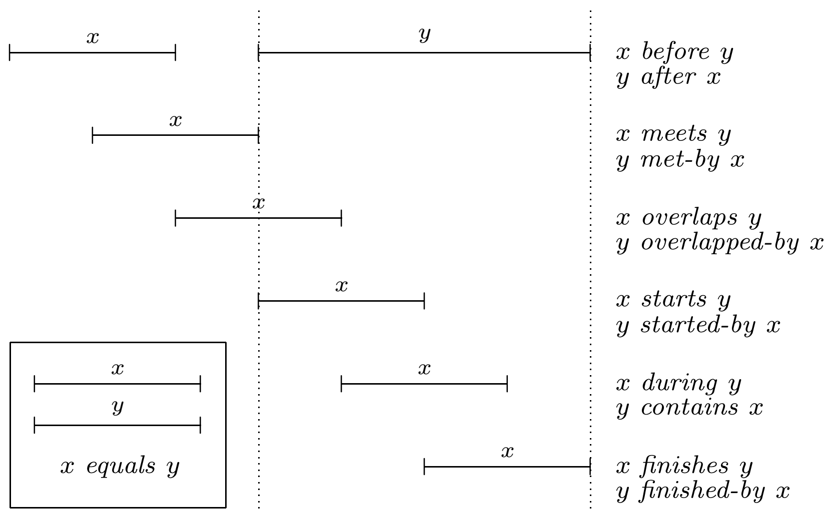
Reasoning with Temporal Constraints

From the operating instructions for a big scary machine:

- The red button has to be pressed ***before*** phase 4711
  — *or it's all going to blow up.*
- The green button has to be pressed ***during*** phase 4711
  — *or it's all going to blow up.*
- The red button has to be pressed ***after*** phase 0815
  — *or it's all going to blow up.*
- Make sure phase 0815 ***overlaps*** with phase 4711
  — *or it's all going to blow up.*

What if it get's more complicated? Can we use a computer to reason about this kind of information?

Allen's Interval Relations



Obtaining Knowledge through Inference

Transitivity. Interval relations are *transitive* in the following sense:

If we know that intervals a and b are in relation R_1 and
if we know that intervals b and c are in relation R_2,
then we can restrict the set of possible relations for a and c.

Examples:

- Given: a *starts* b and b *overlaps* c
  Infer: a *before* c or a *meets* c or a *overlaps* c
  (but certainly not a *after* c, etc.)
- Given: a *after* b and b *after* c
  Infer: a *after* c (this is transitivity in the usual sense of the word)

Transitivity table. The *transitivity table* in Allen's paper gives an overview over all possible inferences of this kind.

Temporal Constraint Networks

Constraints. Given intervals i and j, a *temporal constraint* $(i,j):R$ (where R is a set of Allen relations) says that i and j are supposed to stand in *one* of the relations in R. Example:

$$(i,j):\{\mathit{before}, \mathit{meets}, \mathit{overlaps}\}$$

TCNs. A *temporal constraint network (TCN)* over a set of intervals I is a set of constraints talking about the intervals in I.

Consistency. A TCN over a set of intervals I is called *consistent* iff we can map the left and right endpoints of each interval in I to a (real) number in such a way that all constraints are satisfied (and no interval has length 0). Example:

$$(i,j):\{\mathit{before}, \mathit{meets}\} \;\Rightarrow\; r(i) < \ell(j) \text{ or } r(i) = \ell(j),\; \ell(i) < r(i),\ \text{etc.}$$

Normalising TCNs

We can *normalise* a given TCN:

- Add inverse constraints: for $(i, j) : R$ add $(j, i) : R^{-1}$. Example:

  If $(i, j) : \{before, meets, \textit{finished-by}, equals\}$ is in the TCN, then add $(j, i) : \{after, \textit{met-by}, finishes, equals\}$.

- If there are two constraints $(i, j) : R_1$ and $(i, j) : R_2$ (for the same pair of intervals), replace them with $(i, j) : R_1 \cap R_2$. Example:

  If both $(i, j) : \{meets, starts\}$ and $(i, j) : \{starts, finishes\}$ are in the TCN, replace them with $(i, j) : \{starts\}$.

- Add $(i, i) : \{equals\}$ for every interval i.
- Add the full constraint $(i, j) : \{before, after, meets, \ldots\}$ (all 13 relations), if there is no information for (i, j) in the TCN.

A (normalised) TCN containing an *empty constraint* is inconsistent!

Checking Consistency

Singleton labellings. A normalised TCN is called a *singleton labelling* if it relates any two intervals by just *one* basic relation.

Checking a singleton labelling for consistency is easy (*how?*).

General consistency checking. A general TCN corresponds to a disjunction of singleton labellings. In principle, we can always check whether a given TCN is consistent by checking all possible singleton labellings in turn until we find one that is consistent.

Practical considerations. In practice, this is not possible. Suppose we have 10 intervals, i.e. $(10^2 - 10)/2 = 45$ relevant constraints (plus another 45 inverse constraints plus 10 trivial *equals*-constraints). Further suppose, in each of these 45 constraints we have 3 relations. Then we get $3^{45} \approx 2.95$ *sextillion* different singleton labellings!

Constraint Propagation

Transitivity again. Let $tr(r_1, r_2)$ denote the entry in the transitivity table for the interval relations r_1 and r_2. Example:

$$tr(starts, overlaps) \quad = \quad \{before, meets, overlaps\}$$

We generalise this to sets of relations:

$$constraints(R_1, R_2) \quad = \quad \{r \mid r_1 \in R_1 \ \& \ r_2 \in R_2 \ \& \ r \in tr(r_1, r_2)\}$$

Constraint propagation. Whenever we find $(i, j) : R_1$ and $(j, k) : R_2$ in a TCN, we can add $(i, k) : constraints(R_1, R_2)$. To show that a given TCN is inconsistent, we apply constraint propagation and normalise as much as possible and look for an empty constraint.

Soundness. Constraint propagation (together with normalisation) is a *sound* operation: a consistent TCN will never be turned into an inconsistent one (because we only add implied constraints).

Constraint Propagation is not Complete!

However, constraint propagation does not provide us with a *complete* algorithm to detect inconsistencies. The following is an example for an inconsistent TCN, which cannot be made more specific using constraint propagation. (*check!*)

$$\begin{array}{ll} \{\ (a,b) : \{during, contains\}, & (a,c) : \{finishes, finished\text{-}by\}, \\ \quad (a,d) : \{met\text{-}by, started\text{-}by\}, & (b,c) : \{during, contains\}, \\ \quad (b,d) : \{overlapped\text{-}by\}, & (c,d) : \{met\text{-}by, started\text{-}by\}\ \} \end{array}$$

Still, in practice, constraint propagation will *often* find *most* inconsistencies. And for application where we require completeness, at least, the number of possibilities will be greatly reduced through constraint propagation.