TOVE Digital City Programming Manual  
Part IIb: Constrained Temporal Representation and Reasoning

Mark S. Fox, [msf@eil.utoronto.ca](mailto:msf@eil.utoronto.ca)

20 March 2021

# Introduction

This report defines the constraint extension to the TOVE Digital City (TDC) temporal representation. It includes dependency directed backtracking that can be invoked when an inconsistency is reached.

In the remainder of this report, we use the following ontology prefix’s:

|  |  |
| --- | --- |
| **Prefix** | **IRI** |
| ctime | http://ontology.eil.utoronto.ca/tove/ctime# |
| time | http://www.w3.org/2006/time# |

# Constrained Time

The Constrained Time ontology introduces bounds on the time of an instance and the start and end times of an interval. There are two reasons to bound the values of an Instant and Interval:

1. **Uncertainty**: the actual time of an instance or interval is either unknown or fuzzily known, and
2. **Undetermined**: the time of an Instant or Interval has yet to be determined. For example, scheduling activities starts with bounds on times, which are then refined by the scheduling process.

Our Constrainted Time ontology introduces two classes:

* CInstant: extends time:Instant by introducing a hasMin and hasMax property, representing bounds on the time of the Instant, and
* CDateTimeInterval: extends time:DateTimeInterval by restricting hasBeginning and hasEnd to a range of CInstant, allowing for representing bounds on the beginning and end times of the interval.

Figure 1 depicts the constrained time ontology pattern accessible at: <http://ontology.eil.utoronto.ca/tove/ctime.owl>.

ctime:CDateTimeInterval represents a time interval that has a beginning (time:hasBeginning) instant and an end (time:hasEnd) instant. ctime:CDateTimeInterval differs from OWL-Time time:DateTimeInterval in that both time:hasBeginning and time:hasEnd are functional and the range is restricted to a ctime:CInstant (rather than an time:Instant).



Figure 4: Constrained Time Pattern

ctime:CInstant differs from time:Instant in that rather than specifying a single time point, it has two properties, ctime:hasMin and ctime:hasMax, that specify the min and max bounds on the value of the instant. Both properties are functional and their range is restricted to a time:Instant.

All Instants in the ontology is represented using time:DateTimeDescription where each component of Date and Time are specified as separate properties. Table 1 depicts the properties that comprise time:DateTimeDescription.

Finally, a ctime:CDateTimeInterval can have a duration specified by ctime:CDurationDescription. This class differs from time:DurationDescription in that it has two properties: ctime:hasMin and ctime:hasMax that specify a bound on the value of the duration. The range of each is a time:DurationDescription.

Table 3: ctime Extended Classes

|  |  |  |
| --- | --- | --- |
| **Class** | **Property** | **Value Restriction** |
| ctime:CDateTimeInterval | rdfs:subClassOf | time:TemporalEntity |
| time:hasBeginning | exactly 1 CInstant |
| time:hasEnd | exactly 1 CInstant |
| time:intervalBefore | only ctime:CDateTimeInterval |
| time:intervalAfter | only ctime:CDateTimeInterval |
| time:intervalContains | only ctime:CDateTimeInterval |
| time:intervalDuring | only ctime:CDateTimeInterval |
| time:intervalOverlaps | only ctime:CDateTimeInterval |
| time:intervalOverlappedBy | only ctime:CDateTimeInterval |
| time:intervalMeets | only ctime:CDateTimeInterval |
| time:intervalMetBy | only ctime:CDateTimeInterval |
| time:intervalStarts | only ctime:CDateTimeInterval |
| time:intervalStartedBy | only ctime:CDateTimeInterval |
| time:intervalFinishes | only ctime:CDateTimeInterval |
| time:intervalFinishedBy | only ctime:CDateTimeInterval |
| time:intervalEqual | only ctime:CDateTimeInterval |
| ctime:CInstant | rdfs:subClassOf | time:TemporalEntity |
| time:before | only ctime:CInstant |
| time:end | only ctime:CInstant |
| ctime:hasMin | exactly 1 time:Instant |
| ctime:hasMax | exactly 1 time:Instant |
| ctime:CDurationDescrption | rdfs:subClassOf | time:TemporalEntity |
| ctime:hasMin | exactly 1 time:DurationDescription |
| ctime:hasMax | exactly 1 time:DurationDescription |

# Constrained Time (http://ontology.eil.utoronto.ca/dt/code/ctime.py)

Before using Constrained Time functions, the dependency network must be initialized. This is accomplished by:

dn = ctime.DNetwork()

dn would then be used by the Dependency Directed Backtracking (DDB) functions defined in Section 7.

## Utility Temporal Functions

|  |  |
| --- | --- |
| **createCDTI(ns=None)** | |
| *Instantiates a CDateTimeInterval and returns the instance. It instantiates the entire pattern as depicted in Figure 1. Note: this function must be used to create a CDateTimeInterval if dependency directed backtracking is to be used. Once created instances of CInstants, Instants and DateTimeDescriptions in the pattern must not be replaced. Values of the DateTimeDescription data properties can be changed by using the replaceWith function, defined in the next section.*  *If dependency directed backtracking is not being used, you can manipulate the pattern as you wish.* | |
| **ns** | The namespace in which the instances of the CDateTimeInterval pattern are to be instantiated. |
| **Returns** | instance of CDateTimeInterval – Note that all data property values of DateTimeDescription instances (leaf nodes of the pattern) are set to zero. |
|  | |
| **replaceWith(dtd1, dtd2, ctype)** | |
| *It replaces the values of dtd1 with the values of dtd2. This function is required by dependency directed backtracking. It is the only way to change the values of a DateTimeDescription embedded within a CDateTimeInterval pattern.*  ***Side effect:*** *the previous values of dtd1 are copied and stored in the DDB data structure, along with the type of change being made, e.g., assertion, inference.* | |
| **dtd1** | time:DateTimeDescription instance |
| **dtd2** | time:DateTimeDescription instance |
| **ctype** | A string that specifies the source of change. “assertion” is a reserved word to be used when asserting dtd2 as the new value for dtd1. |
| **Returns** | None |
|  | |
| **printCDTI(cdti, prnt=True)** | |
| *Prints the entire ctime:CDateTimeInterval pattern. The value of hasBeginning and hasEnd are printed using printCInstant.* | |
| **cdti** | time:DateTimeDescription instance |
| **prnt** | Boolean, if true then the datetime string is printed |
| **Returns** | *YYYY-MM-DDThh:mm:ss* |
|  | |
| **printCInstant(cinst)** | |
| *Prints the instance of ctime:CInstant. The value of hasMin and hasMax are printed usingYYYY-MM-DDThh:mm:ss format.* | |
| **cinst** | ctime:CInstant instance to be printed. |
| **Returns** | None |

## Temporal Arc Consistency Functions

|  |  |
| --- | --- |
| **cintervalConsistent(cint)** | |
| *Determines whether an CDateTimeInterval is internally, temporally consistent, i.e., if the hasBeginning min/max and hasEnd min/max are consistent*   1. hasBeginning.hasMin <= hasBeginning .hasMax 2. hasEnd.hasMin <= hasEnd.hasMax 3. has.Beginning.hasMin <= hasEnd.hasMax | |
| **cint** | ctime:CDateTimeInterval instance |
| **Returns** | Boolean: True if consistent, False if not. |
|  | |
| **cIntervalAC(cint)** | |
| *Checks to see if the interval is internally consistent. If the interval hasBeginning min/max and hasEnd min/max are consistent wrt cint.hasBeginning being before cint.hasEnd. It adjusts the values according to the following:*   1. *IF cint.hasBeginning.hasMax > cint.hasEnd.hasMax THEN replaceWith cint.hasBeginning.hasMax with cint.hasEnd.hasMax* 2. *IF cint.hasEnd.hasMin < cint.hasBeginning.hasMin THEN replaceWith cint.hasEnd.hasMin with cint.hasBeginning.hasMin* | |
| **cint** | ctime:CDateTimeInterval instance |
| **Return** | set of intervals that were modified |
|  | |
| **intervalBeforeAC(cint1, cint2**) | |
| *Checks to see if the hasBegining min/max and hasEnd min/max are consistent wrt to cint1 being intervalBefore cint2. If not, it adjusts the values according to the following:*   1. *IF cint1.hasEnd.hasMin > cint2.hasBeginning.hasMin THEN replaceWith cint2.hasBeginning.hasMin with cint1.hasEnd.hasMin* 2. *IF cint2.hasBeginning.hasMax < cint1.hasEnd.hasMax THEN replaceWith cint1.hasEnd.hasMax with cint2.hasBeginning.hasMax* | |
| **cint1** | ctime:CDateTimeInterval instance |
| **cint2** | ctime:CDateTimeInterval instance |
| **Return** | set of intervals that were modified |
|  | |
| **intervalAfterAC(cint1, cint2**) | |
| *Implemented as intervalBeforeAC(cint2, cintt1)* | |
|  | |
| **intervalOverlapsAC(cint1, cint2**) | |
| *Checks to see if the hasBegining min/max and hasEnd min/max are consistent wrt to cint1 being intervalOverlaps cint2. If not, it adjusts the values according to the following:*   1. *IF cint1.hasEnd.hasMin < cint2.hasBegining.hasMin THEN replaceWith cint1.hasEnd.hasMin with cint2.hasBegining.hasMin* 2. *IF cint1.hasEnd.hasMin > cint2.hasBegining.hasMin THEN replaceWith cint2.hasBegining.hasMin with cint1.hasEnd.hasMin* | |
| **cint1** | ctime:CDateTimeInterval instance |
| **cint2** | ctime:CDateTimeInterval instance |
| **Return** | set of intervals that were modified |
|  | |
| **intervalOverlappedByAC(cint1, cint2**) | |
| *Implemented as intervalOverlapsAC(cint2, cintt1)* | |
|  | |
| **intervalContainsAC(cint1, cint2)** | |
| *Checks to see if the hasBegining min/max and hasEnd min/max are consistent wrt to cint1 being intervalContains cint2. If not, it adjusts the values according to the following:*   1. *IF cint2.hasBegining.hasMin < cint1.hasBegining.hasMin THEN replaceWith cint2.hasBegining.hasMin with cint1.hasBegining.hasMin* 2. *IF cint2.hasEnd.hasMax > cint1.hasEnd.hasMax THEN replaceWith cint2.hasEnd.hasMax with cint1.hasEnd.hasMax* | |
| **cint1** | ctime:CDateTimeInterval instance |
| **cint2** | ctime:CDateTimeInterval instance |
| **Return** | set of intervals that were modified |
|  |  |
|  |  |

# Temporal Graph Consistency Functions

The temporal graph consistency functions propagate the effects of temporal arc consistency across the temporal relations. It propagates the changes until:

1. Quiescence is reached, i.e., no more changes are made, or
2. An inconsistency is reached.

If an inconsistency is reached, dependency directed backtracking can be used to undo the latest assertion. See Section 6.4 for the “undo” functions.

|  |  |
| --- | --- |
| **TAC\_Graph(changedInterval)** | |
| *Iteratively propagates the effect of a changed CDateTimeInterval across the temporal network. Starting with changedInterval, it performs internal propagation using cIntervalAC, then for each temporal relation that exists for the changedInterval it performs the corresponding temporal relation arc consistency. If any CDateTimeInterval is changed, it is added to change set that TAC-Graph then iterates over. After each arc consistency operation, consistency of the current interval is checked.* | |
| **changedInterval** | Instance of CDateTimeInterval that contains one or more DTDs that have been changed. If it has a value of None, then all instances of CDateTimeInterval will be used to start the propagation. |
| **Returns** | None, if propagation reaches quiescence, otherwise returns the inconsistent interval. |

# Dependency Directed Backtracking (DDB)

Dependency directed backtracking keeps track of every change to a DateTimeDescription that was made with the replaceWith function, along with the source of the change. It provides an “undo” function that can sequentially undo, each change to a DTD, one at a time, or if provided with a node in the dependency graph, will continue undo’ing DTD assignments until that node is reach (and undoes that node too). The lastAssertion function is the only way to find a node to backtrack to. It returns the node of the last assertion used to change the value of a DTD.

|  |  |
| --- | --- |
| **Dnetwork.print()** | |
| *Prints the dependency stack.* | |
| **Returns** | None |
|  | |
| **Dnetwork.lastAssertion()** | |
| *Returns the node in the dependency network containing the last assertion of a DTD value.* | |
| **Returns** | Dnode instance |
|  | |
| **Dnetwork.undo(dnode)** | |
| *If dnode == None, then undoes the latest DTD change. Otherwise it undoes all DTD changes up to including the DTD designated by dnode. Note that dnode is NOT an instance of DateTimeDescription.* | |
| **dnode** | Dnode instance – accessible only using lastAssertion() |
| **Returns** | None |

# References

Cox, S., and Little, C. (2020), “Time Ontology in OWL”, OGC Document Number OGC 16-071r3, <https://www.w3.org/TR/owl-time/>. Downloaded 3 February 2021.

Hobbs, J. R., & Pan, F. (2006). Time ontology in OWL. *W3C working draft*, *27*, 133.

Lamy JB. Owlready: Ontology-oriented programming in Python with automatic classification and high level constructs for biomedical ontologies. Artificial Intelligence In Medicine 2017;80:11-28