} \]
\[ \text{end if} \]
\[ \text{else 0} \]
\[ \text{end if} \]) \]

by countsize

\[ \text{count_complement: Lemma} \]
\[ \text{count}((\lambda q : \neg \text{ppred}(q)), n) = n - \text{count(ppred. n)} \]

\[ \text{count_exists: Lemma} \]
\[ \text{count(ppred. n)} > 0 \supset (\exists p : p < n \land \text{ppred}(p)) \]

\[ \text{count_true: Lemma} \]
\[ \text{count}((\lambda p : \text{true}), n) = n \]

\[ \text{count_false: Lemma} \]
\[ \text{count}((\lambda p : \text{false}), n) = 0 \]

\[ \text{count_bounded_imp: Lemma} \]
\[ \text{count}((\lambda p : p < n \mid \text{ppred}(p)), n) = \text{count(ppred, n)} \]

\[ \text{count_bounded_and: Lemma} \]
\[ \text{count}((\lambda p : p < n \land \text{ppred}(p)), n) = \text{count(ppred, n)} \]
pigeon_hole: Lemma
\[ \text{count}(\text{pred1}.n) + \text{count}(\text{pred2}.n) \geq n + k \]
\[ \supset \text{count}(\langle \lambda p: \text{pred1}(p) \land \text{pred2}(p) \rangle.n) \geq k \]

pred1, pred2: Var function[nat -- bool]

pred_extensionality: Axiom
\[ (\forall p: \text{pred1}(p) = \text{pred2}(p)) \supset \text{pred1} = \text{pred2} \]

nk_type: Type = Record
\[ \begin{array}{ll}
  n : \text{nat} \\
  k : \text{nat}
\end{array} \\
\text{end record} \]
nk, nk1, nk2: Var nk_type

nk_less: function[nk_type, nk_type -- bool] ==
\[ (\lambda \text{nk1}, \text{nk2}: \text{nk1}.n + \text{nk1}.k < \text{nk2}.n + \text{nk2}.k) \]

Proof

Using natinduction, noetherian

count_bounded_imp0: Lemma
\[ k \geq 0 \supset \text{count}(\langle \lambda p: p < k \supset \text{pred}(p) \rangle.0) = \text{count}(\text{pred}.0) \]

count_bounded_imp_ind: Lemma
\[ (k \geq n \supset \text{count}(\langle \lambda p: p < k \supset \text{pred}(p) \rangle.n)
= \text{count}(\text{pred}.n)) \]
\[ \supset (k \geq n + 1) \]
\[ \supset \text{count}(\langle \lambda p: p < k \supset \text{pred}(p) \rangle.n + 1)
= \text{count}(\text{pred}.n + 1)) \]

count_bounded_imp_k: Lemma
\[ (k \geq n \supset \text{count}(\langle \lambda p: p < k \supset \text{pred}(p) \rangle.n)
= \text{count}(\text{pred}.n)) \]

count_bounded_imp0_pr: Prove count_bounded_imp0 from
\[ \begin{array}{l}
  \text{count} \{i - 0\}, \\
  \text{count} \{\text{pred} \rightarrow \langle \lambda p: p < k \supset \text{pred}(p) \rangle, i - 0\}
\end{array} \]

count_bounded_imp_ind_pr: Prove count_bounded_imp_ind from
\[ \begin{array}{l}
  \text{count} \{i - n + 1\}, \\
  \text{count} \{\text{pred} \rightarrow \langle \lambda p: p < k \supset \text{pred}(p) \rangle, i - n + 1\}
\end{array} \]
count_bounded_imp_k_pr: Prove count_bounded_imp_k from induction
{prop
   \( \lambda n : k \geq n \\) 
   \( \Rightarrow \) count((\( \lambda p : p < k \land ppred(p) \)), n) 
   = count(ppred, n)),
   i - n},
count_bounded_imp0,
count_bounded_imp_ind \{n \rightarrow \}
count_bounded_imp_pr: Prove count_bounded_imp from count_bounded_imp_k \{k \rightarrow \}
count_bounded_and0: Lemma
\( k \geq 0 \Rightarrow \) count((\( \lambda p : p < k \land ppred(p) \)), 0) = count(ppred, 0)
count_bounded_and_ind: Lemma
\( k \geq n \Rightarrow \) count((\( \lambda p : p < k \land ppred(p) \)), n) = count(ppred, n))
\( \Rightarrow \) count((\( \lambda p : p < k \land ppred(p) \)), n + 1) 
   = count(ppred, n + 1))
count_bounded_and_k: Lemma
\( k \geq n \Rightarrow \) count((\( \lambda p : p < k \land ppred(p) \)), n) = count(ppred, n))
count_bounded_and0_pr: Prove count_bounded_and0 from count \{i \rightarrow \}
count \{ppred \rightarrow (\( \lambda p : p < k \land ppred(p) \)), i \rightarrow \}
count_bounded_and_ind_pr: Prove count_bounded_and_ind from count \{i \rightarrow \}
count \{ppred \rightarrow (\( \lambda p : p < k \land ppred(p) \)), i \rightarrow \}
48
count_bounded_and_k.pr: Prove count_bounded_and_k from induction
{prop
-- ( λ n :
k ≥ n
  ⊢ count(( λ p : p < k ∧ ppred(p)). n)
  = count(ppred. n)),
i - n},
count_bounded_and0,
count_bounded_and_ind \{n = n\}
count_bounded_and.pr: Prove count_bounded_and from count_bounded_and_k \{k = n\}
count_false.pr: Prove count_false from count_true,
count_complement \{ppred = ( λ p : true)\},
pred_extensionality
{pred1 = ( λ p : false)\}
c0: Lemma count(( λ q : ¬ppred(q)), 0) = 0 − count(ppred. 0)
cc0: Lemma
(count(( λ q : ¬ppred(q)). n) = n − count(ppred. n))
  ⊢ (count(( λ q : ¬ppred(q)). n + 1)
  = n + 1 − count(ppred. n + 1))
cc0.pr: Prove cc0 from
count \{ppred = ( λ q : ¬ppred(q)), i = 0\},
count \{i = 0\}
cc.ind.pr: Prove cc.ind from
count \{ppred = ( λ q : ¬ppred(q)), i = n + 1\},
count \{i = n + 1\}
count_complement_pr: Prove count_complement from

induction
{prop
  - (λ n :
      count((λ q : ¬pred(q)), n) = n - count(pred(n)),
    i - n},
cc0,
cc_ind {n - j< i 1}

instance: Module is noetherian[nk_type, nk_less]
nk_measure: function[nk_type — nat] =
  (λ nk1 : nk1.n + nk1.k)
nk_well_founded: Prove well_founded {measure — nk_measure}

nk_ph_pred: function[function[nat — bool],
  function[nat — bool], nk_type — bool] =
  (λ ppred1.ppред2, nk :
    count(ppred1.nk.n) + count(ppred2.nk.n) ≥ nk.n + nk.k
    ⊃ count((λ p : ppred1(p) ∧ ppred2(p)).nk.n) ≥ nk.k)
nk_noeth_pred: function[function[nat — bool],
  function[nat — bool], nk_type
  — bool] =
  (λ ppred1.ppред2, nk1 :
    (∀ nk2 :
      nk.less(nk2, nk1) ⊃ nk_ph_pred(ppred1.ppред2, nk2)))

ph_case1: Lemma
  count((λ p : ppred1(p) ∧ ppred2(p)).pred(n)) ≥ k
  ⊃ count((λ p : ppred1(p) ∧ ppred2(p)).n) ≥ k

ph_case1_pr: Prove ph_case1 from
  count {ppred — (λ p : ppred1(p) ∧ ppred2(p)), i - n}

ph_case2: Lemma
  count(ppred1.pred(n)) + count(ppred2, pred(n)) < pred(n) + k
  ∧ count(ppred1.n) + count(ppred2.n) ≥ n + k
  ∧ count((λ p : ppred1(p) ∧ ppred2(p)).pred(n)) ≥ pred(k)
  ⊃ count((λ p : ppred1(p) ∧ ppred2(p)).n) ≥ k

50
\textbf{ph\_case2a: Lemma}
\[ \text{count}(\text{ppred}_1, \text{pred}(n)) + \text{count}(\text{ppred}_2, \text{pred}(n)) < \text{pred}(n) + k \]
\[ \land \text{count}(\text{ppred}_1, n) + \text{count}(\text{ppred}_2, n) \geq n + k \]
\[ \lor \text{ppred}_1(\text{pred}(n)) \land \text{ppred}_2(\text{pred}(n)) \]

\textbf{ph\_case2b: Lemma}
\[ n > 0 \land k > 0 \]
\[ \land \text{count}(\text{ppred}_1, \text{pred}(n)) + \text{count}(\text{ppred}_2, \text{pred}(n)) \]
\[ < \text{pred}(n) + k \]
\[ \land \text{count}(\text{ppred}_1, n) + \text{count}(\text{ppred}_2, n) \geq n + k \]
\[ \lor \text{count}(\text{ppred}_1, \text{pred}(n)) + \text{count}(\text{ppred}_2, \text{pred}(n)) \]
\[ \geq \text{pred}(n) + \text{pred}(k) \]

\textbf{ph\_case2a\_pr: Prove ph\_case2a from}
\[ \text{count} \{\text{ppred} - \text{ppred}_1, \ i = n \}, \]
\[ \text{count} \{\text{ppred} - \text{ppred}_2, \ i = n \} \]

\textbf{ph\_case2b\_pr: Prove ph\_case2b from}
\[ \text{count} \{\text{ppred} - \text{ppred}_1, \ i = n \}, \]
\[ \text{count} \{\text{ppred} - \text{ppred}_2, \ i = n \} \]

\textbf{ph\_case2\_pr: Prove ph\_case2 from}
\[ \text{count} \{\text{ppred} - (\lambda p : \text{ppred}_1(p) \land \text{ppred}_2(p)), \ i = n \}, \]
\textbf{ph\_case2a} \]

\textbf{ph\_case0: Lemma}
\[ (n = 0 \lor k = 0) \]
\[ \lor (\text{count}(\text{ppred}_1, n) + \text{count}(\text{ppred}_2, n) \geq n + k \]
\[ \lor \text{count}((\lambda p : \text{ppred}_1(p) \land \text{ppred}_2(p)), n) \geq k) \]

\textbf{ph\_case0\_n: Lemma}
\[ (\text{count}(\text{ppred}_1, 0) + \text{count}(\text{ppred}_2, 0) \geq k \]
\[ \lor \text{count}((\lambda p : \text{ppred}_1(p) \land \text{ppred}_2(p)), 0) \geq k) \]

\textbf{ph\_case0\_n\_pr: Prove ph\_case0\_n from}
\[ \text{count} \{\text{ppred} - \text{ppred}_1, \ i = 0 \}, \]
\[ \text{count} \{\text{ppred} - \text{ppred}_2, \ i = 0 \}, \]
\[ \text{count} \{\text{ppred} - (\lambda p : \text{ppred}_1(p) \land \text{ppred}_2(p)), \ i = 0 \} \]

\textbf{ph\_case0\_k: Lemma}
\[ \text{count}((\lambda p : \text{ppred}_1(p) \land \text{ppred}_2(p)), n) \geq 0 \]
ph_case0k_pr: Prove ph_case0k from
  nat_invariant
  \{nat_var \leftarrow \text{count}((\lambda p : \text{ppred1}(p) \land \text{ppred2}(p)), n)\}

ph_case0_pr: Prove ph_case0 from ph_caseOn, ph_caseOk

nk_ph_expand: Lemma
  \(\forall n, k:\)
  \[
  (\text{count}(\text{ppred1}.\text{pred}(n)) + \text{count}(\text{ppred2}.\text{pred}(n))) \\
  \geq \text{pred}(n) + \text{pred}(k) \\
  \lor \text{count}((\lambda p : \text{ppred1}(p) \land \text{ppred2}(p)).\text{pred}(n)) \\
  \geq \text{pred}(k)) \\
  \land (\text{count}(\text{ppred1}.\text{pred}(n)) + \text{count}(\text{ppred2}.\text{pred}(n))) \\
  \geq \text{pred}(n) + k \\
  \lor \text{count}((\lambda p : \text{ppred1}(p) \land \text{ppred2}(p)).\text{pred}(n)) \\
  \geq k) \\
  \lor (\text{count}(\text{ppred1}.n) + \text{count}(\text{ppred2}, n) \geq n + k \\
  \lor \text{count}((\lambda p : \text{ppred1}(p) \land \text{ppred2}(p)), n) \geq k))
  \]

nk_ph_expand_pr: Prove nk_ph_expand from
  ph_case0, ph_case1, ph_case2, ph_case2a, ph_case2b

nk_ph_noeth_hyp: Lemma
  \(\forall nk1:\)
  \[
  \text{nk_noeth_pred}(\text{ppred1}.\text{ppred2}.nk1) \\
  \lor \text{nk_ph_pred}(\text{ppred1}.\text{ppred2}.nk1)
  \]

nk_ph_noeth_hyp_pr: Prove nk_ph_noeth_hyp from
  nk_ph_pred \{nk \rightleftharpoons nk1\},
  nk_noeth_pred \{nk2 \rightleftharpoons nk1\} \text{ with } [(n) := \text{pred}(nk1.n)],
  nk_noeth_pred
  \{nk2 \rightleftharpoons nk1\} \text{ with } [(n) := \text{pred}(nk1.n), (k) := \text{pred}(nk1.k)]},
  nk_ph_pred \{nk \rightleftharpoons nk1\} \text{ with } [(n) := \text{pred}(nk1.n)]},
  nk_ph_pred
  \{nk \rightleftharpoons nk1\} \text{ with } [(n) := \text{pred}(nk1.n), (k) := \text{pred}(nk1.k)]},
  nk_phexpand \{n \rightleftharpoons nk1.n, k \rightleftharpoons nk1.k\},
  ph_case0 \{n \rightleftharpoons nk1.n, k \rightleftharpoons nk1.k\},
  nat_invariant \{nat_var \leftarrow nk1.n\},
  nat_invariant \{nat_var \leftarrow nk1.k\}

52
nk_ph_lem: Lemma nk_ph_pred(ppred1, ppred2, nk)

nk_ph_lem.pr: Prove nk_ph_lem from
  general_induction
  \( \{ p \leftarrow (\lambda nk : nk_ph_pred(ppred1, ppred2, nk)),\)
  \( d_2 \leftarrow nk2 \uparrow \rho1,\)
  \( d \leftarrow nk \uparrow \rho c,\)
  nk_ph_noeth_hyp \{ nk1 \leftarrow d_1 \uparrow \rho p1,\)
  nk_noeth_pred \{ nk1 \leftarrow d_1 \uparrow \rho p1 \}

pigeon_hole.pr: Prove pigeon_hole from
  nk_ph_lem \{ nk \leftarrow nk \text{ with } [(n) := n \uparrow \rho c, (k) := k \uparrow \rho c]\},
  nk_ph_pred \{ nk \leftarrow nk \uparrow \rho p1 \}

exists_less: function\[function[nat \rightarrow bool]. nat \rightarrow bool\] =
  \( (\lambda ppred. n : (\exists p : p < n \land ppred(n)))\)

count_exists_base: Lemma
  count(ppred. 0) > 0 \implies \exists less(ppred. 0)

count_exists_base.pr: Prove count_exists_base from
  count \{ i = 0 \}, exists_less \{ n = 0 \}

count_exists_ind: Lemma
  (count(ppred. n) > 0 \implies \exists less(ppred. n))
  \implies (count(ppred. n + 1) > 0 \implies \exists less(ppred. n + 1))

count_exists_ind.pr: Prove count_exists_ind from
  count \{ i = n + 1 \},
  exists_less,
  exists_less
  \{ n = n + 1,
  p \leftarrow (\text{if} \ ppred(n) \ \text{then} \ n \ \text{else} \ p \uparrow \rho p2 \ \text{end if})\}
count_exists_pr: Prove count_exists \( \mu - \mu^{i\mu p1} \) from induction
\[ \{ \text{prop} \]
\[ - (\lambda n : \text{count}(\text{ppred}. n) > 0 \supset \text{exists_less}(\text{ppred}. n)), \]
\[ i - n^{i\mu c} \} \]
count_exists_base,
count_exists_ind \( \{ n - j^{i\mu p1} \} \),
exists_less \( \{ n - i^{i\mu p1} \} \)

count_base: Sublemma \( \text{count}(\text{ppred}. 0) = 0 \)
count_base_pr: Prove count_base from count \( \{ i = 0 \} \)

count_true_ind: Sublemma
\( (\text{count}(\lambda p : \text{true}). n) = n ) \)
\[ \supset \text{count}(\lambda p : \text{true}). n + 1 = n + 1 \]
count_true_ind_pr: Prove count_true_ind from
\[ \text{count} \{ \text{ppred} - (\lambda p : \text{true}), i - n + 1 \} \]

count_true_pr: Prove count_true from induction
\[ \{ \text{prop} - (\lambda n : \text{count}((\lambda p : \text{true}). n) = n), \]
\[ i - n^{i\mu c} \} \]
count_base \( \{ \text{ppred} - (\lambda p : \text{true}) \} \),
count_true_ind \( \{ n - j^{i\mu p1} \} \)

End countmod
natinduction: Module

Theory

\(i, j, m, m_1, n: \text{Var nat}\)
\(p, \text{prop}: \text{Var function[nat \rightarrow bool]}\)

induction: Theorem
\((\text{prop}(0) \land (\forall j : \text{prop}(j) \lor \text{prop}(j + 1))) \lor \text{prop}(i)\)

complete induction: Theorem
\((\forall i : (\forall j : j < i \lor \text{prop}(j)) \lor \text{prop}(i)) \lor (\forall n : \text{prop}(n))\)

induction_m: Theorem
\(p(m) \land (\forall i : i \geq m \land p(i) \lor p(i + 1))\)
\(\lor (\forall n : n \geq m \lor p(n))\)

limited induction: Theorem
\((m \leq m_1 \lor p(m)) \land (\forall i : i \geq m_1 \land i < m \land p(i) \lor p(i + 1))\)
\(\lor (\forall n : n \geq m \land n \leq m_1 \lor p(n))\)

Proof

Using noetherian

less: function[nat, nat \rightarrow bool] == (\lambda m, n : m < n)

instance: Module is noetherian[nat, less]
x: Var nat
identity: function[nat \rightarrow nat] == (\lambda n : n)

discharge: Prove well_founded \{measure \rightarrow identity\}

complete_ind.pr: Prove complete_induction \{i \rightarrow p1\} from
general_induction \{d \rightarrow n, d_2 \rightarrow j\}

ind.proof: Prove induction \{j \rightarrow \text{pred}(d_1 \cap p1)\} from
general_induction \{p \rightarrow \text{prop}, d \rightarrow i, d_2 \rightarrow j\}
ind.m.proof: Prove induction.m \{i \rightarrow j \& p \& m\} from induction
\{prop \rightarrow (\lambda x: p \& c(x + m)),
i \rightarrow if n \geq m then n - m else 0 end if\}

limited.proof: Prove limited.induction \{i \rightarrow i \& p\} from induction.m \{p \rightarrow (\lambda x: x \leq n \& \& p \& c(x))\}

End natinduction
division: Module

Using multiplication. absmod

Exporting all

Theory

\[ x, y, z, x_1, y_1, z_1, x_2, y_2, z_2: \text{Var number} \]

\[ \text{[\(*1\): function[number - int]} \]

ceil_defn: Axiom \[ [x] \geq x \land [x] - 1 < x \]

mult_div_1: Axiom \[ z \neq 0 \supset x * y/z = x * (y/z) \]

mult_div_2: Axiom \[ z \neq 0 \supset x * y/z = (x/z) * y \]

mult_div_3: Axiom \[ z \neq 0 \supset (z/z) = 1 \]

mult_div: Lemma \[ y \neq 0 \supset (x/y) * y = x \]

div_cancel: Lemma \[ x \neq 0 \supset x * y/x = y \]

div_distrib: Lemma \[ z \neq 0 \supset ((x + y)/z) = (x/z) + (y/z) \]

ceil_mult_div: Lemma \[ y > 0 \supset [x/y] * y \geq x \]

ceil_plus_mult_div: Lemma \[ y > 0 \supset [x/y] + 1 * y > x \]

div_nonnegative: Lemma \[ x \geq 0 \land y > 0 \supset (x/y) \geq 0 \]

div_minus_distrib: Lemma \[ z \neq 0 \supset (x - y)/z = (x/z) - (y/z) \]

div_ineq: Lemma \[ z > 0 \land x \leq y \supset (x/z) \leq (y/z) \]

abs_div: Lemma \[ y > 0 \supset |x/y| = |x|/y \]

mult_minus: Lemma \[ y \neq 0 \supset -(x/y) = (-x/y) \]

div_minus_1: Lemma \[ y > 0 \land x < 0 \supset (x/y) < 0 \]

Proof
div_nonnegative_pr: Prove div_nonnegative from
   mult_non_neg \{x - ( if y \neq 0 then (x/y) else 0 end if)\},
   mult_div

div_distrib_pr: Prove div_distrib from
   mult_div.1 \{x - x + y, y - 1, z - z\},
   mult_rident \{x - x + y\},
   mult_div.1 \{x - x, y - 1, z - z\},
   mult_rident,
   mult_div.1 \{x - y, y - 1, z - z\},
   mult_rident \{x - y\},
   distrib \{z - ( if z \neq 0 then (1/z) else 0 end if)\}

div_cancel_pr: Prove div_cancel from
   mult_div.2 \{z - x\},
   mult_div.3 \{z - x\},
   mult_lident \{x - y\}

mult_div_pr: Prove mult_div from
   mult_div.2 \{z - y\},
   mult_div.1 \{z - y\},
   mult_div.3 \{z - y\},
   mult_rident

abs_div_pr: Prove abs_div from
   \{*1\} \{x - ( if y \neq 0 then (x/y) else 0 end if)\},
   \{*1\},
   div_nonnegative,
   div_minus.1,
   mult_minus

mult_minus_pr: Prove mult_minus from
   mult_div.1 \{x - -1, y - x, z - y\},
   \{*1 **2\} \{x - -1, y - x\},
   \{*1 \*2\}
   \{x - -1, y - ( if y \neq 0 then (x/y) else 1 end if)\}
div_minus_1_pr: Prove div_minus_1 from
  mult_div,
  pos_product
  \{x - (\text{ if } y \neq 0 \text{ then } (x/y) \text{ else } 0 \text{ end if}),
    y - y\}\n
div_minus_distrib_pr: Prove div_minus_distrib from
  div_distrib \{y - y\}, mult_minus \{x - y, y - z\}\n
div_ineq_pr: Prove div_ineq from
  mult_div \{y - z\},
  mult_div \{x - y, y - z\},
  mult_gt
  \{x - (\text{ if } z \neq 0 \text{ then } (x/z) \text{ else } 0 \text{ end if}),
    y - (\text{ if } z \neq 0 \text{ then } (y/z) \text{ else } 0 \text{ end if})\}\n
ceil_plus_mult_div_proof: Prove ceil_plus_mult_div from
  ceil_mult_div,
  distrib
  \{x - [(\text{ if } y \neq 0 \text{ then } (x/y) \text{ else } 0 \text{ end if})],
    y - 1,
    z - y\},
  mult_lident \{x - y\}\n
ceil_mult_div_proof: Prove ceil_mult_div from
  mult_div,
  mult_leq
  \{x - [(\text{ if } y \neq 0 \text{ then } (x/y) \text{ else } 0 \text{ end if})],
    y - (\text{ if } y \neq 0 \text{ then } (x/y) \text{ else } 0 \text{ end if}),
    z - y\},
  ceil_defn \{x - (\text{ if } y \neq 0 \text{ then } (x/y) \text{ else } 0 \text{ end if})\}\n
End division
mid_tcc: Module

Using mid

Exporting all with mid

Theory

ft_mid_TCC1: Formula ($F + 1 > 0$)

ft_mid_TCC2: Formula ($N - F \geq 0 \land (N - F > 0)$)

ft_mid_TCC3: Formula ($2 \neq 0$)

Proof

ft_mid_TCC1_PROOF: Prove ft_mid_TCC1

ft_mid_TCC2_PROOF: Prove ft_mid_TCC2

ft_mid_TCC3_PROOF: Prove ft_mid_TCC3

End mid_tcc
mid2.tcc: Module

Using mid2

Exporting all with mid2

Theory

ppred: Var function[naturalnumber — boolean]
p: Var natural number

good_greater_F1.TCC1: Formula
   (ppred(p)) ∧ (count(ppred. N) ≥ N - F) ⊃ (F + 1 > 0)

good_less_NF.TCC1: Formula
   (ppred(p)) ∧ (count(ppred. N) ≥ N - F)
   ⊃ (N - F ≥ 0) ∧ (N - F > 0)

good_greater_F1.pr.TCC1: Formula (F + 1 > 0)

good_less_NF.pr.TCC1: Formula (N - F ≥ 0) ∧ (N - F > 0)

Proof

good_greater_F1.TCC1.PROOF: Prove good_greater_F1.TCC1

good_less_NF.TCC1.PROOF: Prove good_less_NF.TCC1

good_greater_F1.pr.TCC1.PROOF: Prove good_greater_F1.pr.TCC1

good_less_NF.pr.TCC1.PROOF: Prove good_less_NF.pr.TCC1

End mid2.tcc
mid3.tcc: Module

Using mid3

Exporting all with mid3

Theory

\[ X: \text{Var number} \]
\[ Z: \text{Var number} \]
\[ \text{ppred: Var function}[\text{naturalnumber} \rightarrow \text{boolean}] \]
\[ k: \text{Var } \text{countmod.posint} \]
\[ \text{ppred2: Var function}[\text{naturalnumber} \rightarrow \text{boolean}] \]
\[ \text{ppred1: Var function}[\text{naturalnumber} \rightarrow \text{boolean}] \]
\[ \theta: \text{Var function}[\text{naturalnumber} \rightarrow \text{number}] \]
\[ \gamma: \text{Var function}[\text{naturalnumber} \rightarrow \text{number}] \]
\[ q: \text{Var naturalnumber} \]
\[ p3: \text{Var naturalnumber} \]
\[ p1: \text{Var naturalnumber} \]
\[ p: \text{Var naturalnumber} \]
\[ q1: \text{Var naturalnumber} \]
\[ \text{ft} \_ \text{mid}.p_i.TCC1: \text{Formula} (2 \neq 0) \]

**good.geq.F.add1.TCC1: Formula**

\[(\text{ppred}(p)) \land (\text{count}(\text{ppred}, N) \geq N - F) \cup (F + 1 > 0)\]

**okay.pair.geq.F.add1.TCC1: Formula**

\[(\text{ppred}(p1))\]
\[\land (\text{count}(\text{ppred}, N) \geq N - F \land \text{okay.pairs}(\theta, \gamma, X, \text{ppred}))\]
\[\cup (F + 1 > 0)\]

**okay.pair.geq.F.add1.TCC2: Formula**

\[(\text{ppred}(q1))\]
\[\land (\theta(p1) \geq \theta(F+1))\]
\[\land (\text{ppred}(p1))\]
\[\land (\text{count}(\text{ppred}, N) \geq N - F)\]
\[\land \text{okay.pairs}(\theta, \gamma, X, \text{ppred})\]
\[\cup (F + 1 > 0)\]
good_between_TCC1: Formula
\[(\gamma_{F+1} \geq \gamma(p)) \land (p_{pred}(p)) \land (\text{count}(p_{pred}, N) \geq N - F) \land (N - F \geq 0) \land (N - F > 0)\]

ft_mid_prec_sym1_TCC1: Formula
\[(\text{okay}_{\text{Readpred}}(\gamma, Z, p_{pred})) \land (\text{okay}_{\text{Readpred}}(\theta, Z, p_{pred})) \land (\text{okay}_{\text{pairs}}(\theta, \gamma, X, p_{pred})) \land (\text{count}(p_{pred}, N) \geq N - F) \land (F + 1 > 0)\]

ft_mid_prec_sym1_TCC2: Formula
\[(\text{okay}_{\text{Readpred}}(\gamma, Z, p_{pred})) \land (\text{okay}_{\text{Readpred}}(\theta, Z, p_{pred})) \land (\text{okay}_{\text{pairs}}(\theta, \gamma, X, p_{pred})) \land (\text{count}(p_{pred}, N) \geq N - F) \land (N - F \geq 0) \land (N - F > 0)\]

ft_mid_prec_sym1_TCC3: Formula
\[(\text{count}(p_{pred}, N) \geq N - F) \land (\text{okay}_{\text{pairs}}(\theta, \gamma, X, p_{pred})) \land (\text{okay}_{\text{Readpred}}(\theta, Z, p_{pred})) \land (\text{okay}_{\text{Readpred}}(\gamma, Z, p_{pred})) \land (((\theta_{F+1}) + \theta_{(N - F)}) \geq (\gamma_{F+1}) + \gamma_{(N - F)})) \land (F + 1 > 0)\]

ft_mid_prec_sym1_TCC4: Formula
\[(\text{count}(p_{pred}, N) \geq N - F) \land (\text{okay}_{\text{pairs}}(\theta, \gamma, X, p_{pred})) \land (\text{okay}_{\text{Readpred}}(\theta, Z, p_{pred})) \land (\text{okay}_{\text{Readpred}}(\gamma, Z, p_{pred})) \land (((\theta_{F+1}) + \theta_{(N - F)}) \geq (\gamma_{F+1}) + \gamma_{(N - F)})) \land (N - F \geq 0) \land (N - F > 0)\]

mid_gt_imp_sel_gt_TCC1: Formula
\[((c_{fn_{MID}}(p, \theta) \geq c_{fn_{MID}}(q, \gamma))) \land (F + 1 > 0)\]
mid_gt_imp_sel_gt_TCC2: Formula
\(((c_{r_{\text{MID}}}(p, \theta)) \geq c_{r_{\text{MID}}}(q, \gamma))\)
\[ (N - F \geq 0) \land (N - F > 0) \]

ft_mid_prec_sym1.pr_TCC1: Formula \((F + 1 > 0)\)

ft_mid_prec_sym1.pr_TCC2: Formula \((N - F \geq 0) \land (N - F > 0)\)

Proof

ft_mid.Pi_TCC1.PROOF: Prove ft_mid.Pi_TCC1

good_geq_F_add1.TCC1.PROOF: Prove good_geq_F_add1.TCC1

okay_pair_geq_F.add1.TCC1.PROOF: Prove okay_pair_geq_F.add1.TCC1

okay_pair_geq_F.add1.TCC2.PROOF: Prove okay_pair_geq_F.add1.TCC2

good_between.TCC1.PROOF: Prove good_between.TCC1

ft_mid_prec_sym1.TCC1.PROOF: Prove ft_mid_prec_sym1.TCC1

ft_mid_prec_sym1.TCC2.PROOF: Prove ft_mid_prec_sym1.TCC2

ft_mid_prec_sym1.TCC3.PROOF: Prove ft_mid_prec_sym1.TCC3

ft_mid_prec_sym1.TCC4.PROOF: Prove ft_mid_prec_sym1.TCC4

mid_gt_imp_sel.gt_TCC1.PROOF: Prove mid_gt_imp_sel.gt_TCC1

mid_gt_imp_sel.gt_TCC2.PROOF: Prove mid_gt_imp_sel.gt_TCC2

ft_mid_prec_sym1.pr_TCC1.PROOF: Prove ft_mid_prec_sym1.pr_TCC1

ft_mid_prec_sym1.pr_TCC2.PROOF: Prove ft_mid_prec_sym1.pr_TCC2

End mid3_tcc
mid4_tcc: Module

Using mid4

Exporting all with mid4

Theory

q: Var naturalnumber
p: Var naturalnumber
y: Var number
x: Var number
p1: Var naturalnumber
ft_mid_less_TCC1: Formula \((F + 1 > 0)\)

ft_mid_greater_TCC1: Formula \((N - F \geq 0) \land (N - F > 0)\)

Proof

ft_mid_less_TCC1_PROOF: Prove ft_mid_less_TCC1

ft_mid_greater_TCC1_PROOF: Prove ft_mid_greater_TCC1

End mid4_tcc
mid: Module

Using arith. clockassumptions: select_defs. ft_mid_assume

Exporting all with select_defs

Theory

process: Type is nat
Clocktime: Type is number
l, m, n, p, q: Var process
q: Var function[process — Clocktime]
i, j, k: Var posint
T, X, Y, Z: Var Clocktime

cfnMID: function[process. function[process — Clocktime]
— Clocktime] =
(λ p, q: (φ_{(F+1)} + φ_{(N-F)})/2)

ft_mid_trans_inv: Lemma

cfnMID(P, (λ q: φ(q) + X)) = cfnMID(p, φ) + X

Proof

add_assoc.hack: Lemma X + Y + Z + Y = (X + Z) + 2*Y

add_assoc.hack_pr: Prove add_assoc.hack from

*1 + *2 {x — 2, y — Y}
**ft_mid_trans_inv_pr:** Prove ft_mid_trans_inv from

- $cftMidID \{ \emptyset - (\lambda q : \emptyset(q) + X) \}$,
- $cftMidID \{ \emptyset - (\lambda q : \emptyset(q) + X) \}$,
- select_trans_inv \{ $k - F + 1$ \},
- select_trans_inv \{ $k - N - F$ \},
- add_assoc_hack
  - \{ $X - \emptyset_{(F+1)}$, $Z - \emptyset_{(N-F)}$, $Y - X$ \},
- div_distrib
  - \{ $x - (\emptyset_{(F+1)} + \emptyset_{(N-F)})$, $y - 2 \times X$, $z - 2$ \},
- div_cancel \{ $x - 2, y - X$ \},
- ft_mid_maxfaults

End mid
mid2: Module

Using arith, clockassumptions, mid

Exporting all with mid

Theory

Clocktime: Type is number
m, n, p, q, p₁, q₁: Var process
i, j, k, l: Var posint
x, y, z, r, s, t: Var time
D, X, Y, Z, R, S, T: Var Clocktime
v, θ, γ: Var function[process — Clocktime]
ppred, ppred₁, ppred₂: Var function[process — bool]

**good_greater_F1**: Lemma
\[ \text{count}(\text{ppred}, N) \geq N - F \supset (\exists p : \text{ppred}(p) \land \vartheta(p) \geq \vartheta(F+1)) \]

**good_less_NF**: Lemma
\[ \text{count}(\text{ppred}, N) \geq N - F \supset (\exists p : \text{ppred}(p) \land \vartheta(p) \leq \vartheta(N-F)) \]

Proof

**good_greater_F1_pr**: Prove **good_greater_F1** \( \{ p | p \geq p₃ \} \) from
\[ \text{count.greatereselect} \{ k - F + 1 \}, \]
\[ \text{ft.mid.maxfaults}, \]
\[ \text{count.exists} \]
\[ \{ \text{ppred} - (\lambda p₁ : \text{ppred1}₀ p₄(p₁) \land \text{ppred2}₀ p₄(p₁)), \]
\[ n - N \}, \]
\[ \text{pigeon.hole} \]
\[ \{ \text{ppred1} = \text{ppred}, \]
\[ \text{ppred2} - (\lambda p₁ : \vartheta(p₁) \geq \vartheta(F+1)), \]
\[ n - N, \]
\[ k - 1 \} \]

68
good_less_NF.pr: Prove good_less_NF \( \{ p - p^3 \} \) from
count_leq_select \( \{ k - N - F \} \).
ft_mid_maxfaults,
count_exists
\( \{ \text{ppred} = ( \lambda p_1 : \text{ppred1} @ p_4(p_1) \land \text{ppred2} @ p_4(p_1) ), n - N \} \),
pigeon_hole
\( \{ \text{ppred1} = \text{ppred}, \text{ppred2} = ( \lambda p_1 : n_{(N-F)} \geq n(p_1) ), n - N, k - 1 \} \)

End mid2
mid3: Module

Using arith. clockassumptions. mid2

Exporting all with mid2

Theory

Clocktime: Type is number
m, n, p, q, p_1, q_1: Var process
i, j, k, l: Var posint
x, y, z, r, s, t: Var time
D, X, Y, Z, R, S, T: Var Clocktime
\( \theta, \gamma: \) Var function[process \rightarrow Clocktime]
\( \text{ppred, ppred1, ppred2: Var function[process \rightarrow bool]} \)
ft_mid_Pi: function[Clocktime, Clocktime \rightarrow Clocktime] ==
( \lambda X, Z : Z/2 + X)

exchange_order: Lemma
\( \text{ppred}(p) \land \text{ppred}(q) \land \theta(q) \leq \theta(p) \land \gamma(p) \leq \gamma(q) \land \text{okay_pairs}(\theta, \gamma, X, \text{ppred}) \)
\( \triangleright |\theta(p) - \gamma(q)| \leq X \)

good_geq_F_add1: Lemma
\( \text{count}(\text{ppred}, N) \geq N - F \supset (\exists p : \text{ppred}(p) \land \theta(p) \geq \theta_{(F+1)}) \)

okay_pair_geq_F_add1: Lemma
\( \text{count}(\text{ppred}, N) \geq N - F \land \text{okay_pairs}(\theta, \gamma, X, \text{ppred}) \supset (\exists p_1, q_1 : \text{ppred}(p_1) \land \theta(p_1) \geq \theta_{(F+1)} \land \text{ppred}(q_1) \land \gamma(q_1) \geq \gamma_{(F+1)} \land |\theta(p_1) - \gamma(q_1)| \leq X) \)

good_between: Lemma
\( \text{count}(\text{ppred}, N) \geq N - F \supset (\exists p : \text{ppred}(p) \land \gamma_{(F+1)} \geq \gamma(p) \land \theta(p) \geq \theta_{(N-F)}) \)
ft_mid_precision_enhancement: Lemma
ppred(\(p\))
\[\land p\text{ppred}(\eta)\]
\[\land \text{count}(\text{ppred}, N) \geq N - F\]
\[\land \text{okay_pairs}(\theta, \gamma, X, \text{ppred})\]
\[\land \text{okay_Readpred}(\theta, Z, \text{ppred})\]
\[\land \text{okay_Readpred}(\gamma, Z, \text{ppred})\]
\[\sqsupset |cfu_{MID}(\mu, \theta) - cfu_{MID}(\nu, \gamma)| \leq ft_{mid,Pi}(X, Z)\]

ft_mid_prec_enh_sym: Lemma
ppred(\(p\))
\[\land p\text{ppred}(\eta)\]
\[\land \text{count}(\text{ppred}, N) \geq N - F\]
\[\land \text{okay_pairs}(\theta, \gamma, X, \text{ppred})\]
\[\land \text{okay_Readpred}(\theta, Z, \text{ppred})\]
\[\land \text{okay_Readpred}(\gamma, Z, \text{ppred})\]
\[\land (cfu_{MID}(p, \theta) \geq cfu_{MID}(q, \gamma))\]
\[\sqsupset |cfu_{MID}(p, \theta) - cfu_{MID}(q, \gamma)| \leq ft_{mid,Pi}(X, Z)\]

ft_mid_prec_sym1: Lemma
count(\text{ppred}, N) \geq N - F
\[\land \text{okay_pairs}(\theta, \gamma, X, \text{ppred})\]
\[\land \text{okay_Readpred}(\theta, Z, \text{ppred})\]
\[\land \text{okay_Readpred}(\gamma, Z, \text{ppred})\]
\[\land ((\theta_{(F+1)} + \theta_{(N-F)}))\]
\[\geq (\gamma_{(F+1)} + \gamma_{(N-F)})\]
\[\sqsupset (|\theta_{(F+1)} + \theta_{(N-F)}| - |\gamma_{(F+1)} + \gamma_{(N-F)}|)\]
\[\leq Z + 2 \ast X\]

mid_gt_imp_sel_gt: Lemma
\[|cfu_{MID}(\mu, \theta) \geq cfu_{MID}(\nu, \gamma)|\]
\[\sqsupset ((\theta_{(F+1)} + \theta_{(N-F)})) \geq (\gamma_{(F+1)} + \gamma_{(N-F)}))\]

okay_pairs_sym: Lemma
\[\text{okay_pairs}(\theta, \gamma, X, \text{ppred}) \sqsupset \text{okay_pairs}(\gamma, \theta, X, \text{ppred})\]

Proof
ft_mid_prec_sym1_pr: Prove ft_mid_prec_sym1 from
good_between,
okay_pair_eq_F_add1,
good_less_NF \{\theta - \gamma\},
abs_geq
\{ x - (\gamma(q_1 \circ p_2) - \gamma(p^{i} p_3)) + (\theta(p^{i} p_1) - \gamma(p^{i} p_1)) \\
+ (\theta(p_1 \circ p_2) - \gamma(q_1 \circ p_2)), \\
y - (\theta_{(F+1)} + \theta_{(N-F)}) - (\gamma_{(F+1)} + \gamma_{(N-F)}) \},
abs_plus
\{ x - (\gamma(q_1 \circ p_2) - \gamma(p^{i} p_3)) + (\theta(p^{i} p_1) - \gamma(p^{i} p_1)), \\
y - (\theta(p_1 \circ p_2) - \gamma(q_1 \circ p_2)) \},
abs_plus
\{ x - (\gamma(q_1 \circ p_2) - \gamma(p^{i} p_3)), \\
y - (\theta(p^{i} p_1) - \gamma(p^{i} p_1)) \},
okay_pairs \{\gamma - \theta, \theta - \gamma, p_1 - p^{i} p_1\},
okay_Readpred
\{\gamma - \gamma, \\
y - Z, \\
l - q_1 \circ p_2, \\
m - p^{i} p_3\},
distrib \{x - 1, y - 1, z - X\},
mult_lident \{x - X\}

mid_gt_imp_sel_gt_pr: Prove mid_gt_imp_sel_gt from
cfh_{MID} \{\theta - \theta\},
cfh_{MID} \{\theta - \gamma, p - q\},
mult_leq
\{ x - cfh_{MID}(p, \theta), \\
y - cfh_{MID}(q, \gamma), \\
z - 2 \},
mult_div \{ x - (\theta_{(F+1)} + \theta_{(N-F)}), y - 2 \},
mult_div \{ x - (\gamma_{(F+1)} + \gamma_{(N-F)}), y - 2 \}

72
ft_mid_prec_enh_sym.pr: Prove \( ftmidprec\_enh\_sym \) from

\[
\begin{align*}
\text{cf}^{\mu}_{\text{MID}} \{ \vartheta - \vartheta \}, \\
\text{cf}^{\mu}_{\text{MID}} \{ \vartheta - \gamma, p - q \}, \\
\text{div}\_\text{minus}\_\text{distrib} \\
\{ x - (\vartheta_{(F+1)} + \vartheta_{(N-F)}), \\
y - (\gamma_{(F+1)} + \gamma_{(N-F)}), \\
z - 2 \}, \\
\text{abs}\_\text{div} \\
\{ x - (\vartheta_{(F+1)} + \vartheta_{(N-F)}) - (\gamma_{(F+1)} + \gamma_{(N-F)}), \\
y - 2 \}, \\
\text{ft}\_\text{mid}\_\text{prec}\_\text{sym}1, \\
\text{mid}\_\text{gt}\_\text{imp}\_\text{sel}\_\text{gt}, \\
\text{div}\_\text{ineq} \\
\{ x - |(\vartheta_{(F+1)} + \vartheta_{(N-F)}) - (\gamma_{(F+1)} + \gamma_{(N-F)})|, \\
y - Z + 2 \times X, \\
z - 2 \}, \\
\text{div}\_\text{distrib} \{ x - Z, y - 2 \times X, z - 2 \}, \\
\text{div}\_\text{cancel} \{ x - 2, y - X \}
\end{align*}
\]

okay_pairs_sym.pr: Prove \( \text{okay}\_\text{pairs}\_\text{sym} \) from

\[
\begin{align*}
\text{okay}\_\text{pairs} \{ \gamma - \theta, \theta - \gamma, p_3 - p_3@p2 \}, \\
\text{okay}\_\text{pairs} \{ \gamma - \gamma, \theta - \theta \}, \\
\text{abs}\_\text{com} \{ x - \theta(p_3@p2), y - \gamma(p_3@p2) \}
\end{align*}
\]

ft_mid_precision_enhancement.pr: Prove \( ftmidprecision\_enhancement \) from

\[
\begin{align*}
\text{ft}\_\text{mid}\_\text{prec}\_\text{enh}\_\text{sym}, \\
\text{ft}\_\text{mid}\_\text{prec}\_\text{enh}\_\text{sym} \\
\{ p - q@p1, \\
q - p@q2, \\
\theta - \gamma@p1, \\
\gamma - \vartheta@p1 \}, \\
\text{okay}\_\text{pairs}\_\text{sym}, \\
\text{abs}\_\text{com} \{ x - \text{cf}^{\mu}_{\text{MID}}(p, \theta), y - \text{cf}^{\mu}_{\text{MID}}(q, \gamma) \}
\end{align*}
\]
okay_pair.geq.F.add1.pr: Prove
okay_pair.geq.F.add1
\{ p_1 \quad \text{if} \quad (\theta(p^{\text{up}p_2}) \geq \theta(p^{\text{up}p_1})) \}
\quad \text{then} \quad p^{\text{up}p_2}
\quad \text{elsif} \quad (\gamma(p^{\text{up}p_1}) \geq \gamma(p^{\text{up}p_2})) \quad \text{then} \quad p^{\text{up}p_1} \quad \text{else} \quad p^{\text{up}p_3}
\quad \text{end if},

\{ p_1 \quad \text{if} \quad (\theta(p^{\text{up}p_2}) \geq \theta(p^{\text{up}p_1})) \}
\quad \text{then} \quad p^{\text{up}p_2}
\quad \text{elsif} \quad (\gamma(p^{\text{up}p_1}) \geq \gamma(p^{\text{up}p_2})) \quad \text{then} \quad p^{\text{up}p_1} \quad \text{else} \quad q^{\text{up}p_3}
\quad \text{end if} \} \quad \text{from}
good.geq.F.add1 \{ \vartheta = \vartheta \},
good.geq.F.add1 \{ \vartheta = \gamma \},
exchange.order \{ p - p^{\text{up}p_1}, q - p^{\text{up}p_2} \},
okay.pairs \{ \gamma - \theta, \theta - \gamma, p_3 - p^{\text{up}p_1} \},
okay.pairs \{ \gamma - \theta, \theta - \gamma, p_3 - p^{\text{up}p_2} \}
good.geq.F.add1.pr: Prove good.geq.F.add1 \{ p - p^{\text{up}p_1} \} \quad \text{from}
count.exists
\{ \text{ppred} = (\lambda p: ((\text{ppred}_1 \uparrow p_2) p) \land ((\text{ppred}_2 \uparrow p_2) p)),
\quad n = N \},
pigeon.hole
\{ n = N,
\quad k = 1,
\quad \text{ppred}_1 = \text{ppred},
\quad \text{ppred}_2 = (\lambda p: \theta(p) \geq \theta((k \uparrow p_3))) \},
count.geq.select \{ k = F + 1 \},
ft.mid.maxfaults

74
good_between_pr: Prove good_between \( \{ p - p\vec{p}_1 \} \) from
count_exists
\[
\begin{align*}
\{ \text{ppred} & - ( \lambda p: \text{((ppred}_1{p_2})p) \land \text{((ppred}_2{p_2})p)), \\
n & - N \}, \\
pigeon_hole & \\
\{ n - N, \\
k & - 1, \\
\text{ppred}_1 & - ( \lambda p: \text{((ppred}_1{p_3})p) \land \text{((ppred}_2{p_3})p)), \\
\text{ppred}_2 & - ( \lambda p: \theta(p) \geq \theta((k\vec{p}_4))) \}, \\
pigeon_hole & \\
\{ n - N, \\
k & - k\vec{p}_5, \\
\text{ppred}_1 & - \text{ppred}, \\
\text{ppred}_2 & - ( \lambda p: \gamma((k\vec{p}_5)) \geq \gamma(p))) \}, \\
\text{count_geq_select} & \{ \theta - \theta, k - N - F \}, \\
\text{count_leq_select} & \{ \theta - \gamma, k - F + 1 \}, \\
\text{ft_mid_maxfaults} & \end{align*}
\]

exchange_order_pr: Prove exchange_order from
okay_pairs \( \{ \gamma - \theta, \theta - \gamma, p_3 - p \} \),
okay_pairs \( \{ \gamma - \theta, \theta - \gamma, p_3 - q \} \),
abs_geq \( \{ x - (\theta(p) - \gamma(p)), y - \theta(p) - \gamma(q) \} \),
abs_geq \( \{ x - (\gamma(q) - \theta(q)), y - \gamma(q) - \theta(p) \} \),
abs_com \( \{ x - \theta(q), y - \gamma(q) \} \),
abs_com \( \{ x - \theta(p), y - \gamma(q) \} \)

End mid3
mid4: Module

Using arith. clockassumptions. mid3

Exporting all with clockassumptions. mid3

Theory

process: Type is nat
Clocktime: Type is number
m, n, p, q, q1: Var process
i, j, k: Var posint
x, y, z, r, s, t: Var time
D, X, Y, Z, R, S, T: Var Clocktime
v, θ, γ: Var function[process -- Clocktime]
ppred, ppred1, ppred2: Var function[process -- bool]

ft_mid_accuracy preservation: Lemma
ppred(p)
\wedge ppred(q)
\wedge count(ppred, N) \geq N - F \wedge okay_Readpred(\theta, X, ppred)
\Rightarrow |cf_{M1B}(p, \theta) - \theta(q)| \leq N

ft_mid.less: Lemma \ cfn_{M1B}(p, \theta) \leq \theta(F+1)

ft_mid.greater: Lemma \ cfn_{M1B}(p, \theta) \geq \theta(N-F)

abs_q.less: Lemma
count(ppred, N) \geq N - F
\Rightarrow (\exists p_1 : ppred(p_1) \wedge \theta(p_1) \leq cfn_{M1B}(p, \theta))

abs_q.greater: Lemma
count(ppred, N) \geq N - F
\Rightarrow (\exists p_1 : ppred(p_1) \wedge \theta(p_1) \geq cfn_{M1B}(p, \theta))

ft_mid.bnd_by_good: Lemma
count(ppred, N) \geq N - F
\Rightarrow (\exists p_1 : ppred(p_1) \wedge |cf_{M1B}(p, \theta) - \theta(q)| \leq |\theta(p_1) - \theta(q)|)

maxfaults.lem: Lemma F + 1 \leq N - F
ft_select: Lemma $\vartheta_{F+1} \geq \vartheta_{N-F}$

Proof

ft_select_pr: Prove ft_select from
select_ax \{ i = F + 1, k = N - F \}, maxfaults_lem

maxfaults_lem_pr: Prove maxfaults_lem from ft_mid_maxfaults

ft_mid_bnd_by_good_pr: Prove
ft_mid_bnd_by_good
\{ p_1 - ( \begin{array}{l}
\text{if } cf_{MID}(p, \vartheta) \geq \vartheta(q) \\
\text{then } p_1 \oplus p_1 \\
\text{else } p_1 \oplus p_2 \\
\text{end if}
\end{array} ) \} \text{ from }
abs_q_greater,
abs_q_less,
abs_com \{ x - \vartheta(q), y - \vartheta(p_1 \oplus c) \},
abs_com \{ x - \vartheta(q), y - cf_{MID}(p, \vartheta) \},
abs_geq \{ x - x \oplus p_3 - y \oplus p_3, y - x \oplus p_4 - y \oplus p_4 \},
abs_geq \{ x - \vartheta(p_1 \oplus c) - \vartheta(q), y - cf_{MID}(p, \vartheta) - \vartheta(q) \}

abs_q_less_pr: Prove abs_q_less \{ p_1 - p \oplus p_1 \} \text{ from }
good_less_NF, ft_mid_greater

abs_q_greater_pr: Prove abs_q_greater \{ p_1 - p \oplus p_1 \} \text{ from }
good_greater_F1, ft_mid_less

mult_hack: Lemma $X + X = 2 \times X$

mult_hack_pr: Prove mult_hack from $*1 * *2 \{ x - 2, y - X \}$

ft_mid_less_pr: Prove ft_mid_less from
$cf_{MID}$,
ft_select,
div_ineq
\{ x - (\vartheta_{F+1} + \vartheta_{N-F}), \\
y - (\vartheta_{F+1} + \vartheta_{F+1}), \\
z - 2 \},
div_cancel \{ x - 2, y - \vartheta_{F+1} \},
mult_hack \{ X - \vartheta_{F+1} \}
ft_mid_greater_pr: Prove ft_mid_greater from 
\text{cf:}_MID, 
ft_select, 
\text{div.ineq}
\begin{align*}
x &= (d(F+1) + d(N-F), \\
y &= (d(F+1) + d(N-F), \\
z &= 2, \\
\text{div.cancel} &\{x - 2, y - d(N-F)\}, \\
\text{mult.hack} &\{X - d(N-F)\}
\end{align*}

ft_mid_acc_pres_pr: Prove ft_mid_accuracy_preservation from 
ft_mid_bnd_by_good, 
\text{okay.Readpred}
\begin{align*}
\gamma &= \theta, \\
Y &= X, \\
l &= \mathbf{p}_1 \otimes \mathbf{p}_1, \\
m &= q^{\text{var}}
\end{align*}

End mid4
C Proof Chain Status

C.1 Translation Invariance

Terse proof chain for proof ft_mid_trans_inv_pr in module mid

Use of the formula
mid.ft_mid
requires the following TCCs to be proven
mid_tcc.ft_mid_TCC1
mid_tcc.ft_mid_TCC2
mid_tcc.ft_mid_TCC3

Use of the formula
division.div_distrib
requires the following TCCs to be proven
division_tcc.mult_div_l_TCC1
division_tcc.mult_div_TCC1
division_tcc.div_cancel_TCC1
division_tcc.ceil_mult_div_TCC1
division_tcc.div_nonnegative_TCC1
division_tcc.div_ineq_TCC1
division_tcc.div_minus_1_TCC1

=============== SUMMARY ===============

The proof chain is complete

The axioms and assumptions at the base are:
clocksort.funsort_trans_inv
division.mult_div_1
division.mult_div_2
division.mult_div_3
division_ft_assume_ft_mid_maxfaults
Total: 5

The definitions and type-constraints are:
mid.ft_mid
multiplication.mult
Total: 2

The formulae used are:
- `division.div_cancel`
- `division.div_distrib`
- `division_tcc.ceil_mult_div_TCC1`
- `division_tcc.div_cancel_TCC1`
- `division_tcc.div_ineq_TCC1`
- `division_tcc.div_minus_1_TCC1`
- `division_tcc.div_nonnegative_TCC1`
- `division_tcc.mult_div_1_TCC1`
- `division_tcc.mult_div_TCC1`
- `mid.add_assoc_hack`
- `mid_tcc.ft_mid_TCC1`
- `mid_tcc.ft_mid_TCC2`
- `mid_tcc.ft_mid_TCC3`
- `multiplication.distrib`
- `multiplication.mult_lident`
- `multiplication.mult_rident`
- `select_defs.select_trans_inv`

Total: 17

The completed proofs are:
- `division.div_cancel_pr`
- `division.div_distrib_pr`
- `division_tcc.ceil_mult_div_TCC1_PROOF`
- `division_tcc.div_cancel_TCC1_PROOF`
- `division_tcc.div_ineq_TCC1_PROOF`
- `division_tcc.div_minus_1_TCC1_PROOF`
- `division_tcc.div_nonnegative_TCC1_PROOF`
- `division_tcc.mult_div_1_TCC1_PROOF`
- `division_tcc.mult_div_TCC1_PROOF`
- `mid.add_assoc_hack_pr`
- `mid.ft_mid_trans_inv_pr`
- `mid_tcc.ft_mid_TCC1_PROOF`
- `mid_tcc.ft_mid_TCC3_PROOF`
- `multiplication.distrib_proof`
- `multiplication.mult_lident_proof`
- `multiplication.mult_rident_proof`
select_defs.select_trans_inv_pr
tcc_mid.ft_mid_TCC2_PROOF
Total: 18

C.2 Precision Enhancement

Terse proof chain for proof ft_mid_precision_enhancement_pr in module mid3

Use of the formula
mid3.ft_mid_prec_enh_sym
requires the following TCCs to be proven
mid3_tcc.ft_mid_Pi_TCC1
mid3_tcc.good_geq_F_addl_TCC1
mid3_tcc.okay_pair_geq_F_addl_TCC1
mid3_tcc.okay_pair_geq_F_addl_TCC2
mid3_tcc.good_between_TCC1
mid3_tcc.ft_mid_prec_sym1_TCC1
mid3_tcc.ft_mid_prec_sym1_TCC2
mid3_tcc.ft_mid_prec_sym1_TCC3
mid3_tcc.ft_mid_prec_sym1_TCC4
mid3_tcc.mid_gt_imp_sel_gt_TCC1
mid3_tcc.mid_gt_imp.sel_gt_TCC2
mid3_tcc.ft_mid_prec_sym1_pr_TCC1
mid3_tcc.ft_mid_prec_sym1_pr_TCC2

Use of the formula
mid.ft_mid
requires the following TCCs to be proven
mid_tcc.ft_mid_TCC1
mid_tcc.ft_mid_TCC2
mid_tcc.ft_mid_TCC3

Use of the formula
division.div_minus_distrib
requires the following TCCs to be proven
division_tcc.mult_div.l_TCC1
division_tcc.mult_div.TCC1
division_tcc.div_cancel_TCC1
division_tcc.ceil_mult_div_TCC1
division_tcc.div_nonnegative_TCC1
division_tcc.div_ineq_TCC1
division_tcc.div_minus_1_TCC1

Use of the formula
countmod.count_exists
requires the following TCCs to be proven
countmod_tcc.posint_TCC1
countmod_tcc.count_TCC1
countmod_tcc.count_TCC2
countmod_tcc.count_TCC3
countmod_tcc.count_TCC4
countmod_tcc.count_TCC5

Formula countmod_tcc.count_TCC4 is a termination TCC for countmod.count
Proof of
countmod_tcc.count_TCC4
must not use
   countmod.count

Formula countmod_tcc.count_TCC5 is a termination TCC for countmod.count
Proof of
countmod_tcc.count_TCC5
must not use
   countmod.count

Use of the formula
natinduction.induction
requires the following TCCs to be proven
   natinduction_tcc.ind_m_proof_TCC1

Use of the formula
noetherian[naturalnumber, natinduction.less].general_induction
requires the following assumptions to be discharged
   noetherian[naturalnumber, natinduction.less].well_founded

Use of the formula
noetherian[countmod.nk_type, countmod.nk_less].general_induction
requires the following assumptions to be discharged
noetherian[countmod.nk_type, countmod.nk_less].well_founded

Use of the formula
mid2.good_less_NF
requires the following TCCs to be proven
mid2_tcc.good_greater_F1_TCC1
mid2_tcc.good_less_NF_TCC1
mid2_tcc.good_greater_F1_pr_TCC1
mid2_tcc.good_less_NF_pr_TCC1

================ SUMMARY ===============

The proof chain is complete

The axioms and assumptions at the base are:
clocksort.cnt_sort_geq
clocksort.cnt_sort_leq
division.mult_div_1
division.mult_div_2
division.mult_div_3
ft_mid_assume.ft_mid_maxfaults
multiplication.mult_non_neg
multiplication.mult_pos
noetherian[EXPR, EXPR].general_induction
Total: 9

The definitions and type-constraints are:
absmod.abs
clockassumptions.okay_Readpred
clockassumptions.okay_pairs
countmod.count
countmod.countsize
countmod.exists_less
countmod.nk_noeth_pred
countmod.nk_ph_pred
mid.ft_mid
multiplication.mult
The formulae used are:

- `absmod.abs_com`
- `absmod.abs_geq`
- `absmod.abs_plus`
- `countmod.count_exists`
- `countmod.count_exists_base`
- `countmod.count_exists_ind`
- `countmod.nk_ph_expand`
- `countmod.nk_ph_lem`
- `countmod.nk_ph_noeth_hyp`
- `countmod.ph_case0`
- `countmod.ph_case0k`
- `countmod.ph_case0n`
- `countmod.ph_case1`
- `countmod.ph_case2`
- `countmod.ph_case2a`
- `countmod.ph_case2b`
- `countmod.pigeon_hole`
- `countmod_tcc.count_TCC1`
- `countmod_tcc.count_TCC2`
- `countmod_tcc.count_TCC3`
- `countmod_tcc.count_TCC4`
- `countmod_tcc.count_TCC5`
- `countmod_tcc.posint_TCC1`
- `division.abs_div`
- `division.div_cancel`
- `division.div_distrib`
- `division.div_ineq`
- `division.mult_minus`
- `division.mult_div`
- `division_tcc.ceil_mult_div_TCC1`
- `division_tcc.div_cancel_TCC1`
- `division_tcc.div_ineq_TCC1`
multiplication.mult_lident
multiplication.mult_rident
multiplication.pos_product
natinduction.induction
natinduction_tcc.ind.m_proof_TCC1
noetherian[countmod.nk_type, countmod.nk_less].well_founded
noetherian[naturalnumber, natinduction.less].well_founded
select_defs.count_geq_select
select_defs.count_leq_select

Total: 83

The completed proofs are:
absmod.abs_com_proof
absmod.abs_geq_proof
absmod.abs_plus_pr
countmod.count_exists_base_pr
countmod.count_exists_ind_pr
countmod.count_exists_pr
countmod.nk_ph_expand_pr
countmod.nk_ph_lem_pr
countmod.nk_ph_noeth_hyp_pr
countmod.nk_well_founded
countmod.ph_case0_pr
countmod.ph_case0k_pr
countmod.ph_case0n_pr
countmod.ph_case1_pr
countmod.ph_case2_pr
countmod.ph_case2a_pr
countmod.ph_case2b_pr
countmod.pigeon_hole_pr
countmod_tcc.count_TCC1_PROOF
countmod_tcc.count_TCC2_PROOF
countmod_tcc.count_TCC3_PROOF
division.abs_div_pr
division.div_cancel_pr
division.div_distrib_pr
division.div_ineq_pr
division.div_minus_1_pr
division.div_minus_distrib_pr
division.div_nonnegative_pr
division.mult_div_pr
division.mult_minus_pr
division_tcc.ceil_mult_div_TCC1_PROOF
division_tcc.div_cancel_TCC1_PROOF
division_tcc.div_ineq_TCC1_PROOF
division_tcc.div_minus_1_TCC1_PROOF
division_tcc.div_nonnegative_TCC1_PROOF
division_tcc.mult_div_1_TCC1_PROOF
division_tcc.mult_div_TCC1_PROOF
mid2.good_less_NF_pr
mid2_tcc.good_greater_F1_TCC1_PROOF
mid2_tcc.good_greater_F1_pr_TCC1_PROOF
mid3.exchange_order_pr
mid3.ft.mid_prec_enh_sym_pr
mid3.ft.mid_prec_sym1_pr
mid3.ft.mid_precision_enhancement_pr
mid3.good_between_pr
mid3.good_geq_F.add1_pr
mid3.mid_gt.imp.sel.gt.pr
mid3.okay_pair.geq_F.add1_pr
mid3.okay_pairs_sym_pr
mid3_tcc.ft.mid_pi_TCC1_PROOF
mid3_tcc.ft.mid_prec_sym_TCC1_PROOF
mid3_tcc.ft.mid_prec_sym1_TCC3_PROOF
mid3_tcc.ft.mid_prec_sym1_pr_TCC1_PROOF
mid3_tcc.good_geq_F.add1_TCC1_PROOF
mid3_tcc.mid_gt.imp.sel.gt_TCC1_PROOF
mid3_tcc.okay_pair.geq_F.add1_TCC1_PROOF
mid3_tcc.okay_pair.geq_F.add1_TCC2_PROOF
mid_tcc.ft.mid_TCC1_PROOF
mid_tcc.ft.mid_TCC3_PROOF
mid_top.countmod_TCC4_pr
mid_top.countmod_TCC5_pr
mid_top.posint_TCC1_PROOF
multiplication.distrib_minus_pr
multiplication.distrib_proof
multiplication.mult_com_pr
multiplication.mult_gt_pr

87
C.3 Accuracy Preservation

Terse proof chain for proof ft_mid_acc_pres_pr in module mid4

Use of the formula
mid4.ft_mid_bnd_by_good
requires the following TCCs to be proven
mid4_tcc.ft_mid_less_TCC1
mid4_tcc.ft_mid_greater_TCC1

Use of the formula
mid2.good_greater_F1
requires the following TCCs to be proven
mid2_tcc.good_greater_F1_TCC1
mid2_tcc.good_less_NF_TCC1
mid2_tcc.good_greater_F1_pr_TCC1
mid2_tcc.good_less_NF_pr_TCC1
Use of the formula
  countmod.count_exists
requires the following TCCs to be proven
  countmod_tcc.posint_TCC1
  countmod_tcc.count_TCC1
  countmod_tcc.count_TCC2
  countmod_tcc.count_TCC3
  countmod_tcc.count_TCC4
  countmod_tcc.count_TCC5

Formula countmod_tcc.count_TCC4 is a termination TCC for countmod.count
Proof of
  countmod_tcc.count_TCC4
must not use
  countmod.count

Formula countmod_tcc.count_TCC5 is a termination TCC for countmod.count
Proof of
  countmod_tcc.count_TCC5
must not use
  countmod.count

Use of the formula
  natinduction.induction
requires the following TCCs to be proven
  natinduction_tcc.ind_m_proof_TCC1

Use of the formula
  noetherian[naturalnumber, natinduction.less].general_induction
requires the following assumptions to be discharged
  noetherian[naturalnumber, natinduction.less].well_founded

Use of the formula
  noetherian[countmod.nk_type, countmod.nk_less].general_induction
requires the following assumptions to be discharged
  noetherian[countmod.nk_type, countmod.nk_less].well_founded

Use of the formula
mid.ft_mid
requires the following TCCs to be proven
mid_tcc.ft_mid_TCC1
mid_tcc.ft_mid_TCC2
mid_tcc.ft_mid_TCC3

Use of the formula
division.div_ineq
requires the following TCCs to be proven
division_tcc.mult_div_1_TCC1
division_tcc.mult_div_TCC1
division_tcc.div_cancel_TCC1
division_tcc.ceil_mult_div_TCC1
division_tcc.div_nonnegative_TCC1
division_tcc.div_ineq_TCC1
division_tcc.div_minus_1_TCC1

================ SUMMARY ===================

The proof chain is complete

The axioms and assumptions at the base are:
clocksort.cnt_sort_geq
clocksort.cnt_sort_leq
clocksort.funsort_ax
division.mult_div_1
division.mult_div_2
division.mult_div_3
ft_mid_assume.ft_mid_maxfaults
multiplication.mult_pos
noetherian[EXPR, EXPR].general_induction
Total: 9

The definitions and type-constraints are:
absmod.abs
clockassumptions.okay_Readpred
countmod.count
countmod.countsize
countmod.exists_less

90
countmod.nk_noeth_pred
countmod.nk_ph_pred
mid.ft_mid
multiplication.mult
naturalnumbers.nat_invariant
Total: 10

The formulae used are:
absmod.abs_com
absmod.abs_geq
countmod.count_exists
countmod.count_exists_base
countmod.count_exists_ind
countmod.nk_ph_expand
countmod.nk_ph_lem
countmod.nk_ph_noeth_hyp
countmod.ph_case0
countmod.ph_case0k
countmod.ph_case0n
countmod.ph_case1
countmod.ph_case2
countmod.ph_case2a
countmod.ph_case2b
countmod.pigeon_hole
countmod_tcc.count_TCC1
countmod_tcc.count_TCC2
countmod_tcc.count_TCC3
countmod_tcc.count_TCC4
countmod_tcc.count_TCC5
countmod_tcc.posint_TCC1
division.div_cancel
division.div_ineq
division.mult_div
division_tcc.ceil_mult_div_TCC1
division_tcc.div_cancel_TCC1
division_tcc.div_ineq_TCC1
division_tcc.div_minus_1_TCC1
division_tcc.div_nonnegative_TCC1
division_tcc.mult_div_1_TCC1
division_tcc.mult_div_TCC1
mid2.good_greater_F1
mid2.good_less_NF
mid2_tcc.good_greater_F1_TCC1
mid2_tcc.good_greater_F1_pr_TCC1
mid2_tcc.good_less_NF_TCC1
mid2_tcc.good_less_NF_pr_TCC1
mid4.abs_q_greater
mid4.abs_q_less
mid4.ft_mid_bnd_by_good
mid4.ft_mid_greater
mid4.ft_mid_less
mid4.ft_select
mid4.maxfaults_lem
mid4.mult_hack
mid4_tcc.ft_mid_greater_TCC1
mid4_tcc.ft_mid_less_TCC1
mid_tcc.ft_mid_TCC1
mid_tcc.ft_mid_TCC2
mid_tcc.ft_mid_TCC3
multiplication.distrib_minus
multiplication.mult_com
multiplication.mult_gt
multiplication.mult_l_distrib_minus
multiplication.mult_rident
natinduction.induction
natinduction_tcc.ind_m_proof_TCC1
noetherian[countmod.nk_type, countmod.nk_less].well_founded
noetherian[naturalnumber, natinduction.less].well_founded
select_defs.count_geq_select
select_defs.count_leq_select
select_defs.select_ax
Total: 64

The completed proofs are:
absmod.abs_com_proof
absmod.abs_geq_proof
countmod.count_exists_base_pr
countmod.count_exists_ind_pr
countmod.count_exists_pr
countmod.nk_ph_expand_pr
countmod.nk_ph_lem_pr
countmod.nk_ph_noeth_hyp_pr
countmod.nk_well_founded
countmod.ph_case0_pr
countmod.ph_case0k_pr
countmod.ph_case0n_pr
countmod.ph_case1_pr
countmod.ph_case2_pr
countmod.ph_case2a_pr
countmod.ph_case2b_pr
countmod.pigeon_hole_pr
countmod_tcc.count_TCC1_PROOF
countmod_tcc.count_TCC2_PROOF
countmod_tcc.count_TCC3_PROOF
division.div_cancel_pr
division.div_ineq_pr
division.mult_div_pr
division_tcc.ceil_mult_div_TCC1_PROOF
division_tcc.div_cancel_TCC1_PROOF
division_tcc.div_ineq_TCC1_PROOF
division_tcc.div_minus_1_TCC1_PROOF
division_tcc.div_nonnegative_TCC1_PROOF
division_tcc.mult_div_1_TCC1_PROOF
division_tcc.mult_div_TCC1_PROOF
mid2.good_greater_F1_pr
mid2.good_less_NF_pr
mid2_tcc.good_greater_F1_TCC1_PROOF
mid2_tcc.good_greater_F1 pr_TCC1_PROOF
mid4.abs_q_greater_pr
mid4.abs_q_less_pr
mid4.ft_mid_acc_pres_pr
mid4.ft_mid_bnd_by_good_pr
mid4.ft_mid_greater_pr
mid4.ft_mid_less_pr
mid4.ft_select_pr
mid4.maxfaults_lem_pr

93
References


**Title:** A Verified Design of a Fault-Tolerant Clock Synchronization Circuit: Preliminary Investigations

**Authors:** Paul S. Miner

**Abstract:** Schneider [1] demonstrates that many fault-tolerant clock synchronization algorithms can be represented as refinements of a single proven correct paradigm. Shankar [2] provides a mechanical proof (using Ehdm [3]) that Schneider's schema achieves Byzantine fault-tolerant clock synchronization provided that 11 constraints are satisfied. Some of the constraints are assumptions about physical properties of the system and cannot be established formally. Proofs are given (in Ehdm) that the fault-tolerant midpoint convergence function satisfies three of these constraints. This paper presents a hardware design, implementing the fault-tolerant midpoint function, which will be shown to satisfy the remaining constraints. The synchronization circuit will recover completely from transient faults provided the maximum fault assumption is not violated. The initialization protocol for the circuit also provides a recovery mechanism from total system failure caused by correlated transient faults.