一致性
时态逻辑
安全性与进展性

用一条直线来表示系统中每一个处理器的运行，时间从左边开始到右边。每一个共享内存的操作我们把它写在处理器的直线上。两个主要的操作为“读”和“写”，用下面的表达式表示：
W(var) value
该表达式的含义为：将值value写到共享变量var中，而
R(var) value
该表达式的含义为：读取共享变量var，获取它的值value.
比如：W(x)1 表示：将1写到x中，而R(y)3表示：读取变量y的值3.
更多的操作（特别是同步操作）在需要的时候我们再来定义他的记号。为简单起见，假设所有的变量都初始化为0.
我们要特别注意一点，在高级语言中的一条语句（比如x = x + 1;）通常将会涉及到几次内存操作。如果x之前为0，则那条高级语言中的语句将变成（不考虑其它处理器）：
P1：R(x)0 W(x)1

严格一致性(Strict Consistency):

- 任意read(x)操作都要读到**最新**的write(x)的结果。依赖于绝对的全局时钟。

P1：W(x)1

P2: R(x)1 R(x)1
这表示，“处理器P1将1写到变量x中；一段时间之后处理器P2读取x的值1。然后再读取一次，获得相同的值。
我们再给出一个符合”内存严格一致性模型”的场景：
P1： W(x)1

P2：R(x)0 R(x)1
这一次，处理器P2先执行，它先读取x的值0，当它第二次读取x时却获得了处理器P1写入的x的值1。注意这两个场景能够通过将同一个程序在同一个处理器上执行两次而获得。
我们给出一个不符合“内存严格一致性模型”的场景：
P1： W(x)1

P2: R(x)0 R(x)1

顺序一致性 (Sequential Consistency):

- “（并发程序在多处理器上的）任何一次执行结果都相同，就像所有处理器的操作按照某个顺序执行，各个微处理器的操作按照其程序指定的顺序进行。”。

在一个写操作发生之前看到该写操作的结果是可能的，
P1: W(x)1

P2: R(x)1

P1: W(x)1

P2: R(x)1 R(x)2

P3: R(x)1 R(x)2

P4: W(x)2
这个场景是合法的顺序一致性内存模型的原因是，下面的交替操作在严格一致性内存模型中将会是合法的：
P1: W(x)1

P2: R(x)1 R(x)2

P3: R(x)1 R(x)2

P4: W(x)2
下面是一个不符合顺序一致性内存模型的场景：
P1: W(x)1

P2: R(x)1 R(x)2

P3: R(x)2 R(x)1

P4: W(x)2

顺序一致性在分布式算法中的应用例

- 最短路径迭代计算收敛终止判定
  - 每个线程独立反复运行Dijkstra算法，并记录第k轮有多少结点被更新；
  - 无更新且其他线程见过该线程最新一轮则进入终止准备状态；
  - 一个线程见到其他线程相关结点更新后先退出终止准备状态，再告知对方已经见过。
  - 原子性判定所有线程都终止准备状态则终止。

弱一致性 (Weak Consistency):

- 如果我们仅仅将竞争访问分为同步与非同步访问，并且同时要求符合下列条件，那么我们就得到了“弱一致性”：
- 1.对同步变量的访问是“顺序一致性”的
- 2.直到之前对所有同步变量的写操作完成之后，才允许访问这些同步变量。
- 3.直到之前对同步变量的访问完成之后，才允许我们访问（读或写）这些同步变量。

下面给出一个符合“弱一致性”的场景，显示了“弱一致性”的真正用途：

P1: W(x)1 W(x)2

P2: R(x)0 R(x)2 S R(x)2

P3: R(x)1 S R(x)2

换一句话说，根本没有要求一个处理器广播它对变量的修改，直到一个同步访问的发生。在一个基于网络而不是总线的分布式系统中，这能够极大的减少信息的互通（注意到，在现实中没有人会故意写一个具有这样行为的程序；你绝对不想去读一个别人正在更新的变量。读操作必须发生在S之后。我提到过一些同步算法，比如松弛算法，它不要求内存一致性的概念。这些算法在“弱一致性”系统中不能工作，因为在“弱一致性”系统中推迟了数据的交流直到同步点。）

因果一致性 （Casual Consistency）

- **有因果关系的写操作必须按照它们的因果关系的顺序被看到，没有因果关系的写操作可以以任意顺序被别的进程看到。**

思考题：

1. **画出一个满足顺序一致但不满足因果一致的情形**
2. **最短路径迭代计算收敛终止判定算法可否使用因果一致性**

Specifying a Simple Clock

$HCini \triangleq hr \in \{1, \ldots, 12\}$

$HCnxt \triangleq hr' = \text{IF } hr \neq 12 \text{ THEN } hr + 1 \text{ ELSE } 1$

MODULE *HourClock*

EXTENDS *Naturals*

VARIABLE hr

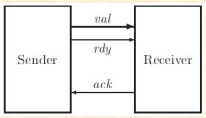
$HCini \triangleq hr \in (1 \,..\, 12)$

$HCnxt \triangleq hr' = \text{IF } hr \neq 12 \text{ THEN } hr + 1 \text{ ELSE } 1$

$HC \triangleq HCini \wedge \Box[HCnxt]_{hr}$

THEOREM $HC \Rightarrow \Box HCini$

An Asynchronous Interface



$$\begin{bmatrix} val = 26 \\ rdy = 0 \\ ack = 0 \end{bmatrix} \xrightarrow{Send\ 37} \begin{bmatrix} val = 37 \\ rdy = 1 \\ ack = 0 \end{bmatrix} \xrightarrow{Ack} \begin{bmatrix} val = 37 \\ rdy = 1 \\ ack = 1 \end{bmatrix} \xrightarrow{Send\ 4}$$

$$\begin{bmatrix} val = 4 \\ rdy = 0 \\ ack = 1 \end{bmatrix} \xrightarrow{Ack} \begin{bmatrix} val = 4 \\ rdy = 0 \\ ack = 0 \end{bmatrix} \xrightarrow{Send\ 19} \begin{bmatrix} val = 19 \\ rdy = 1 \\ ack = 0 \end{bmatrix} \xrightarrow{Ack} \cdots$$

MODULE *AsynchInterface*

EXTENDS *Naturals*
CONSTANT *Data*
VARIABLES *val*, *rdy*, *ack*

$$
\begin{array}{ll}
\mathit{TypeInvariant} \triangleq & \land\ \mathit{val} \in \mathit{Data} \\
& \land\ \mathit{rdy} \in \{0,1\} \\
& \land\ \mathit{ack} \in \{0,1\}
\end{array}
$$

$$
\begin{array}{ll}
\mathit{Init} \triangleq & \land\ \mathit{val} \in \mathit{Data} \\
& \land\ \mathit{rdy} \in \{0,1\} \\
& \land\ \mathit{ack} = \mathit{rdy}
\end{array}
$$

$$
\begin{array}{ll}
\mathit{Send} \triangleq & \land\ \mathit{rdy} = \mathit{ack} \\
& \land\ \mathit{val}' \in \mathit{Data} \\
& \land\ \mathit{rdy}' = 1 - \mathit{rdy} \\
& \land\ \text{UNCHANGED } \mathit{ack}
\end{array}
$$

$$
\begin{array}{ll}
\mathit{Rcv} \triangleq & \land\ \mathit{rdy} \neq \mathit{ack} \\
& \land\ \mathit{ack}' = 1 - \mathit{ack} \\
& \land\ \text{UNCHANGED } \langle \mathit{val}, \mathit{rdy} \rangle
\end{array}
$$

$$\mathit{Next} \triangleq \mathit{Send} \lor \mathit{Rcv}$$

$$\mathit{Spec} \triangleq \mathit{Init} \land \Box[\mathit{Next}]_{\langle \mathit{val}, \mathit{rdy}, \mathit{ack} \rangle}$$

THEOREM $\mathit{Spec} \Rightarrow \Box\, \mathit{TypeInvariant}$

A: "The moon is a satellite of the earth"
B: "The moon is rising"
C: "The moon is setting"

■ (denoting "always") and ◆ (denoting "eventually")

D: ◆ *B*	"The moon will be rising eventually"
E: ■ ◆ *B*	"The moon will be rising again and again"
F: ■ (*B* ⇒ ◆ *C*)	"Moon rise leads to moon setting"

◆ terminated

(■ ◆ sent) ⇒ (◆ delivered)

■ (*P* ⇒ ◆ *Q*)

◊□ P
□ ◊P
◊□ ◊P
□ ◊□ P

□ □ P = ?
◊◊P = ?
¬□ ¬P = ?
¬◊ ¬P = ?

An execution of a concurrent program can be viewed as an infinite sequence of states:

$\sigma = s_0 s_1 \ldots$.

Each state after s_0 results from executing a single atomic action in the preceding state. (For a terminating execution, an infinite sequence is obtained by repeating the final state.) A *property* is a set of such sequences. Since a program also defines a set of sequences of states, we say that a property *holds* for a program if the set of sequences defined by the program is contained in the property.

Informally, a safety property stipulates that some 'bad thing' does not happen during execution [2]. Examples of safety properties include mutual exclusion, deadlock freedom, partial correctness, and first-come-first-serve. In *mutual exclusion*, the proscribed 'bad thing' is two processes executing in critical sections at the same time. In *deadlock freedom* it is deadlock. In *partial cor-*

We now formalize safety.[1] Let S be the set of program states, S^ω the set of infinite sequences of program states, and S* the set of finite sequences of program states. An execution of any program can be modeled as a member of S^ω. We call elements of S^ω *executions* and elements of S* *partial executions* and write $\sigma \models P$ when execution σ is in property P. Finally, let σ_i denote the partial execution consisting of the first i states in σ.

Safety properties

For P to be a safety property, if P does not hold for an execution, then at some point some 'bad thing' must happen. Such a 'bad thing' must be irremediable because a safety property states that the 'bad thing' never happens during execution. Thus, P is a *safety property* if and only if the following holds.

Safety

$$(\forall \sigma : \sigma \in S^{\omega} : \sigma \not\models P \Rightarrow (\exists i : 0 \leqslant i : (\forall \beta : \beta \in S^{\omega} : \sigma_i \beta \not\models P))).$$

任何无限运行序列，如若违反P性质，则存在其有限前缀，使得该前缀之所有无限扩展均违反P。

Liveness properties

Informally, a liveness property stipulates that a 'good thing' happens during execution [2]. Examples of liveness properties include starvation freedom, termination, and guaranteed service. In *starvation freedom*, which states that a process makes progress infinitely often, the 'good thing' is making progress. In *termination*, which asserts that a program does not run forever, the 'good thing' is completion of the final instruction. Finally, in *guaranteed service*,[2] which states that every request for service is satisfied eventually, the 'good thing' is receiving service.

We now formalize liveness. A partial execution α is *live* for a property P if and only if there is a sequence of states β such that $\alpha\beta \models P$. A *liveness property* is one for which every partial execution is live. Thus, P is a liveness property if and only if the following holds.

Liveness

$$(\forall \alpha : \alpha \in S^{*} : (\exists \beta : \beta \in S^{\omega} : \alpha\beta \models P)).$$

任何有限运行序列均可以被无限扩展为一个满足P的无限序列。

Theorem 1. *Every property* P *is the intersection of a safety property and a liveness property.*

Proof. Let $\overline{P}$ be the smallest safety property containing P and let L be $\neg(\overline{P} - P)$. Then,

$$\begin{aligned} L \cap \overline{P} &= \neg(\overline{P} - P) \cap \overline{P} \\ &= (\neg\overline{P} \cup P) \cap \overline{P} \\ &= (\neg\overline{P} \cap \overline{P}) \cup (P \cap \overline{P}) \\ &= P \cap \overline{P} = P. \end{aligned}$$

It remains to show that L is dense, and hence a liveness property. By way of contradiction, suppose there is a nonempty open set O contained in $\neg$L and thus L is not dense. Then, $O \subseteq (\overline{P} - P)$. Consequently, $P \subseteq (\overline{P} - O)$. The intersection of two closed sets is closed, so $\overline{P} - O$ is closed and thus a safety property. This contradicts the hypothesis that $\overline{P}$ is the smallest safety property containing P. □