following arcs are possible by rule (i) in the relation definition:

$$X_R \triangleq \{(\alpha_1, \alpha_3), (\alpha_1, \alpha_4), (\alpha_2, \alpha_3), (\alpha_2, \alpha_4), (\alpha_3, \alpha_4), (\alpha_4, \alpha_3)\}.$$

Rule (ii) of the relation construction is related to the formula φ and requires that as $\langle R \rangle (\mathtt{p} \to \mathtt{q}) \in SF(\varphi)$, if $(\mathtt{p} \to \mathtt{q})$ holds at α_j, the statement $\langle R \rangle (\mathtt{p} \to \mathtt{q})$ must hold in α_i in order to have $(\alpha_i, \alpha_j) \in X_R$.

The implication $(\mathtt{p} \to \mathtt{q})$ holds in atoms α_3 and α_4, whereas $\langle R \rangle (\mathtt{p} \to \mathtt{q})$ holds in atoms α_1 and α_3. Rule (iii) states that if $\langle R \rangle (\mathtt{p} \to \mathtt{q})$ holds in α_i, the implication $(\mathtt{p} \to \mathtt{q})$ must hold in α_j.

This reduces the relation X_R to only two arcs, $\{(\alpha_1, \alpha_3), (\alpha_3, \alpha_4)\}$. Here the atom α_4 is an open leaf, as it does not require any modal formula[5] to be true. As there is a path (although only traversing one arc) from the initial atom α_1 to an open leaf α_4 (that is: an accepting path), a counterexample has been found to the validity of φ and thus the algorithm correctly concludes $\mathcal{M} \nvDash \varphi$, as obviously $(\mathtt{p} \to \mathtt{q})$ does not hold in s_2.

The search for accepting paths can be arranged in a depth-first manner from the initial atoms to their successors, using backtracking techniques. Some storage must be available for the nodes encountered during the search in order to detect cycles. In practice, the choice of the storage facility seems to favor hashtables, where the hash function does not need to take all of the components of the node into account. The reason for this is that the valuation for the atomic proposition determined by the w-component already determines the values of all the boolean formulas in m. Thus from m, storing only the value of those formulas that contain the $\langle R \rangle$-operator suffices.

During the search it is also necessary to identify closed atoms and mark these during backtracking — if a marked atom appears, the search procedure will know not to recurse there again. Also other improvements based on the corresponding decision procedure are available. These are direct consequences of the improvements available for the basic decision procedure of modal logic.

The search can normally be terminated at the discovery of the first accepting path, as only one is needed to make the counterexample. With an appropriate ordering on the child nodes of the current node, finding a counterexample can be quite quick (thus it might be useful to experiment with a couple of different ordering if the optimal one is not known), even though the worst-case scenario naturally includes traversing the entire forest. This occurs whenever φ is valid: there is no counterexample at all to be found.

1.2.2 Linear Temporal Logic (LTL)

LTL is a temporal logic on *natural models* , that is: models where every step on a computation sequence has an unambiguous successor and nothing in between. The

[5] Here, a modal formula contains also at least one modal operation in addition to possible boolean connectives.

general syntax of LTL is

$$\mathbf{LTL} ::= \mathcal{P} \mid \bot \mid (\mathbf{LTL} \rightarrow \mathbf{LTL}) \mid (\mathbf{LTL}\,\mathbf{U}^{+}\mathbf{LTL}) \mid (\mathbf{LTL}\,\mathbf{U}^{-}\mathbf{LTL}).$$

The operator $\mathbf{U}^{+}$ is a "future until" operator demanding that the preceding formula will hold until the latter becomes valid in some future state. The operator $\mathbf{U}^{-}$ is a "since" operator, referring to past time with the following semantics: $w_0 \models (\varphi\,\mathbf{U}^{-}\psi)$ if and only if $\exists w_1 \in U$ such that $w_1 < w_0 \wedge w_1 \models \psi$ when for all intermediate states $w_2 \in U$ where $w_1 < w_2$ and $w_2 < w_0$, $w_2 \models \varphi$.

It is worthwhile to note that each LTL-formula can be translated to an equivalent first order formula. Another interesting feature in LTL is its *separation property*: a temporal formula is said to be

- *pure future* if it has the form $(\varphi\,\mathbf{U}^{+}\psi)$, and neither φ nor ψ contains the operator $\mathbf{U}^{-}$,
- *pure present* if it does not contain any occurrence of either of the operators $\mathbf{U}^{+}$ and $\mathbf{U}^{-}$, and
- *pure past* if it has the form $(\varphi\,\mathbf{U}^{-}\psi)$, and neither φ nor ψ contains the operator $\mathbf{U}^{+}$.

For each LTL-formula, there is an equivalent *separated formula*, which is a formula consisting of boolean combinations of past, present and future formulas, without any nesting of the "until" operators.

Also, every LTL-formula has an equivalent formula, which does not use any past temporal operators. Thus for the sake of simplicity, only the *future fragment* of LTL is attended to in this discussion — the *past-temporal operators* such as the "since" operator $\mathbf{U}^{-}$ are not addressed further in this text.

To check a whether a formula is valid for all possible sequences, it again suffices to examine whether the negation is valid even for just one sequence, i.e. search for a counterexample to the validity of the original specification formula. It can be shown that examining the validity of a LTL formula φ in a model $\mathcal{M}$ is in a sense equivalent[6] to finding a directed Hamiltonian path[7] in a graph G. The below algorithm is presented in full detail by Clarke, Grumberg, and Peled in [CGP99].

The grammar of LTL is rewritten for the forthcoming model checking procedure; each formula can be thought to consist of a conjunction of eventualities[8] of atomic propositions $\mathbf{F}^{*}\mathrm{p}$, and global implications $\mathbf{G}^{*}(\mathrm{p} \rightarrow \mathbf{XG}^{*}\neg\mathrm{p})$. This is in positive conjunctive normal form:

$$\mathbf{F}^{*}\mathrm{p}_1 \wedge \ldots \wedge \mathbf{F}^{*}\mathrm{p}_n \wedge \mathbf{G}^{*}(\mathrm{p}_1 \rightarrow \mathbf{XG}^{*}\neg\mathrm{p}_1) \wedge \ldots \mathbf{G}^{*}(\mathrm{p}_n \rightarrow \mathbf{XG}^{*}\neg\mathrm{p}_n).$$

For an arbitrary directed graph $G = (V, E)$, acting as the foundation of the model $\mathcal{M}$, the number of nodes $v \in V$ is directly bound to the number of atomic propositions

[6] There exists a reduction from HAMILTON PATH to the model checking problem.

[7] A Hamiltonian path is a sequence of edges that forms a path visiting each vertex of the graph exactly once.

[8] An eventuality is any formula that requires something to be valid in a future state.

that appear in the formula φ of the above form that is to be checked for validity in a model: $|V| = n$. There is a node $v_i \in V$ for each $\mathtt{p}_i \in SF(\varphi)$, where $SF(\varphi)$ is again the set of subformulas of φ.

Two additional nodes, a source $s \in V$ for which $\forall v_i \in V : (s, v_i) \in E \wedge (v_i, s) \notin E$, and a sink $t \in V$ for which $\forall v_i \in V : (v_i, t) \in E \wedge (t, v_i) \notin E$, are attached to V. The source node s acts as the initial node, i.e. the starting point of the Hamiltonian path, from which all of the nodes $v_i \in V$ are reached by an edge in E. Thus any node $v_i \in V$ may be the first node to visit on the path. Also, t must be the end point of the Hamiltonian path, as it certainly cannot appear in the middle of the path, as there are no outgoing vertices from it. To make the transition relation total (allowing infinite computation sequences), a self-loop (t, t) is needed for the sink node.

Now define $\mathcal{M} = (U, \mathcal{I}, w_0)$ as follows:

- the set of states $U = V \cup \{s, t\}$, where $s, t \notin V$; initial state $w_0 = s$
- $\mathcal{I}(R) = E \cup \{(s, v_i) \mid v_i \in V\} \cup \{(v_i, t) \mid v_i \in V\} \cup \{(t, t)\}$
- $\mathcal{I}(\mathtt{p})$ for all atomic propositions $\mathtt{p} \in \mathcal{P}$ is such that
  - $v_i \in \mathcal{I}(\mathtt{p}_i)$ if and only if $1 \leq i \leq |V|$
  - $v_i \notin \mathcal{I}(\mathtt{p}_j)$ if and only if $1 \leq i,\ j \leq |V|,\ i \neq j$
  - $\forall 1 \leq i \leq |V| :\ s \notin \mathcal{I}(\mathtt{p}_i)$
  - $\forall 1 \leq i \leq |V| :\ t \notin \mathcal{I}(\mathtt{p}_i)$

Now φ is valid (i.e. satisfiable) on some path clearly if and only if there exists an infinite computation sequences for $\mathcal{M}$ starting at s and visiting all states v_i exactly once, ending in the self-loop of t. Note that $\mathcal{M}$ is now a linear and serial model.

Here the size of the model is bound by the size of the formula. For cases where the formula is much smaller than the model, another approach is necessary. An algorithm that relies on explicit tableau construction by Lichtenstein and Pnueli is summarized by Clarke, Grumberg and Peled in [CGP99]. The algorithm decomposes the formula into a tableau, from which a model can be obtained if and only if the formula is satisfiable.

In this algorithm, the set of temporal operators needed is $\mathbf{X}$ and $\mathbf{U}^*$ Again, the others may be defined by the means of these two: $\mathbf{F}^*\psi \triangleq (\top\, \mathbf{U}^*\psi)$ and $\mathbf{G}^*\psi \triangleq \neg\mathbf{F}^*\neg\psi$.

The *closure* of a formula φ is denoted by $CL(\varphi)$ and defined as the minimal set of formulas containing φ itself where the following applies for $\psi_i \in SF(\varphi)$:

(i) $\neg\psi \in CL(\varphi)$ if and only if $\psi \in CL(\varphi)$

(ii) if $\psi_1 \vee \psi_2 \in CL(\varphi)$, then both $\psi_1, \psi_2 \in CL(\varphi)$

(iii) if $\mathbf{X}\psi \in CL(\varphi)$, then $\psi \in CL(\varphi)$

(iv) if $\neg\mathbf{X}\psi \in CL(\varphi)$, then $\mathbf{X}\neg\psi \in CL(\varphi)$

(v) if $(\psi_2\, \mathbf{U}^*\psi_1) \in CL(\varphi)$, then all $\psi_1, \psi_2, \mathbf{X}(\psi_2\, \mathbf{U}^*\psi_1) \in CL(\varphi)$

The size of the resulting closure set can be shown to be $\mathcal{O}(|\varphi|)$. Similarly to the construction for modal logic in Section 1.2.1, an atom is defined for a model $\mathcal{M}$ and a formula φ as a pair $\alpha_i = (w_i, c_i)$, where $w_i \in U$ and $c_i \in CL(\varphi) \cup \mathcal{P}$ is a maximal propositionally consistent set of formulas that agrees with the valuations of the atomic propositions at w_i. This can be formalized as follows:

- $\forall \mathtt{p} \in \mathcal{P} : \mathtt{p} \in c_i$ if and only if $w_i \in \mathcal{I}(\mathtt{p})$
- $\forall \psi \in CL(\varphi) : \psi \in c_i$ if and only if $\neg\psi \notin c_i$
- $\forall (\psi_1 \vee \psi_2) \in CL(\varphi) : (\psi_1 \vee \psi_2) \in c_i$ if and only if either $\psi_1 \in c_i$ or $\psi_2 \in c_i$
- $\forall \neg\mathbf{X}\psi \in CL(\varphi) : \neg\mathbf{X}\psi \in c_i$ if and only if $\mathbf{X}\neg\psi \in c_i$
- $\forall (\psi_2\, \mathbf{U}^* \psi_1) \in CL(\varphi)$, $(\psi_2\, \mathbf{U}^* \psi_1) \in c_i$ if and only if either $\psi_1 \in c_i$ or both $\psi_2 \in c_i$ and $\mathbf{X}(\psi_2\, \mathbf{U}^* \psi_1) \in c_i$

Again, a graph $G = (V, E)$ is constructed using the atoms as nodes,$\alpha \in V$, very similarly to the atom forest of the previous section. The transition relation between two atoms $\alpha = (w, c)$ and $\alpha' = (w', c')Q$, , i.e. the edges $(\alpha, \alpha') \in E$ of the graph, is defined as follows:

- $(\alpha, \alpha') \in E$ if and only if $(w, w') \in \mathcal{I}(R)$ and
- $\forall \mathbf{X}\psi \in CL(\varphi) : \mathbf{X}\psi \in c$ if and only if $\psi \in c'$

An *eventuality sequence* is defined as an infinite path π in a graph G such that if for some atom $\alpha = (w, c)$ on the path π, $(\psi_2\, \mathbf{U}^* \psi_1) \in c$, then there must exist another atom $\alpha' = (w', c')$ such that $\psi_1 \in c'$. In other words, all "until" formulas on the path π must be possible to satisfy as ψ_1 will eventually be valid.

This enables the formulation of the following lemma from [CGP99], remembering that initial atoms are such atoms where the state w in an initial state (if unambiguous, w_0) and the verified formula φ is in c. Also note that the formula verified by this algorithm is the negation of the original specification formula, as the algorithm searches for some counterexample. As one computation sequence validating the negation of the specification formula is a sufficient counterexample to the validity of the specification, the path quantifier $\mathbf{E}$ is used in the lemma. However strange the notation may seem when used in an LTL formula, the symbol $\mathbf{E}$ is used to denote initial validity and correspondingly $\mathbf{A}$ for universal validity.

Lemma 1.1. $\mathcal{M}, w_0 \models \mathbf{E}\,\varphi$ if and only if there exists an eventuality sequence starting at an initial atom $\alpha = (w_0, c_0)$.

The proof of this, as well as the proof of the following lemma, is given in [CGP99].

The *strongly connected components* of the generated graph need to be calculated in order to examine the fulfillment of the eventualities. An SCC of atoms is said to be *self-fulfilling* if for any atom $\alpha = (w, c)$ of that particular SCC for which $(\psi_2\, \mathbf{U}^* \psi_1) \in c$, there exists some $\alpha' = (w', c')$ in the same SCC such that $\psi_1 \in c'$. This leads to another lemma from [CGP99]:

Lemma 1.2. An eventuality sequence starts from an atom $\alpha = (w, c)$ if and only if there there is a path in the atom graph G such that a self-fulfilling SCC can be reached by it from α.

To evaluate the satisfiability of the formula φ in the model $\mathcal{M}$ used in this calculation, it is only required by Lemma 1.2 that some self-fulfilling SCC is reachable from one of the initial atoms. Thus $\mathcal{M} \models \varphi$ if and only if there exists at least one path from an initial atom (w_0, m_0) to a self-fulfilling SCC of the atom forest. This is so, as a natural model given by such a sequence of atoms either ends in a terminal node or circulates the entire self-fulfilling SCC infinitely. When a maximal SCC is identified, the required and fulfilled eventualities of its nodes can be collected. If the set of those eventualities required is a subset of the set of those fulfilled, the SCC is self-fulfilling by definition.

Example 7: An atom forest for an LTL formula

The formula to be verified is an LTL formula $\neg\varphi = \neg((\mathtt{p} \vee \mathtt{r})\,\mathbf{U}^*\mathbf{X}\mathtt{q}))$ and thus the negation needed for the construction is $\varphi = ((\mathtt{p} \vee \mathtt{r})\,\mathbf{U}^*\mathbf{X}\mathtt{q}))$. The model of Figure 1.8 is the model used for the atom graph construction.

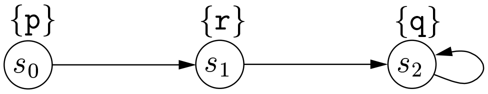

FIGURE 1.8: A simple non-branching Kripke-model $\mathcal{M} = (U, \mathcal{I}, w_0)$ with $\mathcal{I}(\mathtt{p}) = \{s_0\}$, $\mathcal{I}(\mathtt{r}) = \{s_1\}$, and $\mathcal{I}(\mathtt{q}) = \{s_2\}$

From the set of subformulas $SF(\varphi) = \{\mathtt{p}, \mathtt{q}, \mathtt{r}, \mathbf{X}\mathtt{q}, \mathtt{p} \vee \mathtt{r}, \varphi\}$, the closure can be obtained:

$$
\begin{aligned}
CL(\varphi) \;=\; \{\;& \mathtt{p},\, \neg\mathtt{p},\, \mathtt{q},\, \neg\mathtt{q},\, \mathtt{r},\, \neg\mathtt{r},\, \mathbf{X}\mathtt{q},\, \neg\mathbf{X}\mathtt{q},\, \mathbf{X}\neg\mathtt{q}, \\
& \mathtt{p} \vee \mathtt{r},\, \neg(\mathtt{p} \vee \mathtt{r})\,\mathbf{X}((\mathtt{p} \vee \mathtt{r})\,\mathbf{U}^*\mathbf{X}\mathtt{q}), \\
& \neg\mathbf{X}((\mathtt{p} \vee \mathtt{r})\,\mathbf{U}^*\mathbf{X}\mathtt{q}),\, \mathbf{X}\neg((\mathtt{p} \vee \mathtt{r})\,\mathbf{U}^*\mathbf{X}\mathtt{q}),\, \varphi\neg\varphi \quad \}
\end{aligned}
$$

From these, the maximal propositionally consistent sets are constructed as earlier, again treating temporal formulas as they were atomic propositions.

All of these sets are combined with both of the states of the model to obtain the atoms. Of these, those that do not agree with the state on the valuation of atomic propositions are excluded, possibly already during the construction phase. With the remaining admissible atoms, an accessibility relation is defined by using the five rules above. The initial atoms are identified as possible starting points for the eventuality sequences.

Another example of LTL model checking by this algorithm is presented in [CGP99].

The algorithm for LTL model checking presented by Clarke and Schlingloff in [CS01] has much in common with the respective satisfiability algorithm. Again, a forest of atoms is defined for φ and $\mathcal{M}$. The relation X_R between admissible atoms is redefined for LTL as follows[9]:

Definition 1.3. $X_R(w,m),(w',m'))$ applies if all of the following applies:

(i) $(w,w') \in \mathcal{I}(\mathbf{X})$

(ii) if $\mathbf{X}\psi \in SF(\varphi)$ and $\psi \in m'$, then $\mathbf{X}\psi \in m$.

(iii) if $\mathbf{X}\psi \in m$, then $\psi \in m'$

(iv) $\exists \mathbf{X}\psi \in m$

(v) if $\mathbf{F}^*\psi \in SF(\varphi)$ then $\mathbf{F}^*\psi \in m$ if and only if either $\psi \in m$ or $\mathbf{XF}^*\psi \in m$

The rules (i) to (iv) here are very similar to the ones given in Definition 1.2 in the previous section for modal logic; $\langle R\rangle$ has been replaced with $\mathbf{X}$. The fifth rule enforces the recursion axiom $\vdash \mathbf{F}^*\psi \leftrightarrow \psi \vee \mathbf{XF}^*\psi$. Note that there are no rules defined to handle the dual $\langle R\rangle$, $[R]$. Again the goal is to find an accepting path through the forest constructed from the atoms arising from the above definition, only accepting those paths that have *all eventualities* fulfilled.

Finding maximal self-fulfilling SCCs

The construction of maximal strongly connected components for model checking is best done with *Tarjan's algorithm*, which uses enumeration of strong components in a graph while backtracking from a depth-first search. The pseudo-code presented here is from [CS01] and intended to be used on the atom forest of the latter $SF(\varphi)$-based construction presented, but can with modification also suit the earlier construction basing on $CL(\varphi)$. The function `children` is assumed to follow the accessibility relation of the graph of atoms used in either case. First, the procedure dealing with roots of the found (SCCs is provided, and after that, the actual depth-first search routine.

```
procedure processRoot (atom α)

    <set> required := ∅ // SET OF REQUIRED EVENTUALITIES
    <set> fulfilled := ∅ // SET OF FULFILLED EVENTUALITIES

    repeat
        β := pop(SCC)
        storage[β] := ∞ // PUT "BELOW" EVERYTHING ELSE
        required := required ∪ {ψ1 | (ψ2 U+ ψ1) ∈ β}
        fulfilled := fulfilled ∪ {ψ | ψ ∈ β}
    until (α = β) // ALL ELEMENTS IN THE SCC OF α ARE PROCESSED
    if (required ⊆ fulfilled) // THE SCC OF α IS SELF-FULFILLING
        print "φ is satisfiable in M"
        exit
```

[9] Keeping in mind that $\mathbf{F}^*\varphi \triangleq (\top \mathbf{U}^*\varphi)$ and $\mathbf{X}\varphi$ means that φ is valid is a successor, this is easier to relate to the grammar given for LTL

```
procedure dfs(node α) // DEPTH FIRST SEARCH

  if (storage does not contain α) // A "FRESH" NODE
    <integer> currentDepth = recursionDepth
    recursionDepth++ // INCREMENT DEPTH BY ONE
    // INITIAL VALUE AT CURRENT RECURSION DEPTH
    storage[α] = currentDepth
    push(SCC, α) // TO THE TOP OF THE STACK
    // EXAMINE THE ACCESSIBILITY RELATION E:
    <set> successorAtoms := successor(α)

    for all (β ∈ successorAtoms) do
      dfs(β)
      // CHECK WHETHER β IS "ABOVE" α
      storage[α] = min(storage[α], storage[β])
    // IF NOTHING WAS ABOVE ⇒ α IS A ROOT OF AN SCC

    if (storage[α] = currentDepth)
      processRoot(α)
```

The `dfs` procedure together with the procedure `processRoot` constitute an implementation of Tarjan's algorithm including a test for self-fulfillment of the located SCCs. In the forward phase, the algorithm recursively traverses the tree of all atoms reachable from an atom α, storing these on a stack. During backtracking the check is made whether α is the root of an SCC: if there are no cycles starting from above α (i.e. a node β which is both "below" and "above" α in the traversing order), α is a root of an SCC consisting of all the nodes below it, that is: all the nodes stored above it on the stack.

When a SCC is found, it is inspected for self-fulfillment by iterating over all of the atoms of the SCC (i.e. anything above α on the stack and finally the root itself). This involves building two sets of eventualities: those required and those fulfilled, and checking whether the latter includes the former as a subset.

The hashtable is initialized to some invalid value, and as an atom is first encountered. Its recursion depth is stored as the initial value for that atom in the hashtable. The higher in the traversing tree an atom is, the smaller is its hashtable entry. After returning from a recursive call to an successor atom, the hashtable entries of the successor and the root candidate are compared. The smaller of those two values is set as the hashtable entry of the root candidate. Therefore, if the hashtable entry of the root candidate differs from its actual recursion depth after making all of the recursive calls of the successor atoms, it cannot be a root of an SCC. This is so because it has a (possibly not immediate) successor that is also a predecessor.

A main procedure needs to call `dfs` once per each initial atom, i.e. a pair (w_0, m_0) where w_0 is the initial state of $\mathcal{M}$ and $\varphi \in m_0$. For the algorithm to report non-satisfiability, it must go through all the initial atoms and all atoms reachable from them.

Computational complexity of LTL model checking

The algorithm for LTL model checking described above is exponential with respect to the number of formulas with the $\mathbf{U}^+$ operator, as the sets of those formulas determine the propositionally consistent sets. With respect to the size of the Kripke-model $\mathcal{M}$, the algorithm is linear, $\mathcal{O}(\mathcal{M})$. It has been shown that LTL model checking with the past-temporal operators included is complete in **PSPACE** with respect to $|\varphi|$, and **NLOGSPACE** in size of the model $\mathcal{M}$.

The high complexity in regard to the size of the formula is normally not an obstacle, as in practice specification formulas tend to be quite short. Problems arise from the state space explosion: even with a linear complexity in regard to the model, the number of atoms to traverse in the algorithm can be quite large. Only magnitudes such as 10^5 to 10^6 for the number of reachable atoms are technically possible with current technology, according to Clarke and Schlingloff [CS01]. However, methods exist to exceed such limits but are not discussed in this context.

Another algorithm, with explicit tableau construction is presented by Clarke, Grumberg and Peled in [CGP99].

1.3 Model Checking for Propositional μ-Calculus

Propositional μ-calculus (μTL) is a very expressive logic, capable of representing everything expressible by CTL*, which is a language combining CTL and LTL that allows temporal operators to be nested without placing a path quantifier before each one. μTL can be said to be the "assembly language of temporal logic", as most of programmable logics are expressible by it. This is a natural consequence of tha fact that μTL fully characterized the behavior of any finite-state process [Cle90].

The idea of μTL is to allow quantification over certain subsets of the universe U, which is not a first order notion. For reasons of increasing computational complexity, quantifying over any arbitrary subset is a not a practical concept, but the *fix-point* quantification of μTL works quite well. The grammar of μTL is defined as follows:

$$\mu\mathbf{TL} ::= \mathcal{P} \mid \mathcal{Q} \mid \bot \mid (\mu\mathbf{TL} \to \mu\mathbf{TL}) \mid \langle\mathcal{R}\rangle\, \mu\mathbf{TL} \mid \nu\mathcal{Q}\, \mu\mathbf{TL}.$$

In addition to the "diamond"-operator $\langle\mathcal{R}\rangle$, its dual, the "box" operator $[\mathcal{R}]\varphi = \neg\langle\mathcal{R}\rangle\neg\varphi$, is often used in μTL-formulas. The quantifier (i.e. *recursion operator*)ν is a *restricted existential quantifier* on sets of points. Another quantifier μ is defined by ν as $\mu q\, \varphi \triangleq \neg\nu q\, \neg(\varphi\{q := \neg q\})$, where $\varphi\{q := \psi\}$ denotes the formula obtained from φ by substituting ψ in place of every *free occurrence* of q. Free occurrences are such appearances of q that are not themselves under the scope of a quantifier. μ is interpreted as a *restricted universal second order quantifier*. A noteworthy equivalence in discussing μTL is that $\mathcal{M} \models (\varphi\, \mathbf{U}^+\psi) \leftrightarrow \mathcal{M} \models \mu q\, \mathbf{X}(\psi \vee (\varphi \wedge q))$.

Propositional μ-calculus has fairly fluent algorithms both for global and local model checking. After making the following definitions, the model checking algorithm of CTL presented in Section 1.1.2 can be used as a starting point for a global model checking algorithm for μTL. The correspondence is quite strong, as μ-calculus is interpreted not only on natural models, but also branching models.